1 Artificial Intelligence - The Concept

Syllabus

What is AI ? : The AI Problems, The Underlying Assumption, What is an AI Techniques, The Level Of The Model, Criteria For Success, Some General References, One Final Word.

Contents

1.1 The Concept of Artificial Intelligence (AI)

GTU : Winter-12,14,16,17,19, Summer-18,20

1.1.1 Introduction

- Many human mental activities such as developing computer programs, working out mathematics, engaging in commonsense 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 preceive, 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.

1.1.2 Various Definitions of AI

1. AI may be defined as the branch of computer science that is concerned with the automation of intelligent behaviour. (Luger - 1993)
2. Systems that thinks like human.
3. The exciting new effort to make computers think ... machines with minds, in the full and literal sense. (Hallgeland - 1985)
4. "The automation of activities that we associate with human thinking, activities such as devision making, problem solving, learning ..." (Bellman - 1978)
5. Systems that act like humans.
6. "The art of creating machines that perform functions that require intelligence, when performed by people". (Kurzweil - 1990)
7. "The study of how to make computers do things at which, at the moment, people are better". (Rich and Knight - 1991).
8. Systems that think rationally.
9. The study of mental faculties through the use of computational models. (Charniak and McDermott - 1985)

10. "The study of the computations that make it possible to perceive, reason and act". (Winston - 1992)
11. Systems that act rationally
12. "Computational intelligence is the study of the design of intelligent agents". (Poole et al - 1998)
13. "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 :

a) Natural language processing - To enable it to communicate successfully in English.

b) Knowledge representation to store what it knows, what it hears.

c) Automated reasoning to make use of stored information to answer questions being asked and to draw conclusions.

d) 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 interrogater 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".
- To pass total turing test in addition, computer will need following capabilities.

e) Computer vision to perceive objects.

f) 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 interdesciplinary 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 **agre-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 uncertainity to acheive best expected outcome.
- The laws of thought emphasis on correct inference which should be incorported in rational agent.

1.1.3 The Foundation of AI

- 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 date 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 develope date processing theories.
- **Phychology** provides strong concepts of how humans and animals think and act which helps AI for developing process of thinking and actions.

1.1.4 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.

1.1.4.1 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.

1.1.4.2 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.

1.1.5 What AI can do Today ?

1.1.5.1 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.

1.1.5.2 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.

1.1.5.3 Autonomous Control

- The ALVINN computer vision system was trained to stear 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.

1.1.5.4 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 dignosis can be made.

1.1.5.5 Logistic Planning

- In 1991 during the persion Gulf Crisis U.S. forces deployed a dynamic analysis and replanning tool name DART for automated logistics planning and scheduling for transportation.
- AI can provide techniques for making fast and accurate plans.

1.1.5.6 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.

1.1.5.7 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 became more integrated and AI has found common ground with other disciplines.

1.1.6 Human Vs Machine

1.1.6.1 Will Machine behave Exactly as Human ?

- Here are the considerable difference between human and machine.
  1) Machines do not have life, as they are mechanical. On the other hand, humans are made of flesh and blood; life is not mechanical for humans.
  2) Humans have feelings and emotions and they can express these emotions. Machines have no feelings and emotions. They just work as per the details fed into their mechanical brain.
  3) Human can do anything original and machines cannot.
  4) Humans have the capability to understand situations and behave accordingly. On the contrary, machines do not have this capability.
  5) While humans behave as per their consciousness, machines just perform as they are taught.
  6) Humans perform activities as per their own intelligence. On the contrary, machines only have an artificial intelligence.

1.1.6.2 Comparisons between Human and Machines

1) Brains are analogue ; machines are digital.
2) The brain uses content-addressable memory; In machine, information in memory is accessed by polling its precise memory address. This is known as byte-addressable memory.
3) The brain is a massively parallel machine ; machines are modular and serial.
4) Processing speed is not fixed in the brain; machine has fixed speed specification.
5) Brains short - term memory is not like RAM.

6) No hardware / software distinction can be made with respect to the brain or mind.

7) Synapses are far more complex than electrical logic gates.

8) Unlike machine, processing and memory management are performed by the same components in the brain.

9) The brain is a self - organizing system.

10) Brain have bodies, the brain is much, much digger than any [current] machine.

1.1.7 List of Expert Systems Influential in AI Field

1. MACSYMA - Advised the user on how to solve complex maths problems.
2. DENDRAL - Advised the user on how to interpret the output from a mass spectrograph.
3. CENTAUR, INTERNIST, PUFF, CASNET - Are all medical expert systems for various purposes.
4. DELTA - Locomotive engineering.
5. Drilling Advisor - Oilfield prospecting.
6. Exper Tax - Tax minimisation advice.
7. XSEL - Computer sales.
8. PROSPECTOR - Interpreted geological data as potential evidence for mineral deposits. (Duda, Hart, in 1976).
9. NAVEX - Monitored radar data and estimated the velocity and position of the space shuttle. (Marsh, 1984)
10. R1/XCON - Configured VAX computer systems on the basis of customer's needs. (Mc Dermott, 1980)
11. COOKER ADVISER - Provides repair advice with respect to canned soup sterilizing machines. (Texas Instruments, 1986)
12. VENTILATOR MANAGEMENT ASSISTANT - Scrutinised the data from hospital breathing - support machines, and provided accounts of the patient's conditions. (Fagan , 1978)
13. MYCIN - Diagnosed blood infections of the sort that might be contracted in hospital.
14. CROP ADVISOR - Developed by ICI to advise cereal grain farmers on appropriate fertilizers and pesticides for their farms.
15. OPTIMUM - AIV - is a planner used by the European Space Agency to help in the assembly, integration and verification of spacecraft.

1.2 AI Problem

GTU : Winter-12

- Much of the early work in AI focused on formal tasks, such as game playing and theorem proving. For example chess playing, logic theorist was an early attempt to prove mathematical theorems. Game playing and theorem proving share the property that people who do them well are considered to be displaying intelligence.
- Despite this it appeared that computers could perform well at those tasks by being fast at exploring a large number of solution paths and then selecting the best one. But no computer is fast enough to overcome the combinatorial explosion generated by most problems.
- AI focusing on the sort of problem solving we do every day for instance, when we decide to get to work in the morning, often called commonsense reasoning. In investigating this sort of reasoning Newell, Shaw, and Simon built the General Problem Solver (GPS), which they applied to several commonsense tasks as well performing symbolic manipulations of logical expression. However no attempt was made to create a program with a large amount of knowledge about a particular problem domain. Only quite simple tasks were selected.
- As AI research progressed and techniques for handling larger amounts of world knowledge were developed in dealing with problem solving in specialized domains such as medical diagnosis and chemical analysis.
- Perception (vision and speech) is another area for AI problems. Natural language understanding and problem solving in specialized domain are other areas related to AI problems. The problem of understanding spoken language is perceptual problem and is hard to solve from the fact that it is more analog related than digital related. Many people can perform one or may be more specialized tasks in which carefully acquired expertise is necessary. Examples of such as tasks include engineering design, scientific discovery, medical diagnosis, and financial planning. Programs that can solve problems in these domains also fall under the aegis of Artificial Intelligence.
- The tasks that are targets of works in AI can be categorized as follows :
  1. Mundane tasks - Perception (Vision and Speech), Natural language (Understanding, Generation, Translation, Commonsense reasoning, Robot control)
  2. Formal tasks - Games (Chess, etc.), Mathematics (Geometry, Logic, Integral calculus, etc.)
  3. Expert tasks - Engineering (Design, Fault finding, Manufacturing planning), Scientific analysis, Medical diagnosis, Financial analysis

- A person who knows how to perform tasks from several of the categories shown in above list learn the necessary skills in a standard order. First perceptual, linguistic, and commonsense skills are learned. Later expert skills such as engineering, medicine, or finance are acquired. Earlier skills are easier and thus more amenable to computerized duplication than the later, more specialized one. For this reason much of the initial work in AI work was concentrated in those early areas.
- The problems areas where now AI is flourishing most as a practical discipline are primarily the domains that require only specialized expertise without the assistance of commonsense knowledge. Expert systems (AI programs) now are up for day-to-day tasks that aim at solving part, or perhaps all, of practical, significant problem that previously required high human expertise.
- When one is building a expert system, following questions need to be considered before one can progress further :
- What are the underlying assumptions about intelligence ?
- What kinds of techniques will be useful for solving AI problems ?
- At what level if at all can human intelligence be modelled ?
- When will it be realised when an intelligent program has been built ?

1.3 The Underlying Assumption

- A physical symbol system consists of a set of entities called symbols which are patterns that can occur as components of another entity called an expression. At an instant the system will contain a collection of these symbol structures. In addition the system also contains a collection of processes that operate on expressions to produce other expressions ; processes of creation, modification, reproduction and destruction. A physical symbol system is a machine that produces through time an evolving collection of symbol structures. Such a system is machine that produces through time an evolving collection of symbol structures.

Following are the examples of physical systems -

- **Formal logic :** The symbols are words like "and", "or", "not", "for all x" and so on. The expressions are statements in formal logic which can be true or false. The processes are the rules of logical deduction.
- **Algebra :** The symbols are "+", "X", "x", "y", "1", "2", "3", etc. The expressions are equations. The processes are the rules of algebra, that allow you to manipulate a mathematical expression and retain its truth.
- **A digital computer :** The symbols are zeros and ones of computer memory, the processes are the operations of the CPU that change memory.

- **Chess :** The symbols are the pieces, the processes are the legal chess moves, the expressions are the positions of all the pieces on the board

The physical symbol system hypothesis claims that both of these are also examples of physical symbol systems. Intelligent human thoughts are the symbols that are encoded in our brains. The expressions are **thoughts.** The processes are the mental operations of thinking. In a running **artificial intelligence** program the symbols are data, the expressions are more data and the processes are programs that manipulate the data.

The importance of the physical symbol system hypothesis is twofold. It is significant theory of the nature of human intelligence and it forms the basis of the belief that it is possible to build programs that can perform intelligent tasks which are currently performed by people.

1.4 What is an AI Technique ?

Intelligence requires knowledge but knowledge possesses less desirable properties such as, 1. It is voluminous. 2. It is difficult to characterize accurately. 3. It is constantly changing. 4. It differs from data by being organised in a way that corresponds to its application.

An AI technique is a method that exploits knowledge that is represented so that the knowledge captures generalizations and situations that share properties which can be grouped together, rather than being allowed separate representation. It can be understood by people who must provide the knowledge; although for many programs the bulk of the data may come automatically, such as from readings.

In many AI domains people must supply the knowledge to programs in a form the people understand and in a form that is acceptable to the program. Knowledge can be easily modified to correct errors and reflect changes in real conditions. Knowledge can be widely used even if it is incomplete or inaccurate. Knowledge can be used to help overcome its own sheer bulk by helping to narrow the range of possibilities that must be usually considered.

Following are three important AI techniques -

- Search - Provides a way of solving problems for which no more direct approach is available.
- Use of knowledge - Provides a way of solving complex problems by exploiting the structures of the objects that are involved.
- Abstraction - Provides a way of separating important features and variations from the many unimportant ones that would otherwise overwhelm any process.

1.5 The Level of the Model

GTU : Winter-19

Before starting doing something, it is good idea to decide exactly what one is trying to do. One should ask following questions for self analysis : -

- What is the goal in trying to produce programs that do the tasks the same way people do ?
- Are we trying to produce programs that do the tasks the same way people do ? Or are we trying to produce programs that simply do the tasks in whatever way appears easiest ?

Efforts to build program that perform tasks the way people do can be divided into two classes. The first one are those that attempt to solve problems that do not really fit our definition of AI. i.e. problems that computer could easily solve. The second class attempt to model human performance are those that do things that fall more clearly within our definition of Ai tasks; they do things that are not trivial for the computer.

Reasons for modeling human performance for these kind of tasks :-

- To test psychological theories of human performance. E.g. PARRY program written for this reason, which exploited a model of human paranoid behaviour to simulate the conversational behaviour of a paranoid person.
- To enable computer to understand human reasoning. For example, for a computer to be able to read a news paper story and then answer question, such as "Why did Ravana lose the game ?"
- To enable people to understand computer reasoning. In many cases people are reluctant to rely on the output of computer unless they can understand how the machine arrived at its result.
- To exploit what knowledge we can collect from people.
- To ask for assistance from best performing people and ask them how to proceed in dealing with their tasks.

1.6 Criteria for Success

- One of the most important questions to answer in any scientific or engineering research project is "How will we know if we have succeeded ?". So how in AI we have to ask ourselves, how will we know if we have constructed a machine that is intelligent ? The question is hard as unanswerable question "What is Intelligence ?"
- To measure the progress we use proposed method known as Turing Test. Alan Turing suggested this method to determine whether the machine can think. To conduct this test, we need two people and the machine to be evaluated. One person act as interrogator, who is in a separate room from the computer and the other person. The interrogator can ask questions of either the person or computer by typing questions and received typed responses. However the interrogator knows them only as A and B and aims to determine which is the person and which is the machine. The goal of the machine is to fool the interrogator into

believing that it is the person. If the machine succeeds at this, then we will conclude that the machine can think.

1.7 Some General References

- **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 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 Winogard 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; Widro, 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.
- 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 ocurred 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-recogonition, etc.) cannot deliver perfectly reliable information about the environment. Hence reasoning and planning systems must be able to handle uncertainity.** AI has been draw in to much closer contact with other fields such as control theory and economics, that also deal with agents.

1.8 AI Terms

1.8.1 Agents and it's Environment

- An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.
- For example consider human as agent. Human has eyes, ears and other organs which are sensors. Hands, legs, mouth and other body part work as actuators.
- Lets consider another example of agent - Robot. A Robotic agent might have cameras, infrared rangefinders as sensors. Robot can have various motors for actuators.

More examples of agent

1. Agent : Software agent
   Sensors : Keystrokes, file contents and network packets.
   Actuator : Screen, writing files, network packet.
2. Agent : Internet shopping agent
   Sensors : HTML, DHTML, pages (text graphics script)
   Actuators : Forms, display to user, follow URL.

1.8.2 The AI Terminology

1) Percept

The term percept refers to the agent's perceptual inputs at any given instant.

Examples -

1) A human agent percepts "Bird flying in the sky "through eyes and takes its snap (photograph)".
2) A robotic agent perceive "Temperature of a boiler" through cameras and takes the control action.

2) Percept Sequence

An agent's percept sequence is the complete history of everything the agent has ever perceived. Agent has choice of action at any given instant and it can depend on the entire percept sequence agent has recorded. The change in the perception forms a historical case.

For example -

A robotic agent monitoring temperature of a boiler will be sensing it continuously and keep on maintaining the percept sequence. This percept sequence will help robotic agent to know how temperature fluctuates and action will be taken depending on percept sequence for controlling temperature.

3) Agent Function

It is defined as mathematical function which maps each and every possible percept sequence to a possible action.

This function has input as percept sequence and it gives output as action.

Agent function can be represented in a tabular form.

Example -

ATM machine is a agent, it display menu for withdrawing money, when ATM card is inserted. When provided with percept sequence (1) A transaction type and (2) PIN number, then only user **gets cash.**

4) Agent Program

When we want to develop a agent program we need to tabulate all the agent functions that describes any given agent. This can practically lead to infinite functions hence we need to put bound on the length of percept sequence that we need to consider. This table of functions of percept sequences and action will be external characteristics of the agent where as internally agent function for an intelligent agent will be implement by an agent program.

Note :

Agent function is an abstract mathematical description.

Agent program is a concrete implementation, running on the agent architecture.

1.8.3 Architecture of Agent

- The agent program runs on some sort of computing device, which is called the architecture. The program we choose has to be one that the architecture will accept and run. The architecture makes the percepts from the sensors available to the program, runs the program and feeds the program's action choices to the effectors as they are generated. The relationship among agents, architectures and programs can be summed up as follows :

  Agent = Architecture + Program

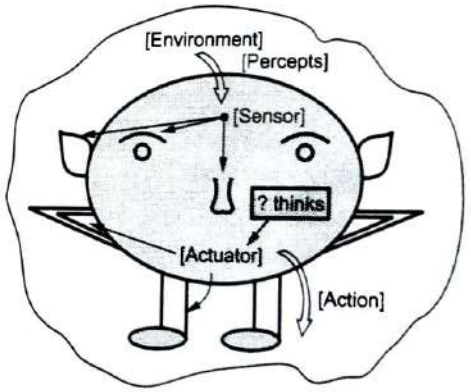


Fig. 1.8.1 Agent and its environment

1.8.4 Schematic of AI's Agent Performing Action

- Following diagram illustrates the agent's action process, as specified by architecture. This can be also termed as agent's structure.

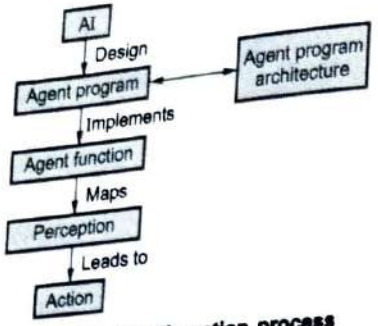


Fig. 1.8.2 Agent's action process

1.8.5 Role of An Agent Program

- An agent program is internally implemented as agent function.
- An agent program takes input as the current percept from the sensor and return an action to the effectors (Actuators).

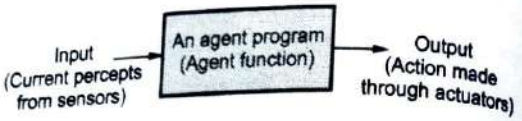


Fig. 1.8.3 Role of an agent program in agent architecture

1.8.6 Simple Example for Tabulation of a Agent

Agent - A shopping agent on internet called as bot.

Tabulation of percepts and action mapping -

Sr. No.	Sequence of Percepts	Actions
1.	[Type URL of greeting site mygreeting.com]	Display website.
2.	[Navigation and observation of greetings to be purchased]	Clicks on the link.
3.	[To get details of greeting (which is purchased), in terms of a form]	Form filling.
4.	[To perceive completion of process]	Receiving receipt or bill.

1.8.7 The Weak and Strong Agent

- An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors/actuators.

1.8.7.1 Weak Agent

- A weak notion says that an agent is a hardware or software based computer system that has the following properties :

1] Autonomy

Agents operate without direct intervention of humans and have control over their actions and internal state.

2] Social ability

Agents interact with other agents (and possibly humans) via an agent communication language.

3] Reactivity

Agents perceive their environment and respond in timely and rational fashion to changes that occur in it.

4] Pro - activeness

Agents do not simply act in response to their environment, they are capable of taking the initiative, generate their own goals and act to achieve them.

1.8.7.2 Strong Agent

A stronger nation says that an agent has mental properties, such as knowledge, belief, intention, obligation. In addition and agent has other properties such as : -

1. **Mobility :** Agents can move around from one machine to another and across different system architectures and platforms.
2. **Veracity :** Agents do not knowingly communicate false information.
3. **Rationality :** Agents will try to achieve their goals and not acts in such a way that would prevent their goals from being achieved.

Strong AI is associated with human traits such as consciousness, sentience, sapience, self-awareness

1. **Conciousness** - To have subjective experience and thought.
2. **Selfawareness** - To be aware of oneself as a separate individual, especially to be aware of one's own thoughts.
3. **Sentience** - The ability to feel perceptions and emotions subjectively.
4. **Sapience** - The capacity for wisdom.

1.8.8 Rational Behaviour and Omniscience

1.8.8.1 Rational Agent

- If every entry in the agent function is filled correctly then the agent will always do the right thing. Such agent is called as rational agent. Doing the right thing makes agent most successful. So now we need certain methods to measure the success of rational agent.
- When an agent is working in the environment, it generates a sequence of actions according to the percept it receives. This sequence of actions leads to various states of environment. If this sequences of environment state change is desirable, then we can say that agent has performed well. So if the tasks and environment change automatically the measuring conditions will change and hence there is no fixed measure suitable for all agents.
- As a general rule, it is better to design performance measures according to what one wants in the environment, rather than according to how one thinks the agent should behave.

The rationality depends upon 4 things -

1) The performance measure that defines the criterion of success.
2) The agent's prior knowledge about the environment.
3) The actions that the agent can perform.
4) The agent's percept sequence till current date.

Based on above 4 statements **rational agent can be defined as follows -**

For each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure given the evidence provided by the percept sequence and whatever built-in knowledge the agent has. Following figure depits performance measuremetric.

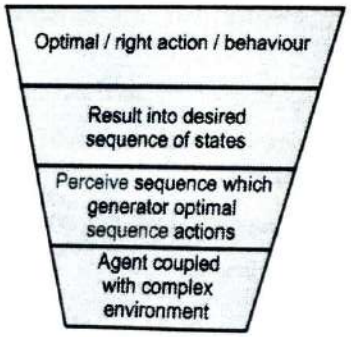


Fig. 1.8.4 Optimal performance triangle

1.8.8.2 The Good and the Bad Agent

- The concept of rational behaviour leads to two types agents, the good agents and the bad agent. Most of the time the good and bad behaviour (that is performance) of the agent depends completely on the environment.
- If environment is completely known then we get agent's good behaviour as depicted in Fig. 1.8.5.
- If environment is unknown then agent can act badly as depicted in Fig. 1.8.6.

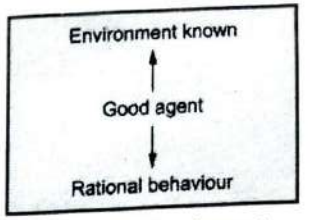


Fig. 1.8.5 Good agent

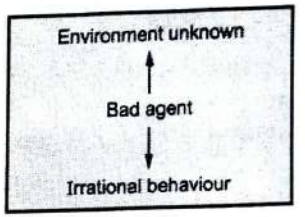


Fig. 1.8.6 Bad agent

1.8.8.3 Omniscience, Learning and Autonomy

- An omniscient agent knows the actual outcome of its actions and can act accordingly, but in reality omniscience is impossible.
- Rationality is not same as perfection. **Rationality maximizes expected performance where as perfection maximizes actual performance.**
- For increasing performance agent must do same actions in order to modify future percepts.
- This is called as information gathering which is important part of rationality. Also agent should explore (understand) environment to increase performance i.e. for doing more correct actions.
- Learning is another important activity agent should do so as to gather information. Agent may know environment completely (which is practically not possible) in certain cases but if it is not known agent needs learn on its own.
- To the extent that an agent relies on the prior knowledge of its designer rather than on its own percepts, we say that agent lacks autonomy. A rational agent should be autonomous - it should learn what it can do to compensate for partial or incorrect prior knowledge.

Figure Depicting rationality and Omniscience Relationship

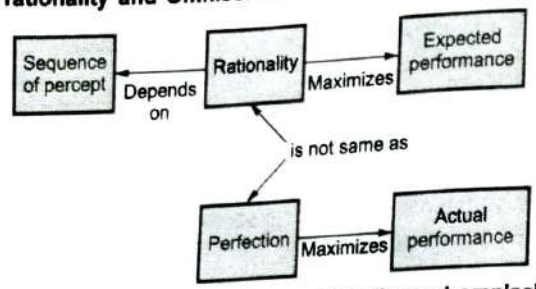


Fig. 1.8.7 The relationship between rationality and omniscience

1.8.9 Agent and it's Environment

1.8.9.1 Agent Description

Consider following example, A BLACK BALLS PICKER

The Picker World (Environment)

It is a simple and made-up world so one can invent many variations.

It has two buckets at two locations, L1 and L2 (for simplicity consider square area for location), full of BLACK and WHITE colour balls.

The Picker and its Perceptions

Picker perceives at which location it is. It can perceive that, is there a BLACK ball at the given location.

The Agent Actions

Picker can choose to MOVE LEFT or MOVE RIGHT, PICK UP BLACK BALL or be ideal that as do nothing.

A function can be devised as follows - if the current location bucket has more BLACK BALLS then PICK, otherwise MOVE to other square.

Diagram Depicting Black Ball Picker

Following is the partial tabulation of a simple agent function for the black ball picker.

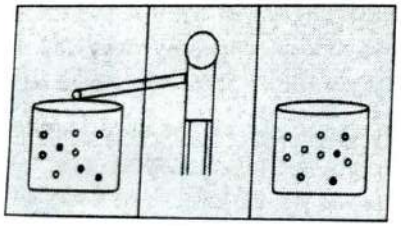


Fig. 1.8.8 Black ball picker world with two buckets at two locations

Percept Sequence	Action
[L1, No Black Ball]	Right
[L1, More Black Balls]	Pick
[L2, No Black Ball]	Right
[L2, More Black Balls]	Pick
:	
:	
[L1, No Black Ball], [L1, No Black Ball]	Right
[L1, No Black Ball], [L1, More Black Balls]	Pick
:	
:	
[L1 No Black Ball], [L1, No Black Ball],	
[L1, No Black Ball]	Right
[L1, No Black Ball], [L1, No Black Ball],	
[L1, More Black Balls]	Pick
:	
:	

1.9 The Environments

GTU : Winter-18,19, Summer-19

1.9.1 Nature of Environment

- In previous section we have seen various types of agents, now let us see the details of environment where in agent is going to work. A task environment is essentially a problem to which agent is a solution.
- The range of task environments that might arise in AI is obviously vast. We can, however, identify a fairly small number of dimensions along which task environments can be categorized. These dimensions determine, to a large extent, the appropriate agent design and the applicability of each of the principle families of techniques for agent implementation.

1.9.2 Types of Task Environment

1.9.2.1 Fully Observable Vs Partially Observable

- If an agent's sensors give it the access to the complete state of the environment at each point of time, then it is fully observable.
- In some environment, if there is noise or agent is with inaccurate sensors or may be some states of environment are missing then such environment is partially observable.

Example -

Fully Observable

The puzzle game environment is fully observable where agent can see all the aspects, that are surrounding it. That is agent can see all the squares of the puzzle game along with values (if any added) in them.

More examples -

1) Image analysis.

2) Tic - tac toe.

Partially Observable

The pocker game environment is partially observable. Game of pocker is a card game that shares betting rule ; and usually (but not always) hand rankings. In this game agent is not able to perceive other player's betting.

Also agent cannot see other player's card. It has to play with reference to its own cards and with current betting knowledge.

More examples -

1) Interactive Science Tutor.

2) Millitary Planning.

1.9.2.2 Deterministic Vs Stochastic

- If from current state of environment and the action, agent can deduce the next state of environment then, it is deterministic environment otherwise it is stochastic environment.
- If the environment is deterministic except for the actions of other agents, we say that the environment is **strategic**.

Examples -

Deterministic : In image analysis whatever is current percept of the image, agent can take next action or can process remaining part of image based on current knowledge. Finally it can produce all the detail aspects of the image.

Strategic : Agent playing tic-tac toe game is in strategic environment as from the current state agent decides next state action except for the action of other agents.

More examples -

1) Video analysis. 2) Trading agent.

Stochastic : Boat driving agent is in stochastic environment as the next driving does not based on current state. In fact it has to see the goal and from all current and previous percepts agent needs to take action.

More examples -

1) Car driving 2) Robot firing in crowd.

1.9.2.3 Episodic Vs Sequential

- In episodic environment agent's experience is divided into atomic episodes such that each episode consists of, the agent perceiving process and then performing single action. In this environment the choice of action depends only on the episode itself, previous episode does not affect current actions.
- In sequential environment on the other hand, the current decision could affect all future decision.
- Episodic environments are more simpler than sequential environments because the agent does not need to think ahead.

Example -

Episodic Environment : Agent finding defective part of assembled computer machine. Here agent will inspect current part and take action which does not depend on previous decisions (previously checked parts).

More Examples -

1) Blood testing for patient. 2) Card games.

Sequential Environment : A game of chess is sequential environment where agent takes action based on all previous decisions.

More examples -

1) Chess with a clock. 2) Refinery controller.

1.9.2.4 Static Vs Dynamic

- If the environment can change while agent is deliberating then we say the environment is dynamic for the agent, otherwise it is static.
- Static environments are easy to tackle as agent need not worry about changes around (as it will not change) while taking actions.
- Dynamic environments keep on changing continuously which makes agent to be more attentive to make decisions for act.
- If the environment itself does not change with time but the agent's performance does, then we say that environment is semidynamic.

Examples -

Static : In crossword puzzle game the environment that is values held in squares can only change by the action of agent.

More examples -

1) 8 queen puzzle. 2) Semidynamic.

Dynamic : Agent driving boat is in dynamic environment because the environment can change (A big wave can come, it can be more windy) without any action of agent.

More examples -

1) Car driving. 2) Tutor.

1.9.2.5 Discrete Vs Continuous

- In discrete environment the environment has fixed finite discrete states over the time and each state has associated percepts and action.
- Where as continuous environment is not stable at any given point of time and it changes randomly thereby making agent to learn continuously, so as to make decisions.

Example :

Discrete : A game of tic-tac toe depicts discrete environment where every state is stable and it associated percept and it is outcome of some action.

More examples -

1) 8 - queen puzzle. 2) Crossword puzzle.

Continuous : A boat driving environment is continuous where the state changes are continuous, and agent needs to perceive continuously.

More examples -

1) Part Picking Robot. 2) Flight Controller.

1.9.2.6 Single Agent Vs Multiagent

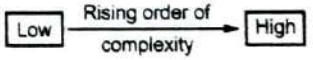
- In single agent environment we have well defined single agent who takes decision and acts.
- In multiagent environment there can be various agents or various group of agents which are working together to take decision and act. In multiagent environment we can have **competitive multiagent** environment, in which many agents are working parallel to miximize performance of individual or there can be **co-operative multiagent** environment, where in all agents have single goal and they work to get high performance of **all of** them together.

Example :

- Multiagent independent environment
  - ↳ Many agent in game of Maze.
- Multiagent cooperative environment
  - ↳ Fantasy football. [Here many agents work together to achieve same goal.]
- Multiagent competitive environment
  - ↳ Trading agents. [Here many agents are working but opposite to each other]
- Multiagent antagonistic environment
  - ↳ Wargames. [Here multiple agents are working opposite to each other but one side (agent/agent team) is having negative goal.]
- Single agent environment
  - ↳ Boat driving[Here single agent perceives and acts]

1.9.2.7 Complexity Comparison of Task Environment

Following is the rising order of complexity of various task environment.



Observable	→	Partially observable
Deterministic	→	Stochastic
Episodic	→	Sequential
Static	→	Dynamic
Discrete	→	Continuous
Single agent	→	Multiple agents.

1.9.3 More Types of Task Environment

Based on specific problem domains we can further classify task environments as follow.

1) Monitoring and Surveillance Environment

Example : Agent monitoring incoming people at some gathering where only authorized people are allowed.

2) Time Constrained Environment

Example : Chess with a clock environment where the move should be done in specified amount of time.

3) Decision Making Environment

Example : The executive agent who is monitoring profit of a organization, can help top level management to take decision.

4) Process Based Environment

Example : The image processing agent who can take input and synthesize it to produce required output, and details about the image.

5) Personal or User Environment

Example : A small scale agent which can be used as personal assistance who can help to remember daily task, who can give notifications about work etc.

6) Buying Environment

Example : A online book shopping bot (agent) who buys book online as per user requirements.

7) Automated Task Environment

Example : A cadburry manufacturing firm can use a agent who automates complete procedure of cadburry making.

8) Industrial Task Environment

Example : An agent developed to make architecture of a building or layout of building.

9) Learning Task Environment (Educational)

Example : We can have a agent who is learning some act or some theories presented to it and later it can play it back which will be helpful for others to learn that act or theories.

10) Problem Solving Environment

Example : We can have agent who solve different types of problems from mathematics or statistics or any general purpose problem like travelling salesman problem.

11) Scientific and Engineering Task Environment

Example : Agent doing scientific calculations for aeronautics purpose or agent develop to design road maps or over bridge structure.

12) Biological Task Environment

Example : Agent working for design of some chemical component helpful for medicine.

13) Space Task Environment

Example : Agent that is working in space for observing space environment and recording details about it.

14) Research Task Environment

Example : Agent working in a research lab where it is made to grasp (learn) knowledge and represent it and drawing conclusions from it, which will helps researcher for further study.

15) Network Task Environment

Example : An agent developed to automatically carry data over a computer network based on certain conditions like time limit or data size limit in same network (same type of agent can be developed for physically transferring items or mails) over same network.

16) Repository Task Environment

Example : If a data repository is to be maintained then agent can be developed to arrange data based on criterias which will be helpful for searching later on.

1.10 Different Types of Agents

1.10.1 Intelligent Agent

"Intelligent agent is an intelligent actor, who observe and act upon an environment".

Intelligent agent is magnum - opus.

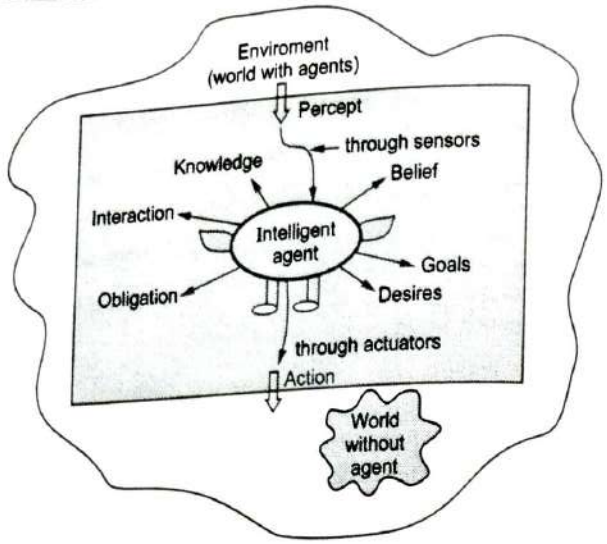


Fig. 1.10.1 Intelligent agent

The term 'Intelligent thinker' is different from intelligent agent. Fig. 1.10.2 shows intelligent agent's behaviour.

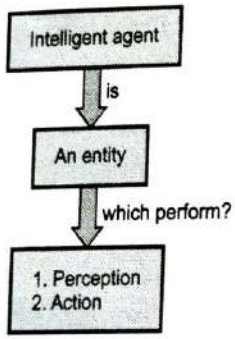


Fig. 1.10.2 Intelligent Agent

Example :

1) A robotic agent (Cameras, Infrared range finders).
2) An embedded real time software system agent.
3) A human agent (Eyes, ears and other organ).

Characteristics of Intelligent Agent (IA)

1) The IA must learn and improve through interaction with the environment.
2) The IA must adapt online and in the real time situation.

3) The IA must learn quickly from large amounts of data.
4) The IA must accommodate new problem solving rules incremently.
5) The IA must have memory which must exhibit storage and retrival capacities.
6) The IA should be able to analyze self in terms of behaviour, error and success.

1.10.2 Different Forms of Agents : (Types of Agents)

In artificial intelligence, there are different forms of intelligent agent and sub-agents.

As the degree of perceived intelligence and capability varies, it is possible to frame agent's into four categories.

1. Simple reflex agents.
2. Model based reflex agents.
3. Goal based agents.
4. Utility based agents.

In the following section we discuss each type of agent in detail.

1.10.2.1 Agent Type 1

Simple Reflex Agent

These agents select actions on the basis of the current percept, ignoring the rest of percept history.

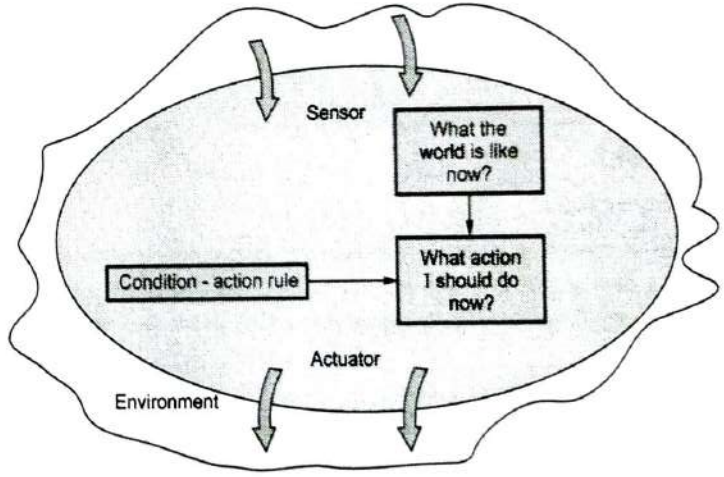


Fig. 1.10.3 Simple reflex agent

Property :

1) These are very simple but their intelligence is limited.
2) They will work only if correct decision can be made on the basis of only the current percept- that is only if the environment is fully observable.
3) A little bit of unobservability can cause serious trouble.
4) If simple reflex agent works in partially observable environment then, it can lead to infinite loops.
5) Infinite loops can be avoided if simplex reflex agent can try out possible actions i.e can randomize the actions.
6) A randomize simple reflex agent will perform better than deterministic reflex agent.

Example :

In ATM agent system if PIN matches with given account number then customer gets money.

Procedure : SIMPLE - REFLEX - AGENT

Input : Percept

Output : An action.

Static : Rules, a set of condition - action rules.

1. State ← INTERPRET - INPUT (percept)
2. rule ← RULE - MATCH (state, rules)
3. action ← RULE - ACTION (rule)
4. return action.

1.10.2.2 Agent Type 2

Model Based Reflex Agent

Internal state of the agent stores current state of environment which describes part of unseen world i.e how world evolves, and effect of agent's own actions. It means that it stores model of possibilities around it. Hence it is called as model based reflex agent.

Property :

1) It has ability to handle partially observable environments.
2) Its internal state is updated continuously which can be shown as :

Old - Internal state + Current percept = Update state.

For example :

A car driving agent which maintains its own internal state and then take action as environment appears to it.

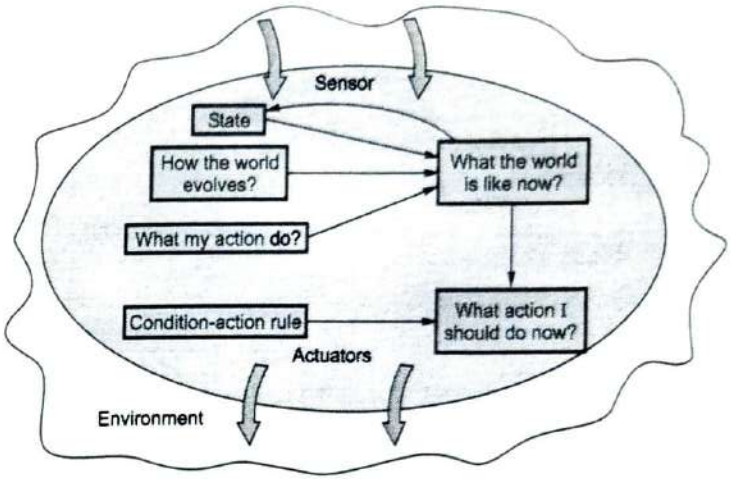


Fig. 1.10.4 Model - based reflex agent

Procedure : REFLEX-AGENT-WITH-STATE

Input : Percept

Output : An action.

Static : State, a description of the current world state, rules, a set of condition-action rules, action, the most recent action, initially none.

1. State ← UPDATE-STATE (state, action, percept)
2. Rule ← RULE-MATCH (state, rules)
3. Action ← RULE-ACTION (rule)
4. return action.

1.10.2.3 Agent Type 3

Goal Based Agent

Goal based agent stores state description as well as it stores goal states information.

Property

1) Goal based agent works simply towards achieving goal.
2) For tricky goals it needs searching and planning.

3) They are dynamic in nature because the information description appears in proper and explicit manner.
4) We can quickly change goal based agent's behaviour for new/unknown goal.

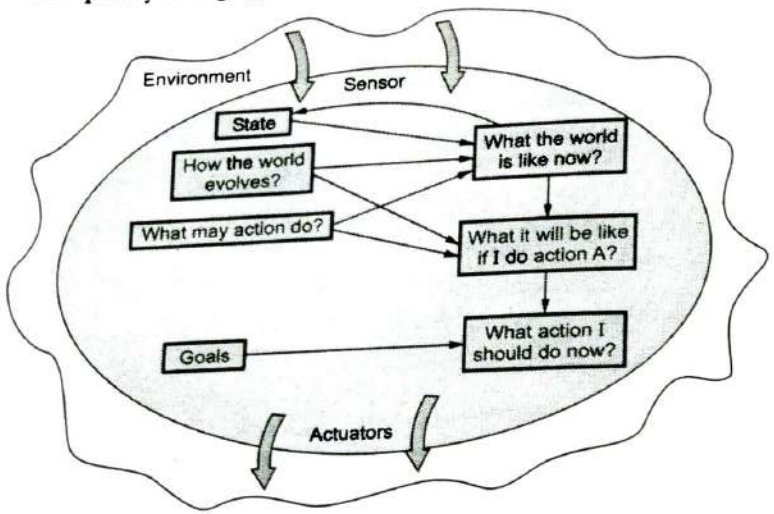


Fig. 1.10.5 Goal-based agent

For example :

Agent searching a solution for 8-queen puzzle.

1.10.2.4 Agent Type 4

Utility Based Agent

In complex environment only goals are not enough for agent designs. Additional to this we can have utility function.

Property :

1) Utility function maps a state on to a real number, which describes the associated degree of best performance.
2) Goals gives us only two outcomes achieved or not achieved. But utility based agents provide a way in which the likelihood of success can be measured against importance of the goals.
3) Rational agent which is utility based can maximize expected value of utility function i.e more perfection can be achieved.

4) Goals gives only two discrete states
   a) Happy b) Unhappy

For example -

Millitary planning robot which provides certain plan of action to be taken. Its environment is too complex, and expected performance is also high.

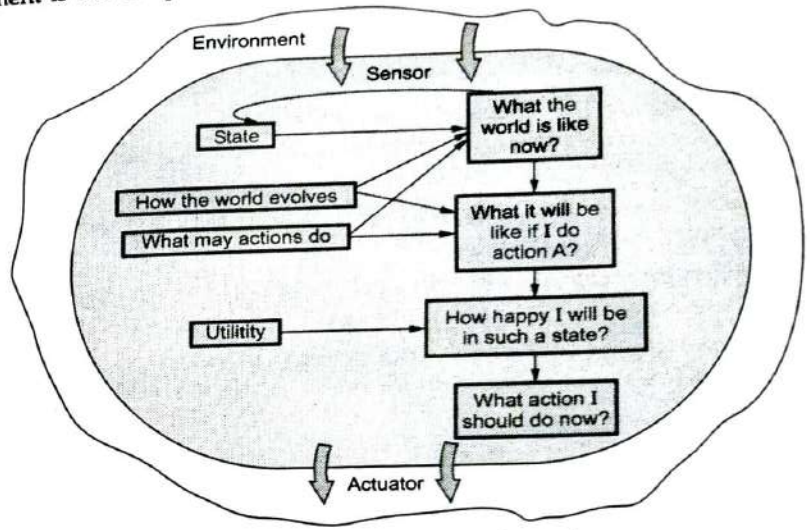


Fig. 1.10.6 Utility - based agent

1.10.3 The Learning Agent

If agent is to operate initially in unknown environments then agent should be self-learner. It should observe and gain and store information. Learning agent can be divided into 4 conceptual components

1) Learning Element - Which is responsible for making improvements.
2) Performance Elements - Which is responsible for selecting external actions.
3) Critic - It tells how agent is doing and determines how the performance element should be modified to do better in the future.
4) Problem Generator - It is responsible for suggesting actions that will lead to new and informative experiences to agent. Agent can ask problem generator for suggestions.

The performance standards distinguishes part of the incoming percept as a reward (success) or penalty (failure) that provides direct feedback on the quality of the agent's behaviour.

All four types agent we have seen can improve their performance through learning and there by become learning agents.

For example :

Aeroplane driving agent which continuously learns from environment and then do safe plane driving.

1.10.3.1 Components of Learning Agent

1) Base/Learner/Learning element - It holds basic knowledge and learn new things from the unfamiliar environment.
2) Capable/Efficient system/Performing elements - Capable system is responsible for selecting external actions. Performance element is the actual agent. It perceives and decides actions.
3) Fault reflector element - It gives feedback. It reflects fault and analyze corrective actions in order to get maximum success.
4) New problem generator element - It generate new and informative experience. It suggests new actions.

The performance standard makes difference between incoming percept as a reward (or penalty), that indicate direct feedback on the quality of the agent's behaviour.

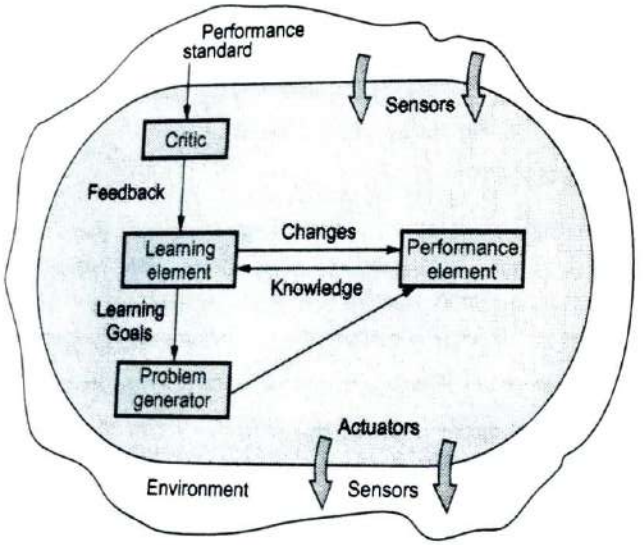


Fig. 1.10.7 Learning agent

1.10.4 More Types of Agents

We can do classification of agents based on various aspects like -

1) Task they perform. 2) Their various control architecture.
3) Depending on sensitivity of their sensors, and effectiveness of their action and internal states they possess.

Following are various types of agents, based on above classification criteria :-

1. **Physical Agents :** A physical agent is an entity which perceives through sensors and acts through actuators.
2. **Temporal Agents -** A temporal agent may use time based stored information to offer instructions or data acts to a computer program or human being and takes program inputs percepts to adjust its next behaviour.
3. **Spatial Agents -** That relate to the physical real-world.
4. **Processing Agents -** That solve a problem like speech recognition.
5. **Input Agents -** That process and make sense of sensor inputs- e.g. neural network based agents.
6. **Decision Agents -** That are geared upto do decision making.
7. **Believable Agents -** An agent exhibiting a personality via the use of an artificial character (the agent is embedded) for the interaction.
8. **Computational Agents -** That can do some complex, lengthy scientific computations as per problem requirements.
9. **Information Gathering Agents -** Who can collect (perceive) and store data.
10. **Entertaining Agents -** Who can perform something which can entertain human like gaming agents.
11. **Biological Agents -** Their reasoning engine works almost identical to human brain.
12. **World Agents -** That incorporate a combination of all the other classes of agents to allow autonomous behaviours.
13. **Life Like Agents -** Which are combinations of other classes of agents which will behave like real world characters. (For example - A robotic dog)

1.11 Designing an Agent System

When we are specifying agents we need to specify performance measure, the environment and the agent's sensors and actuators. We group all these under the heading of the task environment.

For the acronymically we call this PEAS ([P]erformance, [E]nvironment, [A]ctuators, [S]ensors) description.

1.11.1 The Steps in Designing an Agent

1) Define problem area (i.e. task environment) in complete manner. Example-Vaccum world, automated face recognition, automated taxi driver.
2) Define or tabulate PEAS.
3) Define or tabulate agent functions (i.e. percept sequence and action column)
4) Design agent program.
5) Design an architecture to implement agent program.
6) Implement an agent program.

The agent system may be single agent or multiple agents system.

If system is multiagents then we need to consider communication, co-operation strategies among multiple agents.

1.11.2 Examples of Agent Types and Their PEAS Description According to Their Uses

I) General Purpose (uses for common man)

Sr. No.	Agent Type	Performance Measure	Environment	Actuators	Sensors
1.	An automated taxi driver	Safe, fast, legal, comforatable trip, maximize profits.	Roads, other traffic, pedestrians, customers.	Steering acceleration break, Signal, horn, display.	Cameras, sonar, speedometer, GPS, Odometa, accele rometer engine, Sensors, keyboard.
2.	An automated face recognizer	Correct, recognition efficient system.	Human face software, web camera/video camera, infrared light.	Capturing face, feature extraction, classification.	Web/video camera, keyboard, mouse, infrared light.
3.	Part-picking robot	Percentage of parts in correct bins.	Conveyor belt with parts ; bins.	Jointed arm and hand.	Camera, joint angle sensors.
4.	ATM system	Secure, reliable fast service.	ATM machine, human system (customer).	Display menu/screen with options, validity checks.	Touch screen.

II) Industrial Business Purpose :

Sr. No.	Agent Type	Performance Measure	Environment	Actuators	Sensors
1.	E-commerce system	Secure reliable, fast business processing	E-commerce websites, human system. (customer).	Display product lists with price, forms.	Keyboard, mouse.
2.	Refinery controller	Maximize, purity, yield, safety.	Refinery, operators.	Values, pumps, heaters, displays.	Temperature, pressure, chemical sensors.

III) Scientific / Research Purpose

Sr. No.	Agent Type	Performance Measure	Environment	Actuators	Sensors
1.	Satellite image analysis system.	Correct image categorization	Downlink from orbiting satellite.	Display categorization of scene.	Color pixel arrays.
2.	Chemical reaction analyzer in chemistry research lab.	Correct recording of reaction.	A chemistry lab where instruments, chemicals are available for carrying out reactions.	Recording result of reaction.	Knowledge database of chemicals and their characteristics.

IV) Medical Purpose

Sr. No.	Agent Type	Performance Measure	Environment	Actuators	Sensors
1.	Medical diagnosis system.	Healthy patient, minimize costs, lawsuits	Patient, hospital, staff.	Display questions, tests, diagnoses, treatments, referrals.	Keyboard entry of symptoms, findings, patient's answers.
2.	Blood testing system.	Correct reporting on each test.	Blood sample lab.	Detail reporting of each test with specified components.	Database of procedures of test conduction and results.

V) Educational Purpose

Sr. No.	Agent Type	Performance Measure	Environment	Actuators	Sensors
1.	Interactive English tutor.	Maximize student's score on test.	Set of students, testing agency.	Display exercises suggestions, corrections.	Keyboard entry.
2.	A casio teacher.	Learner should be able to play specific musical pieces.	Group of learner or a single learner.	Display of each note, presentation of playing a key, sample music pieces.	Inputs from learner, from mouse or keyboard and database of casio details.

1.11.3 The Detail Example of PEAS

Agent : Interactive English Tutor

I) The [P]erformance Measures :

The Interactive English Tutor agent system must achieve the following performance measures.

1) All the student must get maximum knowledge regarding English subject, such as vocabulary, verbal soft skills, (i.e. communicational skill), reading, writing skills.
2) All the students must score good marks in the english test.

II) The [E]nvironment :

In Interactive English Tutor agent system environment has following properties :-

1) All the students having different grasping power and IQ (Intellectual Quotient).
2) Software modules which gives demonstration.

III) The [A]ctuators (Actions) :

The software model (agent program) will be executed on the agent architecture. (i.e. operating system). The actions performed by interactive english tutor are,

1) Audio / video demonstration on different topics.
2) Practical assignment on verbal written skills, report generation, letter writing, etc.
3) Monitoring and inspection (i.e. checking) of the practical assignment provided with suggestions and corrections, to students.
4) Online test conduction and result analysis.
5) Student's speech and video recording.

IV) The [S]ensors :

Sensor plays a crucial role in interactive English Tutor agent system. The following sensor are required to support sequence of perception :-

1) Keyboard for providing input events.
2) Mouse for GUI interface.
3) Headphone for listening and mike for audio recording.
4) Video/web camera's for video shooting.

1.12 One Final Word

- After taking a brief tour of AI history and its related work it can be seen that goal of AI is to construct working programs that solve the problems which are useful for well being of human.
- In AI major issue is to acquire large and enough amount of data and processed knowledge that can deal with almost all the problems and at least solve the toy problems. It becomes harder to access appropriate things when required, once the amount of knowledge grows up.
- A good programming language is required to process knowledge related to AI problems. LISP has been most commonly used language for AI programming. Specifically, AI programs are easiest to build using languages that have been designed to support symbolic rather than primarily numeric computation.
- AI is still a yet to bloom and a bud in industry. In our syllabus we are going to study some of the basic but major topics related to AI.

Answer in Brief

1. *Define AI.* **(Refer section 1.1)**
2. *What is AI ?* **(Refer section 1.1)**
3. *What is meant by robotic agent ?* **(Refer section 1.1)**
4. *What are advantages one can infer when machines perform intelligently ?* **(Refer section 1.1)**
5. *Define an agent.* **(Refer section 1.8)**
6. *What is role of an agent program ?* **(Refer section 1.8)**
7. *Define rotational agent.* **(Refer section 1.8)**
8. *List down the characteristics of intelligent agent.* **(Refer section 1.10)**
9. *Give general model of learning agent.* **(Refer section 1.10)**
10. *Explain in detail the history of AI.* **(Refer section 1.1)**
11. *What are various domains of AI ?* **(Refer section 1.1)**
12. *Discuss in detail the structure of agent with suitable diagram.* **(Refer section 1.8)**

13. *What is an ideal rational agent ?* **(Refer section 1.8)**
14. *Explain properties of environment.* **(Refer section 1.9)**
15. *Name at least 5 agent types with percepts actions and goals with environment.* **(Refer section 1.9)**
16. *What are requirements of intelligent agents ?* **(Refer section 1.10)**
17. *Discuss model based agents and goals based agents.* **(Refer section 1.10)**
18. *Give the structure of an agent with goals.* **(Refer section 1.10)**
19. *List few agent types and describe their PEAS.* **(Refer section 1.11)**
20. *What is meant by PEAS ?* **(Refer section 1.11)**
21. *What is AI ? Explain how an AI system is different from a convolutional computing system.* **(Refer section 1.1)**
22. *What is AI ? State various characteristics of AI.* **(Refer section 1.1)**
23. *Explain the nature and scope of AI. Why game playing problems are considered AI problems ?* **(Refer section 1.1)**
24. *What are AI techniques ?* **(Refer section 1.4)**
25. *Define AI and justify with suitable example how does conventional computing different from the intelligent computing.* **(Refer section 1.1)**
26. *Explain desirable properties of AI internal representation and AI software.* **(Refer section 1.1)**

1.13 University Questions with Answers

Winter - 12

Q.1 *What is intelligence ? Discuss types of problems requiring intelligence to solve it. Define AI.* **(Refer sections 1.1.2 and 1.2)** [7]

Winter - 14

Q.2 *Define AI ? Explain the characteristics of AI problem.* **(Refer section 1.1.2)** [7]

Winter - 16

Q.3 *Discuss following : i) Turing test* **(Refer section 1.1)** [3]

Winter - 17

Q.4 *Discuss : Turning test.* **(Refer section 1.1)** [3]

Summer - 18

Q.5 *Discuss Turning test.* **(Refer section 1.1)** [4]

Winter - 18

Q.6 *Define and discuss different task domain of artificial intelligence.* **(Refer section 1.9)** [3]

Summer - 19

Q.7 *Define the following words in the context of AI : Intelligence* **(Refer section 1.9)** [4]

Winter - 19

Q.8 *Define the term "Artificial Intelligence". Explain how AI techniques improve real-world problem solving.* **(Refer sections 1.1 and 1.4)** [3]

Q.9 *What is the significance of the "Turing Test" in AI ? Explain how it is performed.* **(Refer section 1.1)** [4]

Q.10 *Enlist and discuss the major task domains of Artificial Intelligence.* **(Refer section 1.9)** [7]

Summer - 20

Q.11 *Define the following words in the context of AI :*
i) Intelligence. **(Refer section 1.1)** [4]

❑❑❑

2 Problems and State Space Search

Syllabus

Problems, State Space Search & Heuristic Search Techniques : *Defining The Problems As A State Space Search, Production Systems, Production Characteristics, Production System Characteristics, And Issues In The Design Of Search Programs, Additional Problems.*

Contents

2.1 Problem Defining and Solving Problem GTU : Summer-13,15, Winter-15

2.1.1 What is Problem Solving ?

- For solving any type problem (task) in real world one needs formal description of the problem.
- One should have clear undenstanding 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.

2.1.2 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 a 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)**.

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

- The most common formulation uses a successor funtion. Given a perticular state x, SUCCESSOR Function (X) returns a set of <action, successor>, 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.

{
< Go (Mumbai), In (Mumbai) >
< Go (AhemdNagar), In (AhemdNagar)>
< 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 is 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 explicitely 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 not 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 requireded 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.

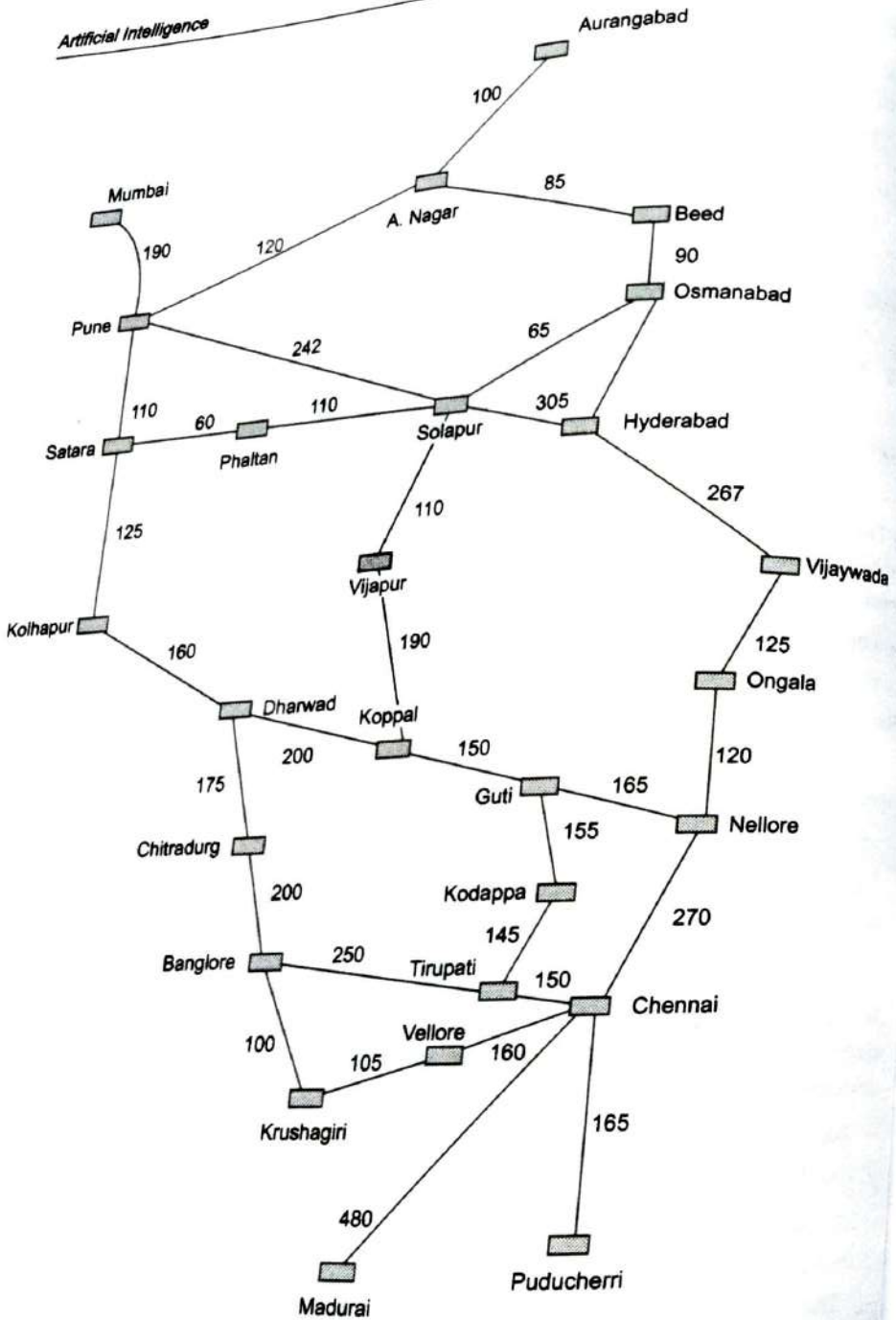


Fig. 2.1.1 Indian traveller map

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.

2.1.3 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.

1) 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 64.63 ... 57 $= 3 \times 10^{14}$ possible sequences to investigate.

2) 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.

2.1.4 Solving the Problem

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

1) Problem definition : Where in detailed specification of inputs and what constitutes an acceptable solution is described.

2) Problem analysis : Where in problem is studied through various view points like inputs, to the problem, environment of the problem, expected outputs.

3) Knowledge representation : Where in the known data about the problem and various expected stimuli from environment is represented in perticular format which is helpful for taking actions.

4) Problem solving : Where in the selection of best suited techniques for problem solutions are thought of and finalized.

2.1.5 Problem Solving Agents

2.1.5.1 Approach of Problem Solving Agent

- Goal based agents are also called as problem solving agent.
- Problem solving agent adapt to the task environment understand goal and achieve success -

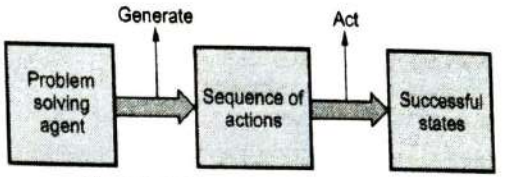


Fig. 2.1.2 Steps in problem solving

- 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.

2.1.5.2 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. 2.1.3 Problem solving agent

2.1.5.3 Algorithm

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

Results : An action.

Input : P → percept (Environment perception)

Static :

1) A → An action sequence, initially with null value.

2) S → State - current state.

3) G → Goal - A goal initially null.

4) P → Problem - A real world situation.

State - update state (State, percept)

If (s) is empty then do

g ← Formulate goal (s)

P ← Formulate problem (s, g)

S ← Search (p)

G ← First (s)

S ← Rest (s)

Return a

Procedure

2.1.5.4 For Example

Consider following simple problem solving agent. Working in open-loop system. Open-loop system means agent is assumed to be working in following environment -

1) Static environment : Where in problem formulation and solution is done by ignoring the changes that can occur in environment.

2) Observable environment : Where in agent has complete knowledge of environment.

3) Discrete environment : Where in the idea of enumerating "alternative courses of actions" is implemented.

4) Deterministic environment : Where in next state is configured from current state.

- **Points to Note**

1) Above kind of working systems is called as open-loop system, because ignoring the percepts breaks loop between agent and environment.
2) In open-loop systems solutions to problem are single sequence of actions, so they cannot handle any unexpected events.
3) Also solutions are executed without paying attention to the percepts.
4) These are most easiest kind of environment to work in for agents.

2.2 State Space Search

GTU : Winter-16, 17, 19, Summer-12, 17, 18

- For finding the solution one can make use of explicit search tree that is generated by the initial state and the successor function that together define the state space. In general, we may have search graph rather than a search tree as the same state can be reached from multiple paths.

2.2.1 Construction of State Space

1) The root of search tree is a search node corresponding to initial state. In this state only we can check if goal is reached.
2) If goal is not reached we need to consider another state. Such a can be done by expanding from the current state by applying successor function which generates new state. From this we may get multiple states.
3) For each one of these, again we need to check goal test or else repeat expansion of each state.
4) The choice of which state to expand is determined by the search strategy.
5) It is possible that some state, surely, can never lead to goal state. Such a state we need not to expand. This decision is based on various conditions of the problem.

2.2.2 Terminology used in Search Trees

1) Node in a tree : It is a book keeping data structure to represent the structure configuration of a state in a search tree.

2) State : It reflects world configuration. It is mapping of state and action to another new state.

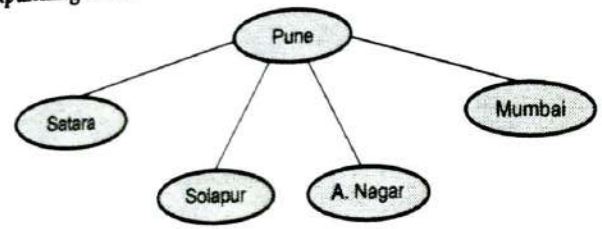
3) Fringe : It is a collection of nodes that have been generated but not yet expanded.

4) Leaf node : Each node in fringe is leaf node (as it does not have further successor node).

For example :

a) The initial state

b) After expanding Pune



c) After expanding Satara

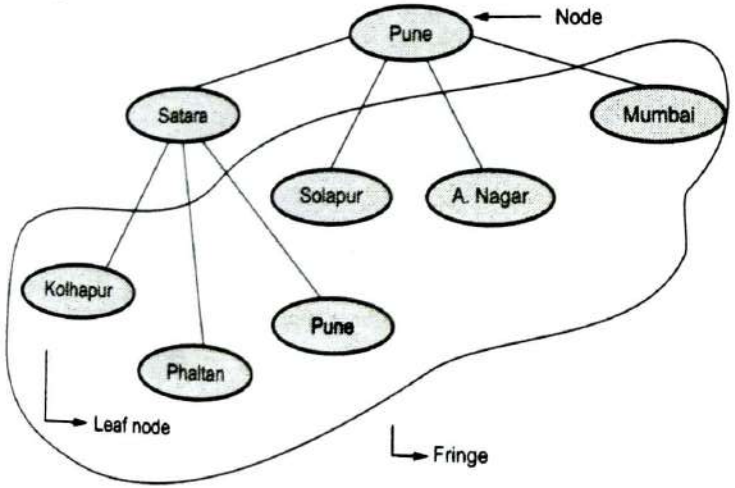


Fig. 2.2.1 Partial search trees for finding route from pune to chennai

5) Search strategy : It is a function that selects the next node to be expanded from current fringe. Strategy looks for best node for further expansion.

For finding best node each node needs to be examined. If fringe has many nodes then it would be computationally expensive.

The collection of un-expanded nodes (fringe) is implemented as queue, provided with, the sets of operations to work with queue. These operations can be CREATE Queue, INSERT in Queue, DELETE from Queue and all necessary operations which can be used for general tree-search algorithm.

2.2.3 Node Representation in a Search-Tree

Formally we can represent the node of search tree with 5 components,

1) State : The state, in the state space to which the node corresponds;

2) Parent-node : The node in the search tree that generated this node;

3) Action : The action that was applied to the parent to generate the node;

4) Path-cost : The cost, traditionally denoted by function g(n), of the path, from the initial state to the node, as indicated by the parent pointers;

5) Depth : The number of steps along the path from the initial state.

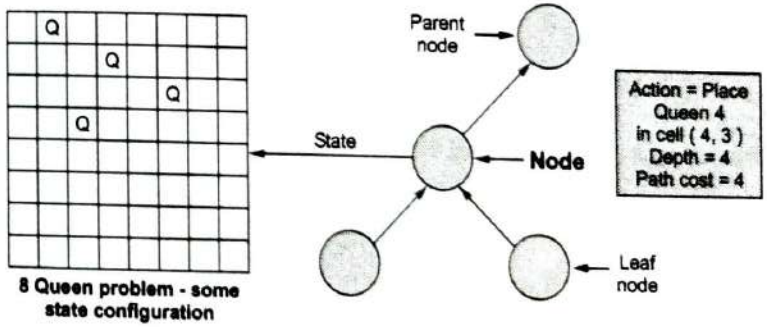


Fig. 2.2.2 Node in search tree

2.2.4 The Node Searching and Expansion Algorithms

2.2.4.1 Algorithm Tree Search

Input - Problem, fringe.

Output - Solution or failure.

1) Create initial node from problem's initial state.

2) Create fringe from initial node.

3) If fringe is empty return failure.

4) Node = first_node from_fringe.

5) If **Goal** test succeeds on node then return solution (node).

6) Expand node and add the newly generated nodes to fringe.

7) Repeat step (3) to (6).

2.2.4.2 Algorithm Expand Node

/* This algorithm will be used in Tree search for expanding the node */

Input - Node, problem

Output - A set of nodes (newly generated) (successor)

1) Initial successor set is empty.

2) For each <action, result>,
in successor function of a problem for a given state do
steps (3) to (9).

3) s = a new node.

4) STATE [s] = result.

5) Parent Node [s] = node.

6) Action [s] = action.

7) Path Cost [s] = PathCost [node] + StepCost (node, action s).

8) Depth [s] = Depth [node] + 1.

9) Add s to successor set.

10) Return successor set.

2.2.5 Measuring Problem Solving Performance

When we are solving a problem we have three possible outcomes 1) We reach at failure state 2) Solution state 3) Algorithm might get stuck in an infinite loop.

Problem solving algorithm's performance can be evaluated on the basis 4 factors -

1) Completeness : Does the algorithm surely finds a solution, if really the solution exists.

2) Optimality : Some times it happens that there are multiple solutions to a single problem. But the algorithm is expected to produce best solution among all feasible solution, which is called as optimal solution.

3) Time complexity : How much time the algorithm takes to find the solution.

4) Space complexity : How much memory is required to perform the search algorithm.

- **Major factors affecting time complexity and space complexity -**

Time and space complexity are majorly affected by the size of the state space graph, because it is the input to the algorithm. State space graph needs to be stored as well as it needs to be processed. So it is straight forward that more complex state space graph more is the space and time required.

- **The state space graphs complexity is affected by 3 factors -**

a) Branching factor : It is maximum number of successors of any node. If this value is less then less nodes will have to be search, there by getting result fast.

b) Depth of goal node : It is the depth of the shallowest goal node, where one can reach fast. If this value is small we will get goal node early.

c) Maximum length of path : It is maximum length of any path in the state space. If length value is more, complexity of state space is more. This is often measured in terms of number of nodes generated.

The major activity in searching is node expansion. The time taken for this is called as search cost. Search cost will typically depend on time complexity.

- **Path cost :** Is time bound cost which is incurred for reaching or going to a perticular node.

The total cost for algorithm is the combined cost of search cost and the path cost of the solution found along with memory usage.

Total cost = Search cost + Path cost + Memory usage.

For the problem of finding a route from Pune to Chennai, the search cost is amount of time taken by the search and the solution cost is the total length of the path.

The cost found will be helpfull for agent to find shorter paths (low cost paths).

2.3 State Space Search Strategies

GTU : Winter-12, 14, 17, 18, 19, Summer-12, 13, 14, 16, 18, 19

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.

2.3.1 Two Basic 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.*

2.3.2 Uninformed Search Strategies

1. B.F.S.
2. D.F.S.
3. Depth limited search
4. Iterative deepening DFS
5. Bidirectional search
6. 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 criterias 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.

2.3.2.1 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 levelwise starting at root level.

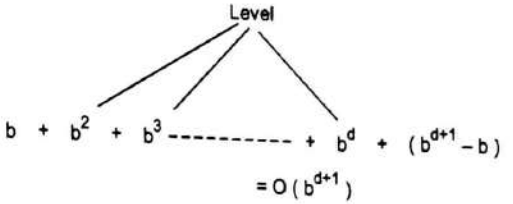
The implementation : BFS can be implemented using first, in first out queue data structure where fringe will be stored and processed. As soon as node is visited it is added to queue. All newly generated nodes are added to the end of the queue, which means that shallow nodes are expanded before deeper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



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] = o) then BFS (i) ;
}
```

BFS

- Time complexity - $O(b^{d+1})$
- Space complexity - $O(b^{d+1})$

BFS

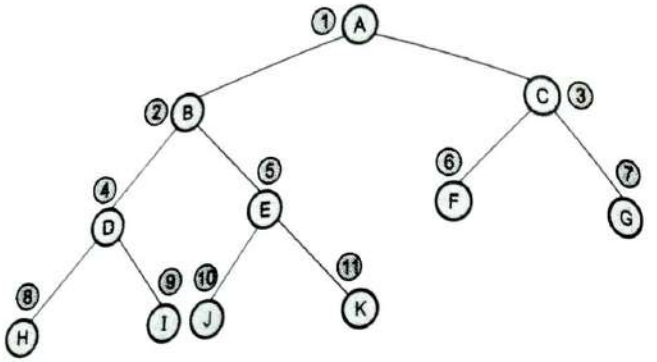


Fig. 2.3.1 Breadth-first search

D and J are goal state.

[D is found through the path] → (A-B-D) (shallowest goal)

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

2.3.2.2 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, their 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 pathcost. **(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 "c".

2) Optimality : If cost of every step is greater than or equal to some small positive constant then uniform cost search will yeild 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 E. Then the algorithm's worst-case time and space complexity is $O(b^{[C^*/\epsilon]})$, which can be much greater than b^d.

- Uniform cost seach

  Time complexity - $O(b^{C/\epsilon})$

  Space complexity - $O(b^{C/\epsilon})$

where - C is the cost of optimal solution.

Assuming J and D are both goal state.

A - B - E - J

Note : The costs are associated with each edge.

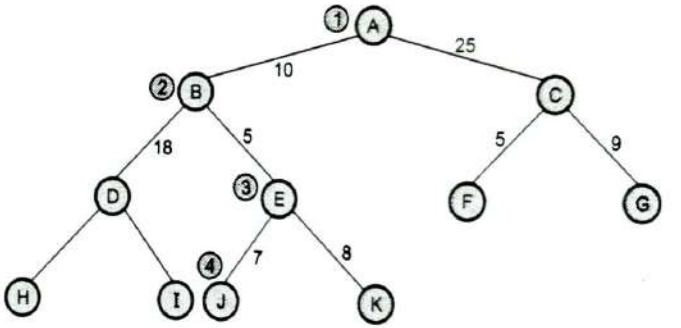


Fig. 2.3.2 Uniform cost search

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

2.3.2.3 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 guareentees 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);
    }
}
```

- Time complexity - $O(b^d)$.
- Space complexity - $O(b^d+1)$.

2.3.2.4 DFS [Depth First Search]

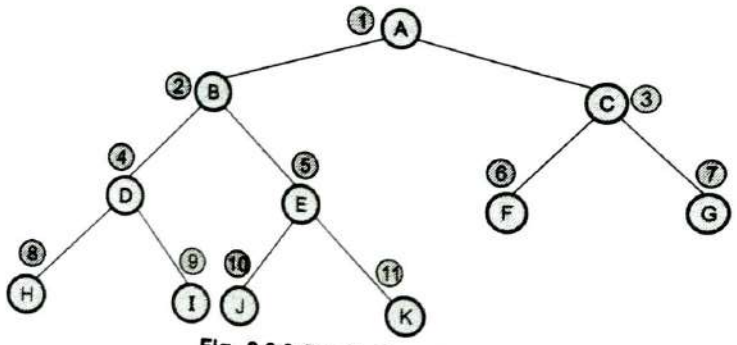


Fig. 2.3.3 Depth First Search

Goal state path - (A - B - D)

where D is a goal state.

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.

2.3.2.5 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 within 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]

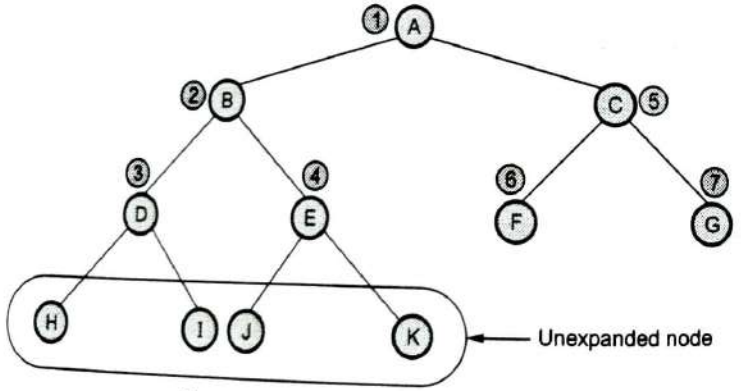


Fig. 2.3.4 Depth-Limited Search

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).

2.3.2.6 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]**

Shallowest Goal Node D Found :

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

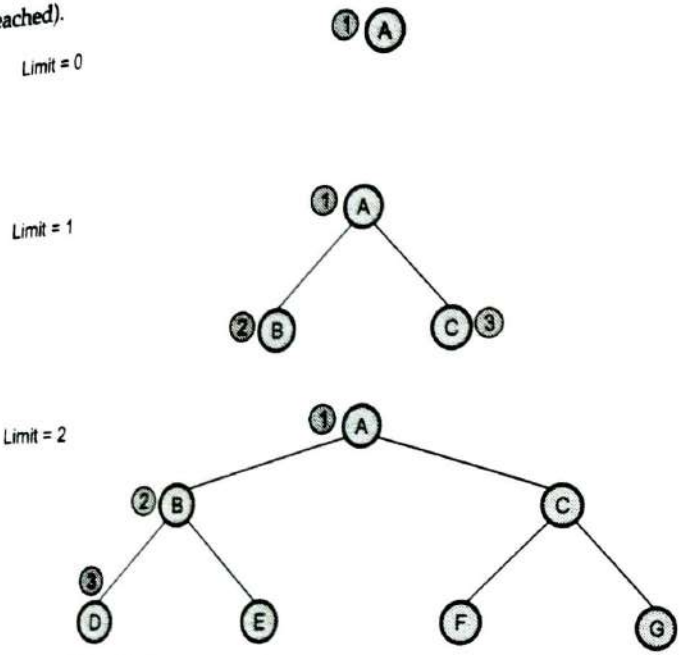


Fig. 2.3.5 Iterative Deepening Depth-First Search

2.3.2.7 Bidirectional Search

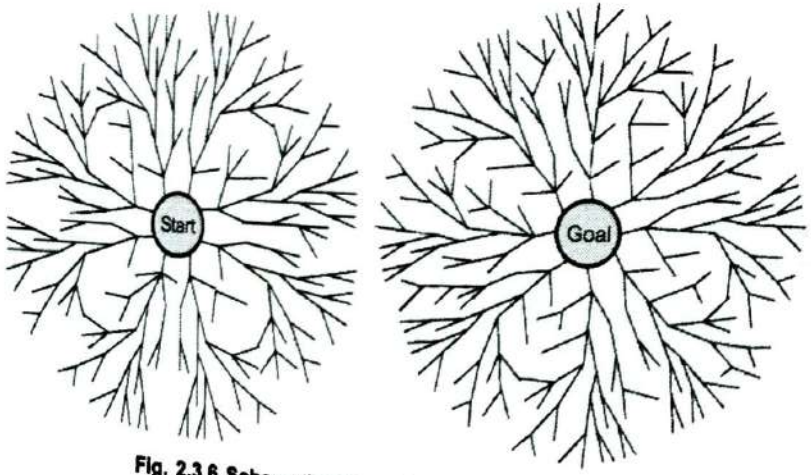


Fig. 2.3.6 Schematic view of bidirectional search

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 Fig. 2.3.6]

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 - $O(b^{d/2})$

2.4 The Repeated States

2.4.1 Avoiding Repeated States

- When we are finding solution by searching in state space tree, some states can be repeatedly explored. This will increase the total cost. Hence we need to avoid repeated states.
- In some problems we need to explore all the repeated states again as they may reach to goal state.

For example -

- Problems where the actions are reversible, such as route-finding problems and sliding-blocks puzzles.
- Search tree for these problems are infinite. But if we cut off some repeated states. We can generate only the portion of the tree that engross the state space graph.
- In some problems repeated states are not required to explore again as these states will not surely lead to goal states.
- In a state space graph we can drop multiple paths and can construct a tree where there is only one path to each state.

For example -

- Consider the 8-queen problem. Here each state can be reached through only 1 path. If we formulate the 8-queens problem such that a queen can be placed in any column, then each state with 'n' queens can be reached by n! different paths.

2.4.2 Serious Problems Caused due to Repeated States

1) Because of repeated states solvable problem can become unsolvable if the algorithm is unable to detect repeated states.
2) Because of repeated states looping paths can be generated which can lead to infinite search which is not practical approach.
3) Because of repeated states memory is unnecessaryly wasted as same state is maintained multiple times.

2.4.3 Example of State Space that Generate an Exponentially Larger Search Tree

a)

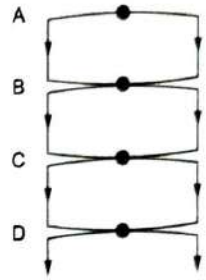


Fig. 2.4.1 State space that generate exponentially large search tree I

Fig 2.4.1 shows a state space in which there are two possible actions leading from A to B, two from B to C and so on. The state space contains d+1 states, where d is maximum depth.

b)

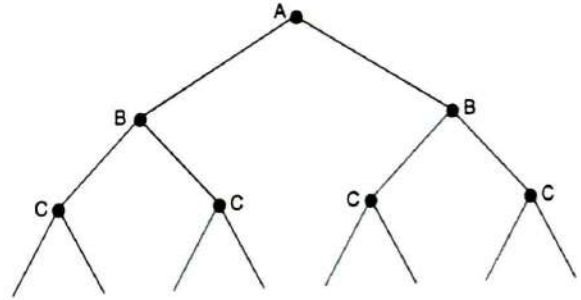


Fig. 2.4.2 State space that generate exponentially large search tree II

In the above search tree, (Fig. 2.4.2) there are 2^d branches corresponding to the 2^d paths through the space.

In the search tree shown in Fig. 2.4.3, A is a root node. On a grid each state has four successors, therefore the tree including repeated states has 4^d leaves. But as we can see, only about $2d^2$ are distinct states within d steps of any given state.

c)

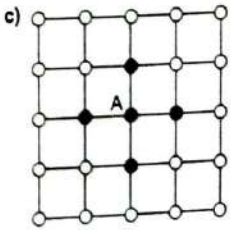


Fig. 2.4.3 State space that generate exponentially large search tree III

Note : If limit of depth in state space tree is specified then it is easy to reduce repeated states which leads to exponential reduction in search cost.

2.4.4 Algorithm that can Avoid Repeated States
(Algorithms that Forget their History are Sure to Repeate the States)

- If an algorithm remembers every state that it has visited, then it can be viewed as exploring state space graph directly.
- We can device a new algorithm called as graph-search algorithm which is more efficient than earlier tree-search algorithm. A graph-search algorithm maintains two data structures closed list and open list to avoid exporation of those node which are already explored (i.e. trying to avoid repeated states).

Close list : A closed list is a data structure maintained by algorithm which stores every expanded node. Algorithm discards the current node if it matches with node on the closed list.

Open list : A open list is a data structure maintained by algorithm which stores fringe of unexpanded node.

Graph-Search Algorithm

[Data structure required] :

- Node n
- State description
- Parent (may be backpointer) (if needed)
- Operator used to generate n (optional)
- Depth of n (optional)
- Path cost from s to n (if available)
- Open list
  - initialization : {s}
  - node insertion removal depends on specific search strategy
    - Closed list
  - initialization : { }
  - organized by backpointers to construct a solution path

[Algorithm] :

```
open : = {s} ;
closed : = { } ;
repeat
{
```



```
n : = select (open) ; /* select one node from open for expansion */
if n is a goal

then exit with success ; /* delayed goal testing */
expand (n)
/* generate all children of n put these newly generated nodes in open (check duplicates)
put n in closed (check duplicates) */
until open = { } ;
}
exit with failure
```

For the above algorithm worst case time and space requirements are proportional to the size of the state space. This may be much smaller than $O(b^d)$.

A repeated state is detected when algorithm has found two paths to same state. if the newly discovered path is shorter than the original one then it will be discarded by algorithm. Then there is a chance that graph-search could miss an optimal solution as it can discard one of the paths that can lead to optimal state.

2.5 Searching with Partial Information

All the earlier search strategies we have seen, assumed that environment is fully observable and deterministic.

In such a environment agent -

1) Knows what is the effect each action.
2) Can calculate exactly which state results from any sequence of actions.
3) Knows which state it is in.
4) Does not get new information after each action.

If environment is not fully observable and knowledge of the states and actions is incomplete then agent has different types of task environment. Such a environment leads to three distinct types of problems -

1) Sensorless problems
2) Contigency problems
3) Exploration problems.

2.5.1 Sensorless Problems (Conformant Problems)

In this type of problems -

A) Agent has no sensors at all

B) According its own knowledge agent could be in one of several possible initial states.

C) As agent has several possible initial states each action can lead to one of the several possible successor states.

- **Solving sensorless problems in a fully observable, completely known, deterministic environment ⇒**

In sensorless problem agent already have predefined knowledge about its various states. Out of these state set only one of the state is going to be initial state. Though agent has no sensor, hence no percepts, still from its own state and action it can reach to certain conclusion states, through which it can further reach to goal state. It means that we can say agent can **coerce (force to reach)** the world to reach at a perticular expected state on his own.

Steps taken to solve the problem are as follows -

1) First agent searches for **belief state** instead of physical state.

[The belief state - it is a set of states representing agent's current belief about the possible physical states it might be in. For finding such belief state agent should have reasoning for each state it can be in, based on its original knowlege].

2) Initial state is belief state which can further mapped to another belief state.

3) Action is applied to belief state by taking union of the results obtained from applying the action to each physical state in the belief state.

4) We now get a path which connects several belief states.

5) Solution is a path that leads to a belief state, all of whose members are goal states.

Note : If the physical state space has 's' states, the belief state space will have 2^s belief states.

For Example

- Black ball picker robot

2.5.2 Contigency Problem

- If environment is partially observable or if actions are uncertain, then agent can sense and perceive new information after each action. This new information can lead to new problem (or critical situation) which is called as contigency. If this situation occurs then agent needs to plan next action to handle this contigency.

• Solving contigency problem -

1) For contigency problem one needs to form a tree, here each branch of a tree may be selected depending upon the percepts received upto that point in the tree.

2) Then by searching and expanding previous level agent can reach to a goal.

In some situations for contigency problems we get purely sequential solutions.

Note -

Searching algorithm for contigency problems are more complex than algorithms we have studied so far.

The agent design is also different for these problems, because agent can act before it has found a guaranteed plan.

Agent continue to solve problem, considering new information and minimizing contigency by checking the output of its action. This is termed as interleaving of searching and execution. [That is alternately search and execution is done].

• For example :

Trading agent

• Special case of contigency problem : A contigency problem is called as adversarial problem if the uncertainty is caused by the actions of another agent. (More adverse situation).

For example -

Agent in game of chess.

2.5.3 Extreme Case of Contigency Problem

Exploration problem : When state as well as the actions of the environment are unknown the agent must act to discover them. The interleaving of searching and execution would be useful to solve exploration problem.

For example -

The boat driving robotic agent.

2.6 Production System

GTU : Winter-18, 19

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 → 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.
- **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.**
- Model for Human Problem Solving (SOAR, ACT*)

2.7 Issues in the Design of Search Problem

GTU : Winter-17

Design is a signature of human intelligence. It was more difficult and big challenge for AI. Designs shaped early ideas on automated problem solving and reasoning. It provided ever since, task for AI which are as follows :

i) Studying, explaining and modeling the faculties underlying intelligent behaviour.

ii) Engineering systems that are capable of exhibiting for a such behavior.

2.7.1 Issues in Engineering Design

- When basically increasing integration on one hand and distribution work on the other hand are the trends that determine the present and even more the future of

engineering design. This type of situation is by no means a paradox. When result of various engineering differ, different activities should be put together. When product is complete and launched in the market, it should be efficient and quality.

- Collaborative design transforms a chaining process into a parallel one with the aim of avoding iteration and utilizing resources, more efficiently, (Gadh-1996). Concurrent engineering has an even broader focus. Acknowledgeing that, engineering design must take into account the intrinsic requirement and properties of the process that bring to life, create, maintain and re-cycle artifacts. It embraces all main life-cycle activities such as marketing, design, manufacturing, distribution, sales, operation, maintenance, disposal and re-cycling. The expectations are greater, fewer modifications, shorter time-to market, more efficient resource utilization, better process and product quality (Sohlenius, 1992) though reliable (Alting and Legarth 1995 ; Jansen and Krause, 1996).

2.7.1.1 Design Synthesis

a) Automated search

- The search problems in design have been addressed mainly by the application of greedy numerical optimization algorithm, random re-start hill-climbing, stochastic local and global search methods. The technique is best if a design problem can be coined as a combinatorial optimization problem. They are account neither for the cost nor for the bounded resources of best for computations.
- Tangible result in economic rationally lies in the center of intelligent behavior (Doyle etal, 1996 ; Russel, 1997)

b) Model based and functional reasoning, constraint satisfaction

- Knowledge about artifacts enables reasoning about them, even if they are only partially defined. Can constrain the design process by evaluating partial design solution can be very complex problem (McGuinness and wright, 1998).
- The design process is not self automanted, rather alternatives are generated deriving the implications of the user provided data.
- The consistency of data could be preserved in the whole interactive design process. Functional models, the traditional decomposition of function to subfunction can even better direct the design process through they cannot work on multiple levels of abstraction and without human interaction (Umeda and Tomiyama, 1997).

c) Case based approach to design

- Lack of sufficient understanding of design synthesis and the resistance of expertise to formalization attempts on one hand, where the surprising complexity of automated problem solving process on the other hand led AI practitioners back to

tranditional design. Practice to the re-use of previous solution. In case base design (Maher and Garza, 1997) cases are episodic descriptions of problem together with their associated solution and trace of the solution process. Reasoning goes through the steps of adapting such stored cases that are particulary relevant to a new problem. Side-stepping the main line of reasoning, new cases be added to the case-base, CBR solve any problem to essay.

2.7.2 Issues in Design of Search Program

Following are major issues faced while designing a search program,

1. The direction in which to conduct the search that is either forward or backward reasoning is to be used.
2. How to select applicable rules for matching that would depend on data set.
3. How to represent each node of the search process. That is how the knowledge representation should be done so as to reduce the accessing time while handling the data.

2.8 AI Problem Characteristics

GTU : Summer-12,18, Winter-12,14,17

2.8.1 Problem Characteristics

To choose an appropriate method for a particular problem it is important to study the problem characteristics which are as follows :

1. **Is the problem decomposable to smaller or easier problems ?** Can the problem be broken down to smaller problems so that it can be solved independently ? Such a decomposed problem can be solved easily.

2. **Can problem solution steps be ignored or undone ?**

The problem can fall in one of the following three categories.

- **Ignorable** - In this type of problem solution steps can be ignored. For example - Theorem proving. In theorem proving if later it is known that certain step is not useful then still one can proceed further as nothing is lost due to repeated steps.
- **Recoverable** - In this type of problem solution steps can be undone and backtracked. For example - The 8-Puzzle problem. In this problem while moving from initial state to goal state one may make some wrong moves but later can backtrack and recover by making correct move.
- **Irrecoverable** - In this type of problem moves cannot be retracted. For example - Playing Chess : in the game of chess one can make wrong move but then this move is neither to be ignored nor it can be recovered.

The above knowledge about the problem helps in determining the control structure. Ignorable problems can be solved using a simple control structure that never backtracks. Recoverable problems can be solved using backtracking. Irrecoverable problems can be solved by recoverable style methods via planning.

3. **Is the problem universe predictable ?**

Further problem can be categorized as problem with certain outcome and problem with uncertain outcome. In certain outcome problems planning can be done to generate a sequence of operations that lead to a solution. For example in the 8-Puzzle problem every time move is made, it is known exactly what will happen. That is there is certain outcome !

In uncertain outcome problem planning can generate a best sequence of operations that can probably lead to problem solution. For example in Bridge game one cannot know exactly where all the cards are or what the other players will do on their turns. That is there is uncertain outcome.

For certain-outcome problems, planning can used to generate a sequence of operators that is guaranteed to lead to a solution where as for uncertain-outcome problems, a sequence of generated operators can only have a good probability of leading to a solution. Plan revision is made as the plan is carried out and the necessary feedback is provided.

4. **Is a good solution absolute or relative ?**

Another category of problem is that, when we get solution is it the best one or may be some other path can give optimal (best among all) solution. For certain problems multiple feasible solutions exist but out them one is the best and it is the required solution.

Consider following example,

1. Marcus was a man
2. Marcus was a Pompeian
3. Marcus was born in 40 A.D.
4. All men are mortal.
5. All Pompeians died when the volcano erupted in 79 A.D
6. No mortal lives longer than 150 years.
7. It is now 2008 A.D.

Question : Is Marcus alive ?

8. Marcus is mortal (1, 4)
9. Marcus'age is 1964 years. (3, 7)

10. Marcus is dead. (6, 8, 9)

——— OR ———

11. All Pompeians are died in 79 A.D.

12. Marcus is dead.

In above problem different reasoning paths lead to the answer. It does not matter which path we follow.

In Travelling Salesman Problem one has to try all paths to find the shortest one. Any-path problems can be solved using heuristics that suggest good paths to explore. For best-path problems, much more exhaustive search will be performed to get optimal solution.

5. Is the solution a state or a path ?

Consider following example,

Finding a consistent interpretation :

-"The bank president ate a dish of pasta salad with the fork".

"bank" refers to a financial situation or to a side of a river ?

"dish" or "pasta salad" was eaten ?

Does "pasta salad" contain pasta, as "dog food" does not contain "dog" ?

Which part of the sentence does "with the fork" modify ? What if "with vegetables" is there ?

In the above problem no processing record is required only final answer is outcome.

In the Water Jug Problem the path that leads to the goal must be reported.

A path-solution problem can be reformulated as a state-solution problem by describing a state as a partial path to a solution. The question is whether that is natural or not.

6. Is the problem using consistent knowledge base ? Does the problem require lots of knowledge or uses knowledge to constrain solutions ?

To solve problem correctly it is highly necessary that the knowledgebase is consistent on the scale of time and data. In some problems the knowledge base is consistent and in some it is not. For example for evaluating Boolean expression complete knowledge of theorems and laws is required which are always true. Where as in data about rate card of hourly labour would change as per time and need to be updated so that manufacturing cost is correctly computed. Many reasoning system work well in consistent domain that are un-appropriate for inconsistent domain.

7. What is the role of knowledge ?

In certain domain, knowledge plays crucial role while solving the problem. Consider following two problem,

i) Playing Chess

Here knowledge is important only to constrain the search for a solution. Additional knowledge of strategies would help to get faster solution.

ii) Reading Newspaper

Here knowledge is required even to be able to recognize a solution.

8. Does the task require periodic human-interaction with computer ?

Here one need to distinguish between 2 types of problems :

i) Solitary problem, in which there is no intermediate communication and no demand for an explanation of the reasoning process.

ii) Conversational problem, in which intermediate communication is to provide either additional assistance to the computer or additional information to the user.

Problem Classification

There is a variety of problem-solving methods, but there is no one single way of solving all problems.

Not all new problems should be considered as totally new. Solutions of similar problems can be exploited.

2.8.2 Example Problems and their Characteristics

1. Analysis of the water-Jug problem with respect to seven problem characteristics.

Consider the water Jug problem with the capacity of the two jugs of 3 and 4 litres with no marking in them. Given a water supply with a large storage using these two jugs, how one can separate 2 litres of water.

Analysis of **water jug problem** with respect to the seven problem characteristics are as follows -

i) Is the problem decomposable ?

In water jug problem, the goal is to get exactly 2 gallons of water in the 4-gallon jug. The idea of this solution is to reduce the problem into two subproblem, which is not possible. Even if one devide it into subproblem, they are not independent.

ii) Can solution be ignored or undone ?

In our problem suppose program makes a stupid move and realizes it a couple of moves later. It cannot simply play as through it had never made the stupid move. All it can do is to simply back up and start the game over from that point.

iii) Is the problem's universe predictable ?

In the problem, every time one make a move one know exactly what will happen. This means that it is possible to plan as entire sequence of move and be confident that are know what the resulting state will be. Hence problem is not universe predictable.

iv) Is a good solution absolute or relative ?

In water jug problem, the goal is to get exactly 2-gallon of water in 4-gallon jug. It does not matter which path one follows or how one got the goal state. If one do follow one path successfully to the answer, there is no reason to go back and see if some other path might also lead to a solution. Hence the good solution is absolute.

v) Is the solution a state or a path ?

Here the solution is state in which 4-gallon jug is filled with exactly 2-gallon water, but the way or path does matter.

vi) What is the role of knowledge ?

In water jug problem, the knowledge required is, just the set of rules for determining legal move and some simple control mechanism that implement an appropriate search procedure.

vii) Does the task require interaction with a person.

The problem does not require any interaction with a person. It just requires problem, set of rules and a goal state at the initial stage. Now between execution no knowledge and no interaction with person is required.

(a) State :

{(i, j), i = 0, 1, 2, 3, 4

j = 0, 1, 2, 3 }

i - Number of liters of water in 4-litre jug.

j - Number of litres of water in 3-litre jug.

b) Initial state :

{(i, j) = (0, 0) }

c) Goal state :

{ (i, j) = (2, n) }, n value can be in between 0 to 3.

d) Condition :

1) We can fill jug with pump and large storage is there of water.

2) We can pour water out of a jug to the ground.

3) We can pour water from one jug to another.

4) There is no measuring device available.

e) State space representation

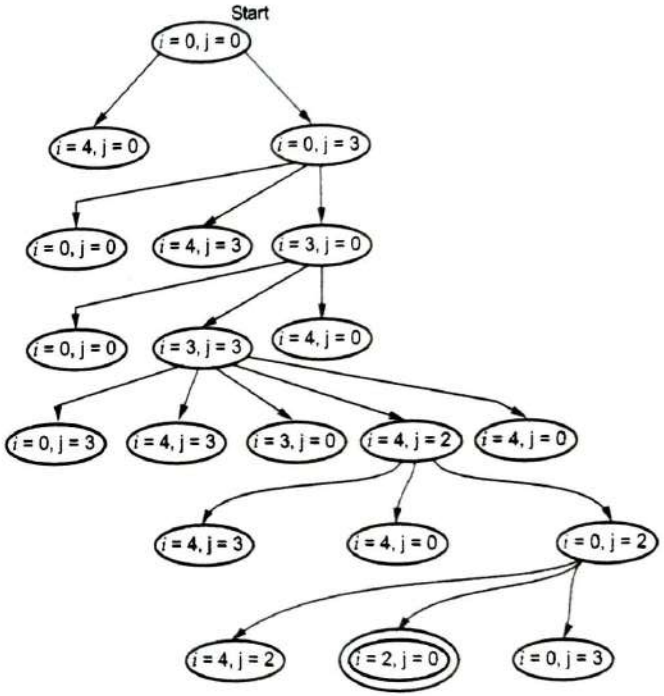


Fig. 2.8.1 State space representation

Note :

- Each state enclosed within a circle shows the resultant state after application of appropriate rule on that state.
- Initial state is represented as start state.
- Final state is enclosed within double circle.

2. Travelling salesman problem (TSP).

Analysis of **TSP** with respect to seven problem characteristics are as follows -

i) Is the problem decomposable ?

This problem is not decomposable. In this problem, the goal is to find shortest path that visit each city exactly once, and hence whole problem will be considered simultaneously.

ii) Can solution steps be ignored or undone ?

In TSP suppose program makes a stupid move means choose the path which is not a shortest path and realizes it a couple of move later. It cannot simply go further as through it had never made wrong move. All it can do is to start the game again and try to make the best of the current situation and go on from there. So solution steps cannot be ignored or undone.

iii) Is the universe predictable ?

In travelling salesman problem, every time one choose a city it is know exactly which will be the next choice, because here all the distance between each cities are known. So here it is possible to plan an entire sequence of move. One can use planning to avoid having to undo actual move. So there is nothing universe predictable.

iv) Is a good solution absolute or relative ?

In the problem, goal is to find the shortest route that visit each city exactly once. Now the next city he has to visit cannot be sure unless we also try all other path to make sure that none of them is shorter. Hence, the good solution is relative but not absolute.

v) Is the solution a state or a path ?

The goal of the problem is to find the shortest path that visit each city exactly once. For this type of problem one really must report is not the final state but the path one found to that state. Thus, a statement of a solution to this problem must be a sequence of operations that produces the final state.

vi) What is the role of knowledge ?

The required knowledge is just the knowledge of the cities to be visited, the distance between these cities and some simple control mechanism that implement an appropriate search process.

vii) Does task require interaction with a person.

In travelling salesman problem, there is no need to interact with person.

3. 8 puzzle problem

Problem characteristic	Satisfied	Reason
Is the problem decomposable ?	No	One game have Single solution
Can solution steps be ignored or undone ?	Yes	We can undo the previous move
Is the problem universe predictable ?	Yes	Problem Universe is predictable because to solve this problem it require only one person .we can predict what will be **position of blocks** in next move.
Is a good solution absolute or relative ?	absolute	Absolute solution : Once you get one solution you do need to bother about other possible solution. Relative solution : once you get one solution you have to find another possible solution to check which solution is best (i.e low cost). By considering this 8 puzzle is **absolute .**
Is the solution a state or a path ?	Path	Is the solution a state or a path to a state ? – For natural language understanding, some of the words have different interpretations. Therefore sentence may cause ambiguity. To solve the problem we need to find interpretation only, the workings are not necessary (i.e path to solution is not necessary). So In 8 puzzle winning state (goal state) describe path to state.
What is the role of knowledge ?		Lot of knowledge helps to constrain the search for a solution.
Does the task require human-interaction ?	No	Conversational· In which there is intermediate communication between a person and the computer, either to provide additional assistance to the computer or **to provide additional information to the user,** or both. In 8 puzzle additional assistance is not required.

- **State space representation for 8 puzzle problem**

In 8 puzzle problem, we have a square frame containing 8 tiles and an empty slot, i.e. total 9 sub squares. The tiles are numbered from 1 to 8. We can move the tiles in square frame by moving a tile into empty slot.

The goal state is one having the tiles in numerical order with the last square an empty slot as shown in Fig. 2.8.2.

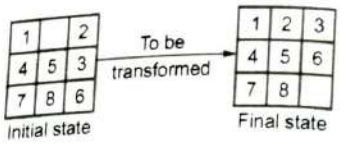


Fig. 2.8.2 State Space of 8-puzzle

Initial and final states are shown in Fig. 2.8.2 and the operators for this problem are -

UP - If the hole is not touching the top frame, move it up.

DOWN - If the hole is not touching the bottom frame, move it down.

LEFT - If the hole is not touching the left frame, move it left.

RIGHT - If the hole is not touching the right frame, move it right.

In state space representation we have all possible set of states for a given problem.

Or states space is a directed graph with all the states as nodes.

A node is said to exist if it is possible to obtain it from the initial state by application of a set of operators.

We can reach to another state by applying an operator on first state.

If the entire space representation is given of a problem, then one can easily trace the path from initial state to the goal state and identify the set of operators and their sequence necessary for reaching to goal state.

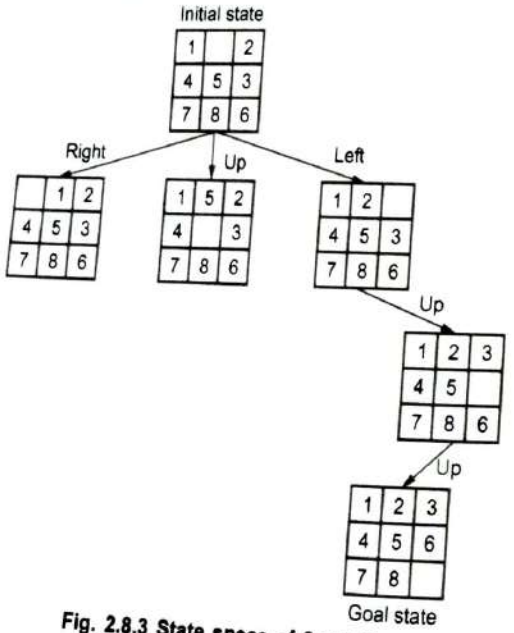


Fig. 2.8.3 State space of 8-puzzle

4. Cryptarithmatic problem

Problem characteristic	Satisfied	Reason
Is the problem decomposable ?	No	One problem have Single solution
Can solution steps be ignored or undone?	No	In actual problem we can't undo previous steps.
Is the problem universe predictable?	Yes	Problem Universe is predictable as we can figure out all possible moves.
Is a good solution absolute or relative?	Absolute	Absolute solution : once you get one solution you do need to bother about other possible solution. Relative solution : once you get one solution you have to find another possible solution to check which solution is best (i.e low cost).
Is the solution a state or a path?	Path	In this problem goal state describe path to state
What is the role of knowledge?		Lot of knowledge helps to constrain the search for a solution.
Does the task require human-interaction?	No	In cryptarithmetic additional assistance is not required

5. Chess

Problem characteristic	Satisfied	Reason
Is the problem decomposable ?	No	One game have single solution
Can solution steps be ignored or undone ?	No	In actual game (not in PC) we can't undo previous steps.
Is the problem universe predictable ?	No	Problem Universe is not predictable as we are not sure about move of other player (second player).
Is a good solution absolute or relative ?	absolute	Absolute solution : once you get one solution you do need to bother about other possible solution. Relative solution : once you get one solution you have to find another possible solution to check which solution is best (i.e low cost). By considering this chess is absolute.
Is the solution a state or a path ?	Path	Is the solution a state or a path to a state ? - For natural language understanding, some of the words have different interpretations. Therefore sentence may cause ambiguity. To solve the problem we need to find interpretation only, the workings are not necessary (i.e path to solution is not necessary). So in chess winning state (goal state) describe path to state.
What is the role of knowledge ?		Lot of knowledge helps to constrain the search for a solution.

Does the task require human-interaction ?	No	Conversational - In which there is intermediate communication between a person and the computer, either to provide additional assistance to the computer or **to provide additional information to the user**, or both. In chess additional assistance is not required.

6. Tower of Hanoi problem

Problem characteristic	Satisfied	Reason
Is the problem decomposable ?	No	One game have single solution
Can solution steps be ignored or undone ?	Yes	
Is the problem universe predictable ?	Yes	
Is a good solution absolute or relative ?	absolute	Absolute solution : once you get one solution you do need to bother about other possible solution. Relative solution : once you get one solution you have to find another possible solution to check which solution is best (i.e low cost). By considering this Tower of Hanoi is **absolute .**
Is the solution a state or a path ?	Path	Is the solution a state or a path to a state ? – For natural language understanding, some of the words have different interpretations . Therefore sentence may cause ambiguity. To solve the problem we need to find interpretation only , the workings are not necessary (i.e path to solution is not necessary). So In tower of Hanoi winning state(goal state) describe path to state .
What is the role of knowledge ?		Lot of knowledge helps to constrain the search for a solution.
Does the task require human-interaction ?	No	Conversational· In which there is intermediate communication between a person and the computer, either to assistance to the computer or **to provide additional information to the user,** or both. provide additional In tower of Hanoi additional assistance is not required.

State space representation

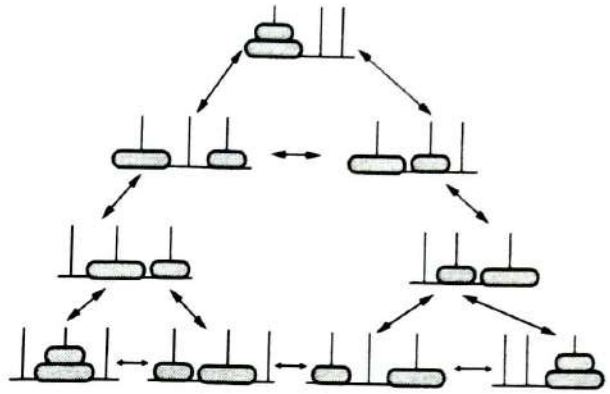


Fig. 2.8.4 Towers of Hanoi state space representation

The legal moves in this state space involve moving one ring from one pole to another, moving one ring at a time and ensuring that a larger ring is not placed on a smaller ring.

7. The missionaries and cannibals problem

- Three missionaries and three cannibals find themselves on one side of a river. They have agreed that would all like to get to the other side. But missionaries are not sure what else the cannibals have agreed to, so missionaries want to manage the trip across the river in such a way that number of missionaries on either side of the river is never less than the number of cannibals who are on the same side. The only one boat available hold only two people at a time. How can everyone get across the river without missionaries risking being eaten ?
- **Analysis of problem with respect to problem characteristics** - It encompasses a variety of specific technique, each of which is particularly effective for a small class of problem. In order to choose most appropriate method, it is necessary to analyze along seven key dimension -

1) Is the problem decomposable into a set of independent smaller or easier subproblem ?

The missionaries and cannibals problem is not decomposable because here every movement of either missionaries or cannibals effect next movement missionaries or cannibals. If one try to decompose it, he will unable to get any subproblem which is independent of others.

2) Can solution steps be ignored or at least undone if they prove ? unwise. Here many solution steps in the problem are ignored or undone when they proved unwise. undone when they proved unwise. In missionaries and cannibals problem, in initial state one have three possible move like, two missionaries move simultaneously to other side, two cannibals move simultaneously to other side, or one cannibal and one missionary can move to other side. In first case, if two missionaries move then the number of cannibals is more than the number of missionaries so this solution step can be ignored or undone. Other two possible moves can be used but second possible move (two cannibals move) will be ignored further.

3) Is the problem universe predictable ?

In missionaries and cannibals problem, every time one make a move, one know exactly what will happen. This means that it is possible to plan an entire sequence of move and be confident that we know what the resulting state will be. So, by explaining all the above, it means to say that, here nothing is universe predictable, means result of every possible move is exactly known.

4) Is a good solution to the problem obvious without comparison to all other possible solution ?

In the problem there is one and only one good solution to the problem. Hence no need to compare it with other. If there exists more than one path of the solution then one cannot be sure unless we also try all other path to make sure that none of them is more efficient.

5) Is the desired solution a state of the world or a path to a state ?

In missionaries and cannibals problem it is not sufficient to report that one has solved the problem and that the final state is reached. Here what we really must report is not the final state but the path that leads to the state. Thus, a statement of a solution to this problem must be a sequence of operations, that produce the final state.

6) Is a large amount of knowledge absolutely required to solve the problem, or is knowledge important only to constraint the search ?

Here there is no need of large amount of knowledge. The required knowledge is very little - just the rule for determine the legal moves and some simple control mechanism that implements an appropriate search procedure. Additional knowledge about such things as good strategy and tactics could of course help considerably to constraint the search and speed up the execution of the program.

7) Can a computer, given the problem return the solution, or will the solution of the problem require interaction between the computer and person ?

In the problem, there is no need of interaction between the computer and person. Here, given the problem, set of rule and a final state it returns the solution with each intermediate step.

- **State space representation of missionaries and cannibals problem** - In state space representation, every missionary is denoted by M and cannibals is denoted by C. Numbers of M's and C's show the number of missionaries and cannibals. Circle represents the state and arrow represents the forward and backward move.

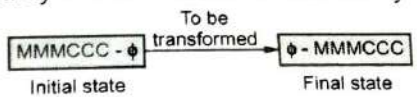


Fig. 2.8.5 Initial and final state of the problem

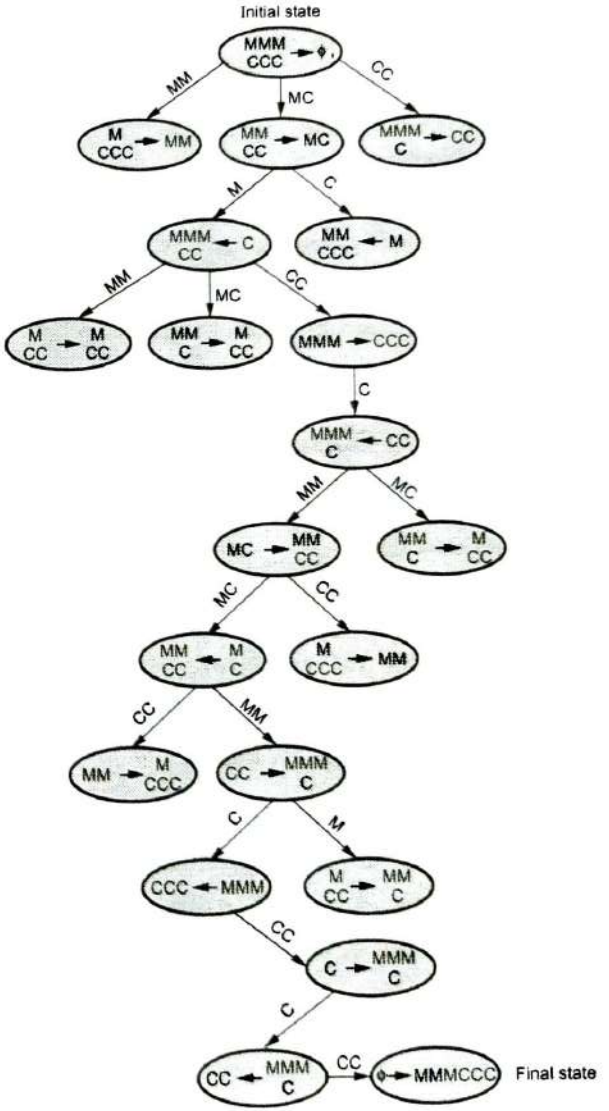


Fig. 2.8.6 State space representation

GTU : Winter-12,14,15,18,19, Summer-12,16,17,19

2.9 Additional Problems

- We have seen problem formulation. As we can see there are variety of task environments ranging from easier to complex ones, therefore our problem solving approach should be realistic, which can give solution for any type of task environment, which inturn is useful for mankind.
- Basically we can distinguish problems in two catagories. Toy problems and real-world problems.

2.9.1 Toy Problem

A Toy problem is a problem which illustrates various problem-solving methods.

Characteristics of Toy problem

1) For the Toy problem exact and precised description can be given.
2) These problem provide basis for solving some real-life problems.
3) They can be used by researchers to compare performance of algorithms.

For example : 8-queen puzzle, vaccum world, ball picker robot.

2.9.2 Real World Problem

- A real-world problem is a problem which needs to be solved so that its solution can be utilized in practical life. They will not have some predefined. Well described, single specification. Infact while considering these problem general formulation needs to be done.
- People do care about the solutions of real-world problem as they are benefited from it.

For example : Route finding for a trip, Travelling salesman problem, Robot navigation, Car reversing guide.

2.9.3 Problem Formulation for Toy Problems

Now let us see some toy problem examples.

Each example gives all 4 aspects of problem formulation namely

1) Initial state 2) Successor function 3) Goal state 4) Path cost.

Also we list all the possible states of problem solution, through which agent can pass or move.

Example 1 : Ball Picker Robot Problem Formulation

Problem statement : 2 buckets are full of balls. Either a ball is white colour or red colour. Objective is to pick all black colour balls from bucket.

1) Initial state : Any state can be designated as initial state.

2) Successor function : This function generates legal states that result from trying the three actions (Left, Right, **Pick Black** colour ball).

3) Goal test : This checks if all **Black** coloured balls from both the buckets are picked up.

4) Path cost : Each step cost is 1, so path cost is the number of steps in the path.

5) States : The agent is at one of the 2 location (at bucket A or at bucket B), from each of which black colour balls may have been picked up completely or may not be.

Thus there are 2 (locations) $\times$ $2^{2\text{ (Two possibilities)}}$ = 8 possible world states.

- **Comparison with real world -**

If we compare with real world situation we can see that above toy problem has all well specified description. Agent has discrete location (in real world many locations can be there, also as time passes over, more locations can come up). Agent has discrete pick up item i.e. **Black** colour ball (In real world this items can have more dimensions). Also once all **Black** colour balls are picked up from bucket then again new black colour balls are not expected. (Where as in real world we may expect bucket to have new **Black** colour balls).

- **The state space representation for ball picker robot**

(See Fig. 2.9.1 on next page)

Example 2 : 8-Queen Problem

Problem Statement : Given a chess board of 8×8 size, objective is to place 8 queens on a chess board such that no two queens are attacking. (A queen attacks any queen in the same row, column or diagonal).

1) Initial state : No queens on the board.

2) Successor function : Add a queen to any empty square, such that it do not attack other queen.

3) Goal test : 8-queens on the board such that no queens attack each other.

4) Path cost : Each step costs 1, so the path cost is the number of steps in the path.

5) States : Any arrangements of 1 to 8 queens on the board is a state.

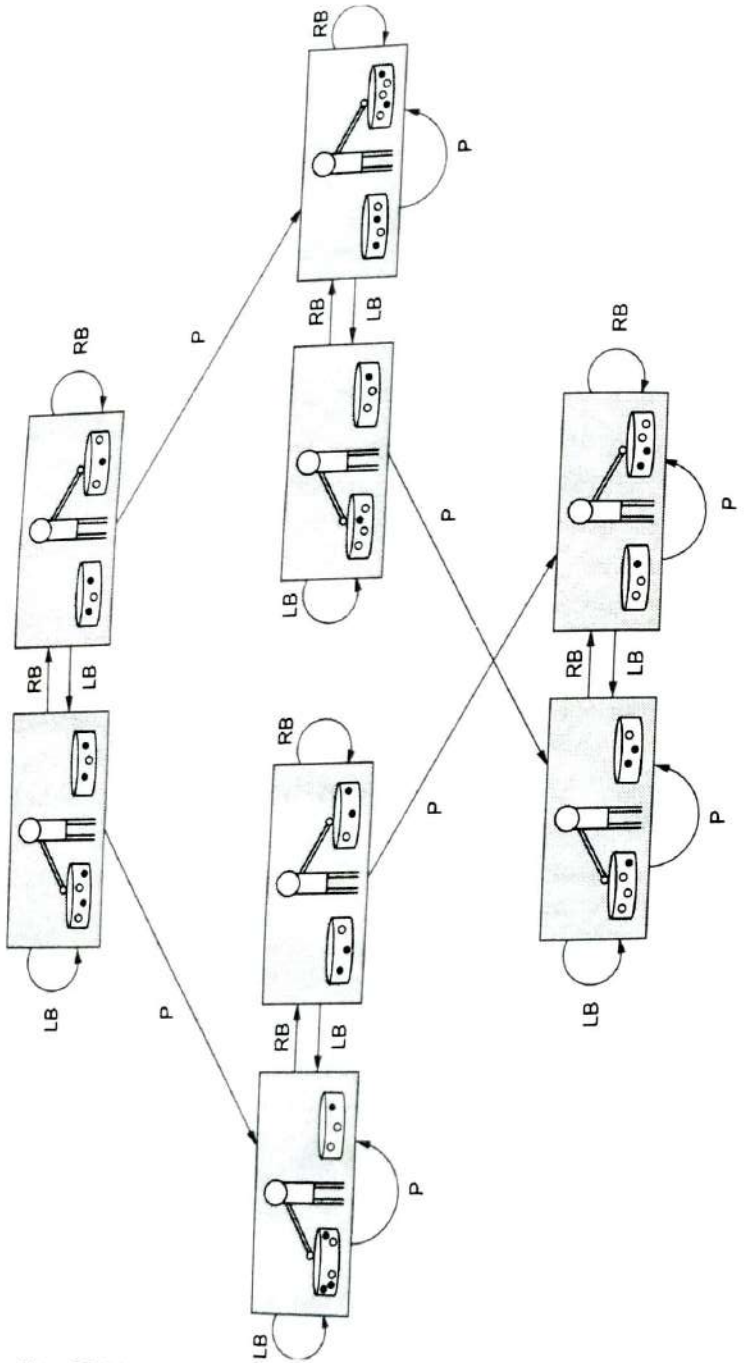
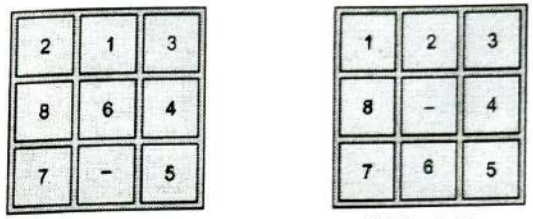


Fig. 2.9.1 The state space representation for ball picker robot

This formulation has, $64 \times 63 \times ... 57 \approx 3 \times 10^{14}$ possible sequences to investigate. But the condition of non-attacking reduces the states to 2057 from 3×10^{14}.

Example 3 : 8-Puzzle

Problem Statement : Consists of 3×3 board with eight numbered tiles and a blank space. A tile adjacent to the blank space can slide into the space. The object is to reach a specified goal state, such as the one shown on the right of the figure.



(a) Initial state **(b) Goal state**

Fig. 2.9.2 8 Puzzle problem

1) States : A state description specifies the location of each of the eight tiles and the blank in one of the nine squares.

2) Initial state : Any state can be designated as the initial state. Note that any given goal can be reached from exactly half of the possible initial states.

3) Successor function : This function generates the legal states that result from trying the four actions. (blank moves, left, right, up or down).

Example 4 - Cryptarithmetic

Problem Statement : Find an assignment of digits (0, ..., 9) to letters so that a given arithmetic expression is true.

For example - Applying digit to a letter.

	F	O	R	T	Y	→		2	9	7	8	6
+			T	E	N	→	+			8	5	0
+			T	E	N	→	+			8	5	0
S	I	X	T	Y	→			3	1	4	8	6

1) States : Puzzle with letters and digits.

2) Initial state : Only letters present.

3) Successor function (operators) : Replace all occurrences of a letter by a digit not used yet.

4) Goal test : Only digits in the puzzle. Calculation is correct.

Solution for above example -

F = 2, O = 9,
R = 7, T = 8,
Y = 6, E = 5,
N = 0, I = 1,
X = 4, S = 3.

5) Goal test : This checks whether the state matches the goal configuration.

6) Path cost : Each step costs 1, so the path cost is the number of steps in the path.

Example 5 - Missionaries and Cannibals

• **Problem Statement :** There are 3 missionaries, 3 cannibals, and 1 boat that can carry upto two people on one side of a river.

Goal : Move all missionaries and cannibals across the river.

Constraint : Missionaries can never be out numbered by cannibals or either side of the river, or else the missionaries are killed.

1) States

- Number of missionaries, cannibals, and boats on the banks of a river.
- Illegal states : Missionaries are outnumbered by cannibals on either bank.

2) Initial states

- All missionaries, cannibals and boats are on one bank.

3) Successor function (operators)

- Transport a set of upto two participants to the other bank

{1 missionary} or {1 cannibal } or {2 missionaries } or {2 cannibals } or {1 missionary and 1 cannibal }

4) Goal test

- Nobody left on the initial river bank.

5) Path cost

- Number of crossings.
- Also known as "Goats and cabbage, Wolves and sheep", etc.

Example 6 - Water Jug Problem

Problem Statement : Given 3 jugs (9, 5 and 3 liters), a water pump and a sink, how do you get exactly 7 liters into the 9 litre jug.

1) State : (X, Y, Z) for liters in jugs 1, 2 and 3. Integers 0 to 9 are assigned to all possible permutations of 1, 2, 3.

Operations -

Empty jug, fill jug

Ex = fill (0, 5, 0)

2) Initial state : (0, 0, 0)

3) Goal state : (X, X, 7)

4) Solution sequence : (5, 0, 0 (0, 5, 0 (0, 0, 0 etc)

2.9.4 Real-World Problem Examples

Example 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 immignation procedures, seat quality, time of day, type of airplane, frequent-flyer mileage awards, and so on.

Example 2 - Travelling Sales Person Problem

Problem Statement : It is a touring problem in which each city must be visited exactly once. The aim is to find the shortest tour. The problem is known to be NP-hard.

1) States

- Locations/cities.
- Illegal states.

a) Each city may be visited only once.

b) Visited cities must be kept as state information.

2) Initial state

- Starting point.
- No cities visited.

3) Successor function (operators)

- Move from one location to another one.

4) Goal test

- All locations visited.
- Agent at the initial location.

5) Path cost

- Distance between locations.

In addition to planning trips for traveling sales persons, these algorithms have been used for tasks such as planning movements of automatic circuit-board drills, and for stocking machines on shop floors.

Example 3 - 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 discription 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

a) 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.

b) The overlapping between the cell component is avoided. Some sort of room is provided for connecting wires.

2) Channel routing

a) The cell components are fixed on a chip.

b) The channel routing task is to search a perticular route for each wire. The functional task is achieved through the gaps between the cells.

c) 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.

Example 4 - Robot Navigation

Problem Statement : Robot navigation is a generalization of the route-finding problem. Rather than a discrete set of routers, a robot can move in a continuous space with (in

principle) an infinite set of possible actions and states. For a circular robot moving on a flat surface the space is essentially two-dimensional. When the robot has arms and legs or wheels then it must also be controlled, the search space becomes many-dimensional.

1) States

- Locations.
- Position of actuators.

2) Initial state

- Start position (dependent on the task).

3) Successor function (operators)

- Movement, actions of actuators.

4) Goal test

- Task-dependent.

5) Path cost

- May be very complex.
- Distance, energy consumption.

Example 5 - Assembly Sequencing

Problem Statement : In assembly problems, the aim is to find an order in which to assemble the parts of some object. If the wrong order is choosen, there will be no way to add some part later in the sequence without undoing some of the work already done. Checking a step in the sequence for feasibility, is a difficult geometrical search problem, closely related to robot navigation.

1) States

- Location of components.

2) Initial state

- No components assembled.

3) Successor function (operators)

- Place component.

4) Goal test

- System fully assembled.

5) Path cost

Number of moves

Another example of assembly sequencing is protein design, in which the goal is to find a sequence of amino acids that with fold into a three-dimensional protein with the right properties to cure some disease.

Example 6 - Monkey - Banana Problem

Problem Statement : A monkey is in a room containing a box and a bunch of bananas. The bananas are hanging from the ceiling out of reach of the monkey. What sequence of actions will allow the monkey to get the bananas ? (The monkey is supposed to go to the box, puch it under the bananas, climb on top of it and grasp the bananas).

Goal Monkey to grasp all the bananas in hand.

1) States - the states can be, monkey is on the floor, monkey is on the box, monkey is under the bananas, the position of box, monkey has bananas.

2) Initial states - Monkey on the floor without bananas -

3) Successor function (operators) -

- Walk from (Monkey, Box)
- Push (Box)
- Climb (Monkey)
- Grasp (Monkey, Bananas)

4) Goal test

- Grasp (Monkey, Bananas, Yes)

5) Path cost

- The number of moves taken by monkey to get bananas.

Answer in Brief

1. *How will you measure the problem solving performance ?* **(Refer section 2.2)**
2. *What is application of BFS ?* **(Refer section 2.3)**
3. *State on which basis search algorithms are chosen ?* **(Refer section 2.1)**
4. *Evaluate performance of problem solving method based on DFS algorithm.* **(Refer section 2.3)**
5. *How a problem is formally defined ? List down the components of it.* **(Refer section 2.1)**
6. *Formulate the 8-puzzle problem using standard formulation and 8-queen problem using incremental and complete state formulation.* **(Refer section 2.8)**
7. *What do you mean by state space search ?* **(Refer section 2.2)**
8. *Discuss any 2 uniformed search methods with examples.* **(Refer section 2.3)**
9. *Write algorithm for BFS and DFS.* **(Refer section 2.3)**
10. *Write algorithm for BFS, when does one prefer it ? Also discuss A* algorithm.* **(Refer section 2.3)**

3 Heuristic Search Techniques

Syllabus

Generate-And-Test, Hill Climbing, Best-First Search, Problem Reduction, Constraint Satisfaction, Means-Ends Analysis, A and AO* search.*

Contents

3.1 The Searching Techniques

GTU : Summer-12,14,15,16,18,19,20. Winter-12,14,15,16,17,18,19

3.1.1 Introduction to Informed Search

- In the previous chapter 2 we have seen the concept of uninformed or blind search strategy. Now we will see more efficient search strategy, the informed search strategy. 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 wherein node expansion is done based on certain criteria.

3.1.2 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.

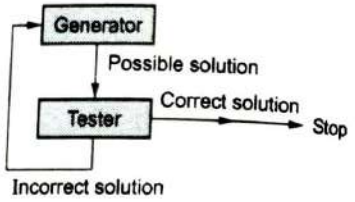


Fig. 3.1.1 Generate-And-Test

Following diagram illustrates the algorithm steps :

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

3.1.3 Best First Search Technique (BFS)

1) Search will start at root node.
2) The node to be expanded next is selected on the basis of an evaluation function, f(n).
3) 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

1. Use two ordered lists OPEN and CLOSED.
2. Start with the initial node 'n_1' and put it on the ordered list OPEN.
3. Create a list CLOSED. This is initially an empty list.
4. If OPEN is empty then exit with failure.
5. Select first node on OPEN. Remove it from OPEN and put it on CLOSED. Call this node n.
6. 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'.
7. Expand node 'n'. This will generate successors. Let the set of successors generated, be S. Create arcs from 'n' to each member of S.
8. 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

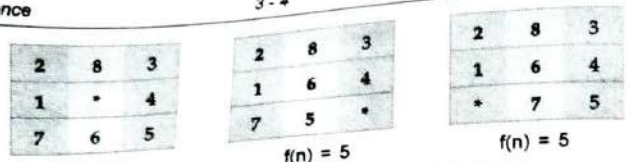
2	8	3
1	6	4
7	*	5

f(n) = 4

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

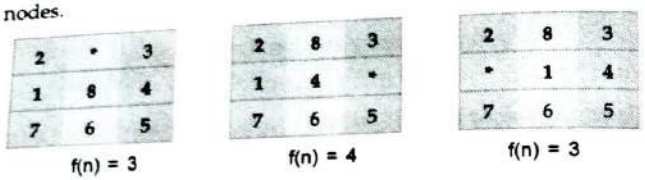
This will be the next step of the tree.



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

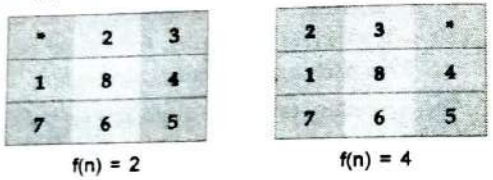
Step 3

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

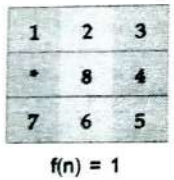


Step 4

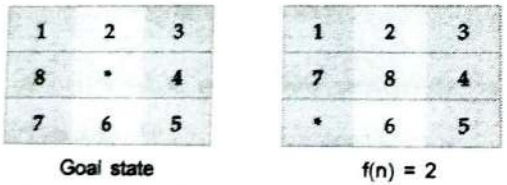
Here there is tie for f(n) value. We continue to expand node.



Step 5



Step 6



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

3.1.4 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

1) 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.
2) 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.
3) A heuristic function guide the search in an efficient way.
4) A heuristic function h(n) consider a node as input but it rely only on the state at that node.
5) h(n) = 0, if n is goal state.

3.1.5 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 inturn 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 devicing good heuristic function.

3.1.6 A* Search

The A* algorithm

1) A* is most popular form of best first search.
2) A* evaluates node based on two functions namely
   1) g(n) - The cost to reach the node 'n'.
   2) h(n) - The cost to reach the goal node from node 'n'.

These two function's costs are combined into one, to evaluate a node. New function f(n) is deviced as,

$f(n) = g(n) + h(n)$ that is,

$f(n)$ = Estimated cost of the cheapest solution through 'n'.

Working of A*

1) The algorithm maintains two sets.
   a) OPEN list : The OPEN list keeps track of those nodes that need to be examined.
   b) CLOSED list : The CLOSED list keeps track of nodes that have already been examined.
2) Initially, the OPEN list contains just the initial node, and the CLOSED list is empty. Each node n maintains the following : g(n), h(n), f(n) as described above.
3) Each node also maintains a pointer to its parent, so that later the best solution, if found, can be retrieved. A* has a main loop that repeatedly gets the node, call it 'n', with the lowest f(n) value from the OPEN list. If 'n' is the goal node, then we are done, and the solution is given by backtracking from 'n'. Otherwise, 'n' is removed from the OPEN list and added to the CLOSED list. Next all the possible successor nodes of 'n' are generated.
4) For each successor node 'n', if it is already in the CLOSED list and the copy there has an equal or lower 'f' estimate, and then we can safely discard the newly generated 'n' and move on. Similarly, if 'n' is already in the OPEN list and the copy there has an equal or lower f estimate, we can discard the newly generated n and move on.
5) If no better version of 'n' exists on either the CLOSED or OPEN lists, we remove the inferior copies from the two lists and set 'n' as the parent of 'n'. We also have to calculate the cost estimates for n as follows :

   Set g(n) which is g(n) plus the cost of getting from n to n;

   Set h(n) is the heuristic estimate of getting from n to the goal node ;

   Set f(n) is g(n) + h(n)
6) Lastly, add 'n' to the OPEN list and return to the beginning of the main loop.

Performance measurement for A*

1) Completeness : A* is complete and guarantees solution.

2) Optimality : A* is **optimal if h(n)** is admissible heuristic. **Admissible heuristic** means h(n) never over estimates the cost to reach the goal. Tree-search algorithm gives optimal solution if h(n) is admissible.

If we put extra requirement on h(n) which is consistency also called as monotonicity then we are sure expect the optimal solution. A heuristic function h(n) is said to be consistent, if, for every node 'n' and every successor 'ns' of 'n' generated by action 'a', the estimated cost of reaching the goal from 'n' is not greater than the step cost of getting to 'ns'.

$h(n) \leq \text{cost}(n, a, ns) + h(ns)$

A* using graph-search is optimal if h(n) is consistent.

Note : The sequence of nodes expanded by A* using graph-search is in, increasing order of f(n). Therefore the first goal node selected for expansion must be an optimal solution because all latter nodes will be either having same value of f(n) (same expense) or greater value of f(n) (more expensive) than currently expanded node.

A* never expands nodes with $f(n) > C^*$ (where C^* is the cost of optimal solution).

A* algorithm also do pruning of certain nodes.

Pruning means ignoring a node completely without any examination of that node, and thereby ignoring all the possibilities arising from these node.

3) Time and space complexity : If number of nodes reaching to goal node grows exponentially then time taken by A* eventually increases.

The main problem area of A* is memory, because A* keeps all generated nodes in memory.

A* usually runs out of space long before it runs out of time.

A* optimality proof

- Assume A* finds the (sub optimal) goal G2 and the optimal goal is G.
- Since h is admissible

  $h'(G2) = h'(G) = 0$
- Since G2 is not optimal

  $f(G2) > f(G)$
- At some point during the search, some node 'n' on the optimal path to G is not expanded.

We know,

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

3.1.7 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*).

3.1.7.1 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.

Its space complexity is O(bd).

Its 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.

3.1.7.2 MA*

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

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

1) **If expands the best leaf until memory is full.**
2) At this point it cannot add new node to the search tree without dropping an old one.
3) **It always drops the node with highest f-value.**
4) 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.
5) 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.

3.1.8 AO* Search and Problem Reduction Using AO*

When a problem can be divided in a set of sub problems, where each sub problem can be solved seperately 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

1) The decomposition of the problem or problem reduction generates AND arcs.
2) One AND arc may point to any number of successor nodes.
3) 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.

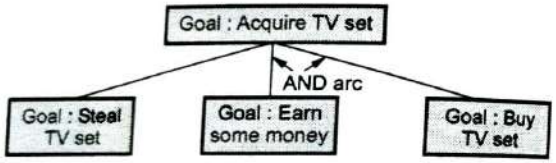


Fig. 3.1.2 [AND-OR graph example]

4) AO* is a best-first algorithm for solving problems represented as a cyclic AND/OR graphs problems.
5) An algorithm to find a solution in an AND-OR graph must handle AND area appropriately.

The comparative study of A* and AO*

1) Unlike A* algorithm which used two lists OPEN and CLOSED, the AO* algorithm uses a single structure G.
2) G represents the part of the search graph generated so far.
3) 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.

4) 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.
5) AO* algorithm serves as the estimate of goodness of a node.
6) A* algorithm cannot search AND-OR graphs efficiently.
7) AO* will always find minimum cost solution.

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

AO* algorithm

1) Initialize the graph to start node.
2) Traverse the graph following the current path accumulating nodes that have not yet been expanded or solved.
3) 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.
4) If 'f' is 0 then mark the node as SOLVED.
5) Change the value of 'f' for the newly created node to reflect its successors by back propagation.
6) Wherever possible use the most promising routes and if a node is marked as SOLVED then mark the parent node as SOLVED.
7) If starting node is SOLVED or value greater than FUTILITY, stop, else repeat from 2.

3.1.9 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 **metalevel state space.** Each state in a metalevel 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

1) In A* algorithm it maintains internal state which consists of the current search tree.
2) 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.

3) 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.

3.1.10 Heuristic Functions and Their Nature

3.1.10.1 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,

$$N+1 = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d$$

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.

3.1.10.2 Designing Heuristic Functions for Various Problems

Example 3.1.1 *8-puzzle problem*

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

7	2	4
5		6
8	3	1

Start state

	1	2
3	4	5
6	7	8

Goal state

1) The average solution cost for a randomly generated 8-puzzle instance is about 22 steps.

2) 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.

3) An exhaustive search to depth 22 would look at about $3^{22} \approx 3.1 \times 10^{10}$ states.

4) 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.

5) If we use A*, we need a heuristic function that never overestimates the number of steps to the goal.

6) 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 $h_1 = 8 \cdot h_1$, 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.

Example 3.1.2 *Blocks world problem.* **GTU : Winter-17, Marks 3**

Solution :

Heuristic for Blocks World

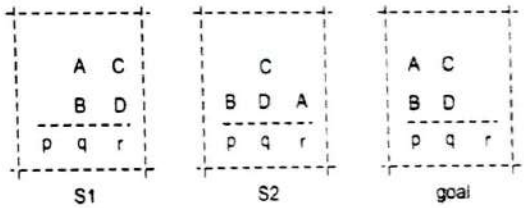
A function that estimates the cost of getting from one place to another (from the current state to the goal state.)

Used in a decision process to try to make the best choice of a list of possibilities (to choose the move more likely to lead to the goal state.)

Best move is the one with the least cost.

The "intelligence" of a search process, helps in finding a more optimal solution.

h1(s) = Number of places with incorrect block immediately on top of it.



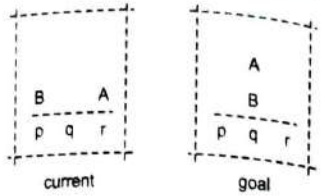
h1 (S1) = 3 h1 (S2) = 1

A more informed heuristic

Looks at each bottom position, but takes higher positions into account. Can be broken into 4 different cases for each bottom position.

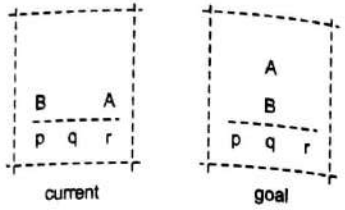
1. Current state has a blank and goal state has a block.

Two blocks must be moved onto q. So increment the heuristic value by the number of blocks needed to be moved into a place.



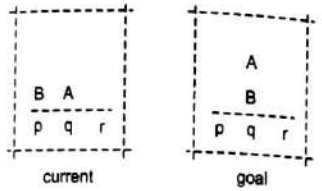
2. Current state has a block and goal state has a blank.

One block must be removed from p. So increment the heuristic value by the number of blocks needed to be moved out of a place.



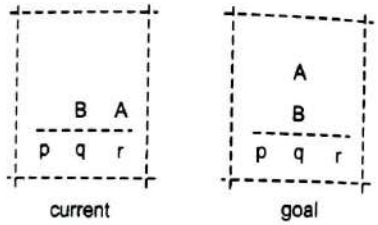
3. Both states have a block, but they are not the same.

On place q, the blocks don't match. The incorrect block must be moved out and the correct blocks must be moved in. Increment the heuristic value by the number of these blocks.

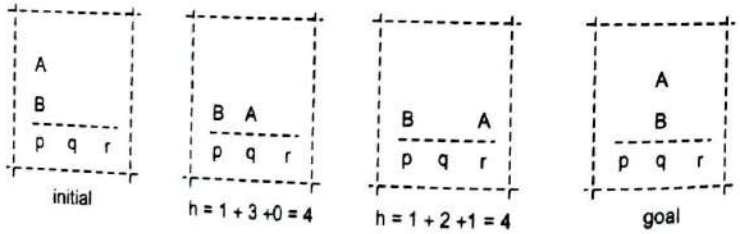


4. Both states have a block, and they are the same.

B is in the correct position. However, the position above it is incorrect. Increment the heuristic value by the number of incorrect positions above the correct block.



Example



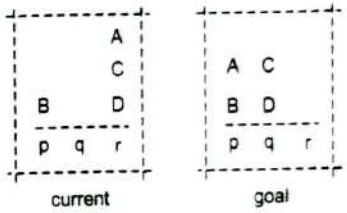
Know the second state leads to the optimal solution, but not guaranteed to be chosen first. Can the heuristic be modified to decrease its value? Yes, since the block on p can be moved out and put on q in the same move. Need to compensate for this overcounting of moves.

For a given (nonempty) place, find the top-most block. Look for its position in the goal state. If it goes on the bottom row and the place is currently empty, the block can be moved there, so decrement heuristic by one.

Now the second state's heuristic will be 3, and it's guaranteed to be chosen before the other one. A little like a one move look ahead.

If the top-most block doesn't go on bottom row, but all blocks below it in the goal state are currently in their correct place, then the block can be moved there, so decrement heuristic by one.

Example



p: case 4: +1

q: case 1: +2

r: case 2: +3

A can be put on B: -1

so h=5

optimal number of moves is 4

Example 3.1.3 *Travelling salesman problem.* **GTU : Summer-18, Marks 3**

Solution : Heuristic function, h(n) is defined as,

h(n) = estimate cost of the cheapest path from node 'n' to a goal state.

Heuristic function for Travelling salesman problem :

- Travelling salesman problem (TSP) is a routing problem in which each city must be visited exactly once. The aim is to find the shortest tour.
- The goal test is, all the locations are visited and agent at the initial location.
- The path cost is distance between locations.
- As a heuristic function for TSP, we can consider the sum of the distances travelled so for. The distance value directly affects the cost therefore, it should be taken into calculation for heuristic function.

Heuristic function for 8 puzzle problem.

- The objective of the 8 puzzle is to slide the tiles horizontally or vertically into the empty space until the configuration matches the goal configuration.
- 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 $h_1 = 8 \cdot h_1$, 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.

Example 3.1.4 *Tic Tac Toe problem.* **GTU : Summer-18, Marks 3**

Solution :

(ii) **Heuristic for Tic Tac Toe problem**

This problem can be solved all the way through the minimax algorithm but if a simple heuristic/evaluation function can help save that computation. This is a static evaluation function which assigns a utility value to each board position by assigning weights to each of the 8 possible ways to win in a game of tic-tac-toe and then summing up those weights.

One can evaluate values for all of 9 fields, and then program would choose the field with highest value.

For example,

Every Field has several WinPaths on the grid. Middle one has 4 (horizontal, vertical and two diagonals), corners have 3 each (horizontal, diagonal and one vertical), sides have only 2 each (horizontal and vertical). Value of each Field equals sum of its WinPaths values. And WinPath value depends on its contents:

- Empty : [| |] - 1 point
- One symbol: [X| |] - 10 points // can be any symbol in any place
- Two different symbols: [X|O|] - 0 points // they can be arranged in any possible way
- Two identical opponents symbols: [X|X|] - 100 points // arranged in any of three ways
- Two identical "my" symbols: [O|O|] - 1000 points // arranged in any of three ways
- This way for example beginning situation has values as below :

3	2	3
2	4	2
3	2	3

3.1.10.3 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 $f(n) = c^*$)

3.1.10.4 Admissible Heuristic Functions

1) A problem with fewer restrictions on actions is called a **relaxed problem.**
2) 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.
3) As the derived heuristic is an exact cost for the relaxed problem, therefore it is consistant.
4) If problem definitions are written in formal languages then it is possible to construct relaxed problem automatically.
5) 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.
6) 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.

   We can compute an admissible heuristic function 'h' for each complete state encountered during a search simply by looking up the corresponding subproblem configuration in the database.
7) 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.

3.1.11 Learning Heuristic from Experience

1) A heuristic function h(n) is supposed to estimate the cost of a solution beginning from the state at node n. Therefore it is really difficult to design h(n).
2) One solution is to devise relaxed problems for which an optimal solution can be found.
3) 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.

3.2 Local Search Algorithms and Optimization Problems

GTU : Summer-12,13,14,15,16,17,18,19,20, Winter-12,14,15,16,17,18,19

- The search algorithms we have seen so far, more often concentrate on path through which the goal is reached. But if the problem does not demand the path of the solution and it expects only the final configuration of the solution then we have different types of problem to solve.
- Following are the problems where only solution state configuration is important and not the path which has arrived at solution.
  1) 8-queen (where only solution state configuration is expected).
  2) Integrated circuit design. 3) Factory-floor layout.
  4) Job-shop scheduling. 5) Automatic programming.
  6) Telecommunication network optimization.

  7) Vehicle routing. 8) Portfolio management.

If we have such type of problem then we can have another class of algorithms, the algorithms that do not worry about the paths at all. These are local search algorithms.

3.2.1 Local Search Algorithms

- They operate using single state. (rather than multiple paths).
- They generally move to neighbours of the current state.
- There is no such requirement of maintaining paths in memory.
- They are not "Systematic" algorithm procedure.
- The main advantages of local search algorithm are

1) They use very little and constant amount of memory.
2) They have ability to find resonable solution for infinite state spaces (for which systematic algorithms are unsuitable).

Local search algorithms are useful for solving **pure optimization problems.** In pure optimization problems main aim is to find the best state according to required objective function.

Local search algorithm make use of concept called as **state space landscape.** This landscape has two structures -

1) Location (defined by the state)
2) Elevation (defined by the value of the heuristic cost function or objective function).

- If elevation corresponds to the cost, then the aim is to find the **lowest valley - (a global minimum).**
- If elevation corresponds to the objective function then aim is to find the **highest peak - (a global maximum).**
- Local search algorithms explore the landscape.

Performance measurement

- Local search is complete i.e. it surely finds goal if one exists.
- Local search is optimal as it always find global minimum or maximum.

Local search representation

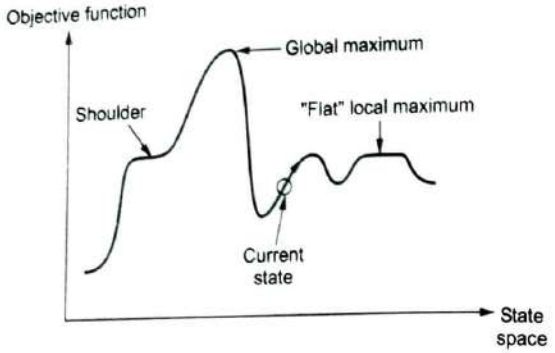


Fig. 3.2.1 Local search

3.2.2 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.

3.2.2.1 Algorithm for Hill Climbing

The algorithm for hill climbing is as follows : -

1) Evaluate the initial state. If it is goal state quit, otherwise make current state as initial state.
2) Select a new operator that could be applied to this state and generate a new state.
3) 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.
4) If the current state is goal state or no new operators are available, quit. Otherwise repeat from 2.

3.2.2.2 Problems with Hill Climbing

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

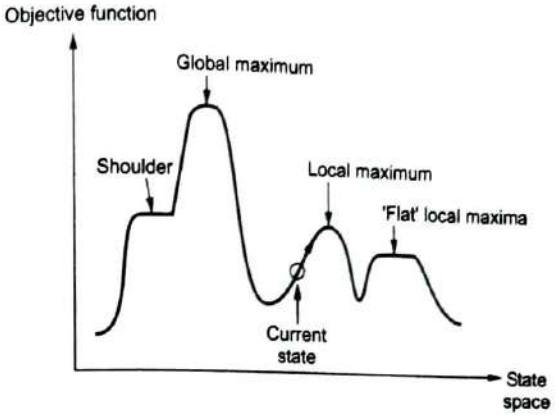


Fig. 3.2.2 Various stages in hill climbing search

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. 3.2.3]**

Fig. 3.2.3 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. 3.2.4]**

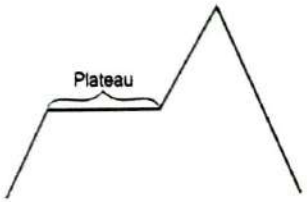


Fig. 3.2.4 Plateau

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. 3.2.5]**

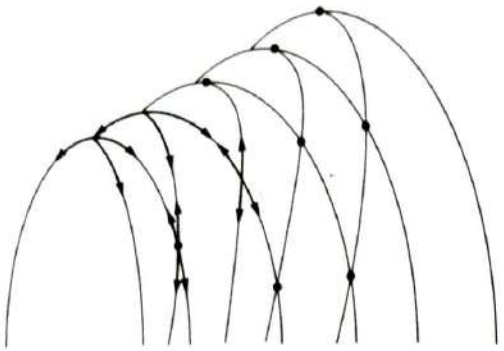


Fig. 3.2.5 Ridges

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.

3.2.2.3 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.

3.2.2.4 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.

3.2.2.5 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.

3.2.2.6 Variations of Hill Climbing

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

1) Stochastic hill climbing : Chooses at random from among the uphill moves ; the probability of selections can vary with the steepness of the uphill move.

2) 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.

3) 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.

4) 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

1. Evaluate the initial state

2. If it is goal state then quit otherwise make the current state this initial state and proceed;
3. 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.
4. 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.

3.2.3 Simulated Annealing Search

1) In simulated annealing, initially the whole space is explored.
2) This is a variation of hill climbing.
3) This avoids the danger of being caught on a plateau or ridge and makes the procedure less sensitive to the starting point.
4) Simulated annealing is done to include a general survey of the scene, to avoid climbing, false foot hills.
5) There are two additional changes -
   i) Rather than creating maxima, minimisation is done.
   ii) The term objective function is used rather than heuristic.
6) This concept is taken from physical annealing where physical substances are melted and then gradually cooled until some solid state is reached. In physical annealing the goal is to produce a minimal-energy state. The annealing schedule

states, that if the temperature is lowered sufficiently slowly, then the goal will be attained. ΔE is called the change in the value of the objective function.

7) It becomes clear that in this algorithm we have valley descending rather than hill climbing. The probability that the metal will jump to a higher energy level is given by $P = \exp^{8-E/kT}$ where k is Boltzmann's constant.

The algorithm

1) Start with evaluating the initial state.
2) Apply each operator and loop until a goal state is found or till no new operators left to be applied as described below : -
   i) Set T according to an annealing schedule.
   ii) Select and apply a new operator.
   iii) Evaluate the new state. If it is a goal state quit.

   ΔE = Val (current state) – Val (new state)

- If $\Delta E < 0$ then this is the new current state.

Else find a new current state with probability $e - E / kT$.

3.2.4 Local Beam Search

1) In the local beam search, instead of single state in the memory k states are kept in the memory.
2) In local beam search states are generated in random fashion.
3) A successor function plays an important role by generating successor of all K states.
4) If any one successor state is goal then no further processing is required.
5) In other case i.e. if goal is not achieved, it observes the K best successors from the list of all successor and process is repeated.
6) At first glance the random parallism is achieved in the sequence by a local beam search with K states.
7) In a random-restart search every single search activity run independently of others.
8) To implement local search, threads are used. The K parallel search threads carry useful information.
9) The algorithm work on principle of successful successors. If one state generate efficient/goal reaching successor and other K-1 state generate poor successor, then in this situation the successful successors leads all other states.
10) The algorithm drops/leaves the unsuccessful search and concentrate on successful search.

Limitation of local beam search

1) The local beam search has limitation of, lack of variation among the K states.
2) If the state concentrate on small area of state space then search becomes more expensive.

3.2.5 Stochastic Beam Search

1) Its one of the flavour of local beam search, which is very similar to that of stochastic hill climbing.
2) It resolves the limitation of local beam search.
3) Stochastic beam search concentrate on random selection of K successor instead of selecting k best successor from candidate successor.
4) The probability of random selection is increasing function of its success rate.
5) A stochastic beam search is very similar to natural selection, where the child (successor) of a parent state is eugenic (good production) according to its success rate (fitness). Thus the next generation is also very much powerful.

3.2.6 Genetic Algorithms

1) Evolutionary pace of learning algorithm is genetic algorithm. The higher degree of eugency can be achieved with new paradigm of AI called a genetic algorithms.
2) A genetic algorithm is a rich flavour of stochastic beam search.
3) In the genetic algorithm two parent states are combined together by which a good successor state is generated.
4) The analogy to natural selection is the same as in stochastic beam search, except now we are dealing with sexual rather than asexual reproduction.

3.2.6.1 Term used in Genetic Algorithm

1) Population : Population is set of states which are generated randomly.

2) Individual : It is a state or individual and it is represented as string over a finite alphabet.

Example - A string of 0s and 1s.

For example - In 8-queen all states are specified with the position of 8-queens. The memory required is

$8 \times \log_2 8 = 24$ bits. Each state is represented with 8 digits.

3) Fitness function : It is a evaluation function. On its basis each state is rated. A fitness function should return higher values for better states. For the state, the probability of being chosen for reproducing is directly proportional to the fitness score.

In 8-queen problem the fitness function has 28 value for number of nonattacking pairs.

4) Crossover : Selection of state is dependent on fitness function. If fitness function value is above threshold then only state is selected otherwise discarded. For each state, pairs are divided, that division point or meeting point is called crossover point, which is chosen in random order from the positions in the string.

5) Mutation : Mutation is one of the generic operator. Mutation works on random selections or changes. For example mutation select and changes a single bit of pattern switching 0 to 1 or 1 to #.

6) Schema : The schema is a substring in which position of some bit can be unspecified.

3.2.6.2 Working of a Genetic Algorithm

Input : 1) State population (a set of individuals)
2) Fitness function (that rates individual).

Steps

1) Create an individual 'X' (parent) by using random selection with fitness function 'A' of 'X'.
2) Create an individual 'Y' (parent) by using random selection with fitness function 'B' of 'Y'.
3) Child with good fitness is created for **X + Y.**
4) For small probability apply mutate operator on child.
5) Add child to new population.
6) The above process is repeated until child (an individual) is not fit as specified by fitness function.

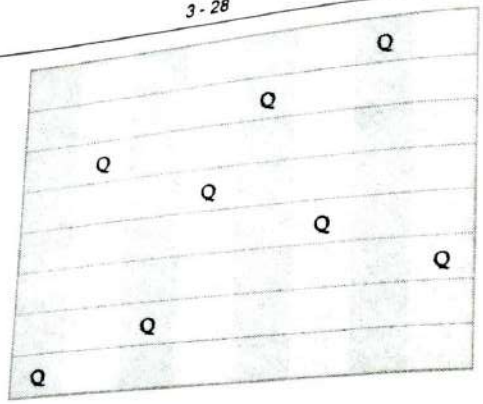
Genetic algorithm example

Example : 8-queens problem states.

States :

Assume each queen has its own column, represent a state by listing a row where the queen is in each column (digits 1 to 8).

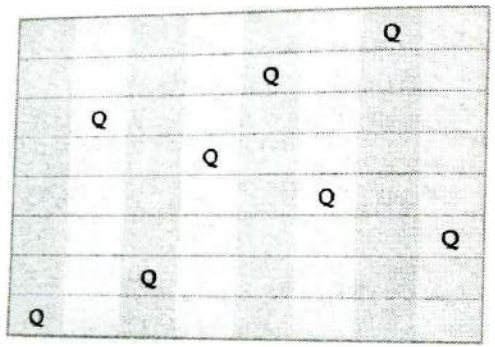
For example, the state below will be represented as 16257483.



3.2.6.3 Example : 8 Queens Problem Fitness

Fitness function : Instead of – h as before, use the number of non-attacking pairs of queens. There are 28 pairs of different queens, smaller column first, all together, so solutions have fitness 28. (Basically, fitness function is 28 – h)

For example, fitness of the state below is 27 (queens in columns 4 and 7 attack each other).



Example : 8 queens problem crossover.

Choose pairs for reproduction (so that those with higher fitness are more likely to be chosen, perhaps multiple times).

For each pair, choose a random crossover point between 1 to 8, say 3.

Produce offspring by taking substring 1-3 from the first parent and 4 - 8 from the second (and vice versa). Apply mutation (with small probability) to the offspring.

Importance of representation

Parts we swap in crossover should result in a well-informed solution (and in addition better be meaningful).

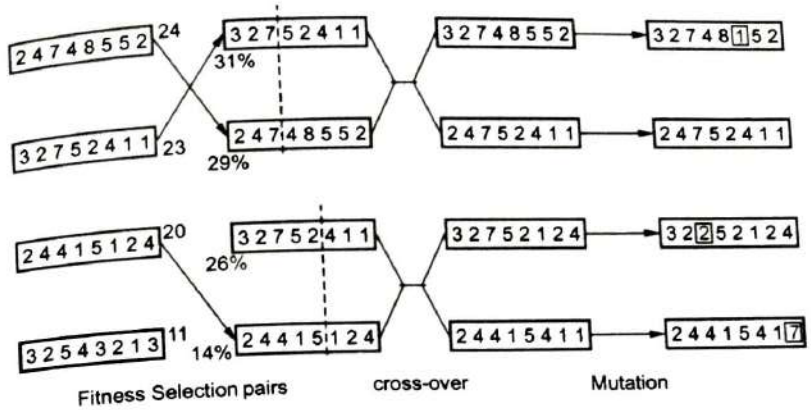
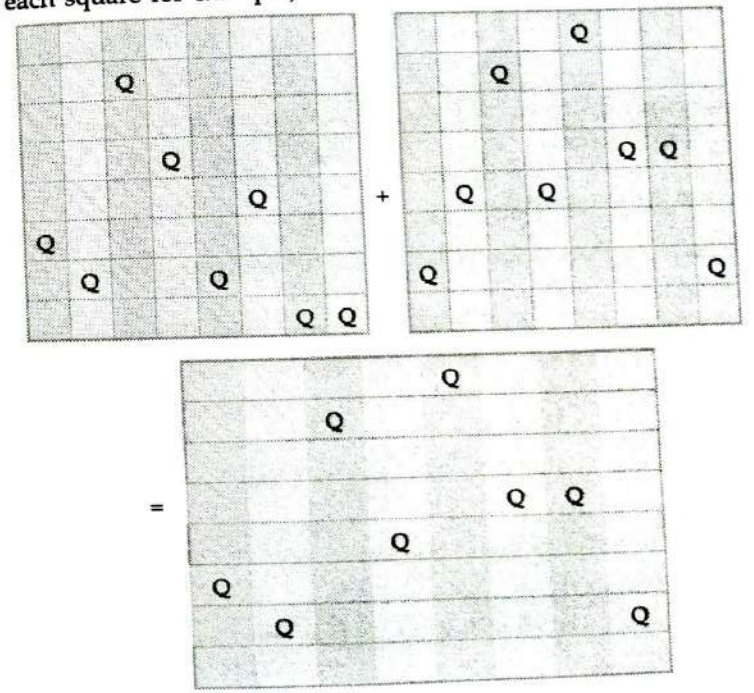


Fig. 3.2.6 8 Queen problem crossover

Consider what would happen with binary representation (where position requires 3 digits).

Also chosen representation reduces search space considerably (compared to representing each square for example).



3.2.6.4 Optimization using Genetic Algorithm

1) Genetic algorithm is biological model of intelligent evolution. It generates the population of competing children.
2) The poor candidate solution will be vanished, as genetic algorithm generates best child.

Thus, in practice genetic algorithm is most promising survive and reproduction technique by constructing new optimized solution. Thus it is optimization oriented technique.

Application of genetic algorithm on optimization.

1) Circuit layout.
2) Job-shop scheduling.

3.3 Local Search in Continuous Spaces

As we know that environment can be descrete or continuous, but the most real-world environment are continuous.

1) It is very difficult to handle continuous state space. The successor for real-world problem are many infinite states.
2) Origin of local search in continuous spaces lies in Newton and Leibnitz in the 17^{th} century.
3) Optimal solution for given problem in continuous spaces can be found with "Local search techniques".

3.3.1 Search and Evaluation Theory

1) Search is basic problem solving technique. Basically it is always related with evolution theory.
2) Charles Darwin is father of evolutionary theory. The theory was based on the origin of species by means of natural selection (1859).
3) The variations (mutations) are well known attributes of reproduction and best features are preserved in next generation in proper propagation.
4) The qualities are inherited or modified. This fact was not associated with Darwin theory.
5) Gregor Mendel (1866) theory found the fact of inheritance. He performed artificial fertilization on peas.
6) DNA module structure was identified by Watson and Crick.

A → Adenin, G - Guanine,

T → Thymine, C - Cytosine

7) The key difference between stochastic beam search and evolution is that successors are generated from multiple organisms (states) rather than one organism (state).
8) The theory of evolution is very much rich than genetic algorithm.
9) The process of mutations involves duplication, reversals and motion of large group of DNA.
10) Most important is the fact that the genes themselves encode the mechanisms whereby the genome is reproduced and translated into an organism. In genetic algorithms, those mechanisms are a separate program that is not represented within the strings being manipulated.
11) French naturalist Jean Lamarck (1809) proposed a theory of evolution whereby traits acquired by adaptation during an organism's lifetime would be passed on to its offspring. Such a process would be effective, but does not seem to occur in nature.
12) James Baldwin (1896) proposed a superficially similar theory : - that behavior learned during an organism's lifetime could accelerate the rate of evolution.

For example -

Suppose we want to place **three new airports anywhere in India,** such that the sum of squared distances from each city to its nearest airport is minimized.

i) The state space, is defined by the co-ordinates of the airports :

(x_1, y_1), (x_2, y_2) and (x_3, y_3).

ii) This is a six-dimensional space : We also say that states are defined by six variables. (In general, states are defined by an n-dimensional vector of variables, x).

iii) Moving around in this space corresponds to moving one or more of the airports on the map.

iv) The objective function $f(x_1, y_1, x_2, y_2, x_3, y_3)$ is relatively easy to compute for any particular state, once we compute the closest cities, but rather tricky to write down in general.

3.3.2 Problems Associated with Local Search

- Local search methods suffer from local maxima, ridges and plateaux in continuous state spaces just as much as in discrete spaces. Random restarts and simulated annealing can be used and are often helpful. High-dimensional continuous spaces are, however, big places in which it is easy to get lost.

3.3.3 Constrained Optimization Problem

- An optimization problem is constrained if solutions must satisfy some hard constraint like sites to be inside India and on dry land (rather than in the middle of rivers). The difficulty of constrained optimization problems depends on the nature of the constraints and the objective function. The best-known category is that of linear programming problems, in which constraints must be linear inequalities forming a convex region and the objective function is also linear. Linear programming problems can be solved in time which is polynomial in the number of variables.

3.4 Handling Unknown Environments

- Earlier algorithms we have seen are offline search algorithms. They compute a complete solution before setting foot in the real world, and then execute the solution without considering new percepts. Now we will consider how to handle dynamic or semidynamic environments.

3.4.1 Online Search Agents

1) The online search agent works on the concept of interleaving of two tasks namely computation and action. In the real-world, the agent perform an action and it observes the environment and then work out on the next action.
2) If an environment is dynamic or semidynamic or stochastic the online search work in best manner.
3) An agent needs to pay a penalty for lengthy computation and for sitting around.
4) An offline search is costly. For offline search we need to have proper planning which consider all related information. In online search there is no need of planning just see what happens and act accordingly.

   Example - Chess moves.
5) In online search the states and actions are unknown to the agent. Thus it has an exploration problem. (it needs to explore the world).
6) In an online search the actions are used to predict next states and actions.
7) For predicting next states and actions computations are performed.

For example -

- Consider example of robot who builds a map between 2 locations say LOC 1 and LOC 2. For building a map robot needs to explore the world so as to reach to LOC 2 from LOC 1. Parallely it will build the map.

- A natural real world example of exploration problem and online search is a baby trying to understand the world around it. It is baby's online search process. A baby tries an action and go in certain state. It keeps on doing this, gathering new experience and learning from it.

3.4.2 Solving Online Search Problems

1) The purely computation process cannot handle an online search problem properly. Agent needs to execute actions for solving problem.
2) The agent has knowledge about -
   a) ACTIONS (s) which returns a list of actions allowed in state s.
   b) The step-cost function cost (s_1, a, s') (note that cost function cannot be used until the agent knows that s' is the outcome).
   c) GOAL-TEST (s)
3) It is not possible for agent to access the successors of a state. In fact the agent try related actions, in that state.
4) As basic objective of an agent is to minimize cost, the goal state must be found with minimum cost.
5) The another possible goal can be to explore the entire environment. The total path cost is cost of path across which the agent travels.
6) In the online algorithm we can find the competitive ratio means shortest path. The competitive ratio must be very small. But in reality many a time competitive ratio is infinite.
7) If the online search is with irreversible action then it will reach to a dead-end state and from that state it can't achieve goal state.
8) We must build an algorithm which does not explore dead-end path. But accidently or in reality an algorithm can avoid dead ends in all state spaces.
9) Consider two goal state space as shown in Fig. 3.4.1.

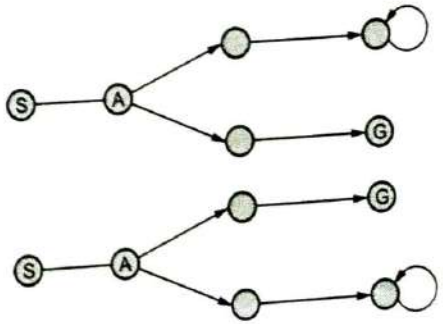


Fig. 3.4.1 Two goal state space which may lead to a dead end

In the Fig. 3.4.1 online search algorithm visit states S and A, both the state spaces are identical so it must be explored in both cases. One link is to goal state (G) and other lead to dead end in both states spaces. It is an

example of an adversary arguments. The exploration of state space is in an adversary manner. One can put the goals and dead ends likewise.

3.4.3 Online Local Search

Hill climbing search

1) Hill climbing search has the property of locality in its node expansions because it keeps just one current state in memory, hill-climbing search is already an online search algorithm.
2) Unfortunately, it is not very useful in its simplest form because it leaves the agent sitting at local maxima with nowhere to go.
3) Moreover, random restarts cannot be used, because the agent cannot transport itself to a new state.

Random walk

1) Instead of random restarts, one might consider using a random walk to explore the environment.
2) A random walk simply selects at random one of the available actions from the current state; preference can be given to actions that have not yet been tried.
3) It is easy to prove that a random walk will eventually find a goal or complete its exploration, provided that the space is finite.

Hill climbing with memory

1) Augmenting hill climbing with memory rather than randomness turns out to be a more effective approach.
2) The basic idea is to store a "current best estimate", H(S) the cost to reach the goal from each state that has been visited.
3) H(S) starts out being just the heuristic estimate h(s) and is updated as the agent gains experience in the state space.

3.4.4 Learning Real Time A*

1) An agent can use a scheme called learning real-time A* (LRTA*).
2) It builds a map of the environment using the result table. It updates the cost estimate for the state it has just left and then chooses the "apparently best" move according to its current cost estimates.
3) One important detail is that actions that have not yet been tried in a state S are always assumed to lead immediately to the goal with the least possible cost,

namely h(s). This optimism under uncertainty encourages the agent to explore new, possibly promising paths.

4) An LRTA* agent is guaranteed to find a goal in any finite, safely explorable environment.
5) It is not complete for infinite state spaces. There are cases where it can be led infinitely astray. It can explore an environment of n states in $O(n^2)$ steps in the worst case.

3.4.5 Learning in Online Search

At every step online search agent can learn various aspects of current situation.

The agents learn a "map" of the environment - more precisely, the outcome of each action in each state-simply by recording each of their experiences.

The local search agents acquire more accurate estimates of the value of each state by using local updating rules.

By learning, the agent can do better searching.

3.5 Constraint Satisfaction Problems

GTU : Summer-12, 13, 14, 16, 18, Winter-14, 15

- 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.

3.5.1 Constraint Satisfaction Problems - The Concept

1. Constraint satisfaction problem has various states and goal test, a tranditional problem has been converted into standard structured and very simple "representation.".
2. 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.
3. The structure of the problem is represented in different form such as, standard representation of the goal test.
4. The revealed structured is very efficient in many ways such as,
   i) Problem decomposition.
   ii) To understand structure of problem and the diffculty of solving it and their connection.

5. If the problem is treated as CSP we have many advantages, as discussed below.

   i) 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.

   ii) We can develop effective generic heuristic that require no additional domain specific expertise.

   iii) The structure of the constraint graph can be used to simplify the solution process in some cases giving exponential reduction in complexity.

3.5.2 Formal Definition of Constraint Satisfaction Problems

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

Consider graph colouring problem as shown in Fig. 3.5.1. 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)}

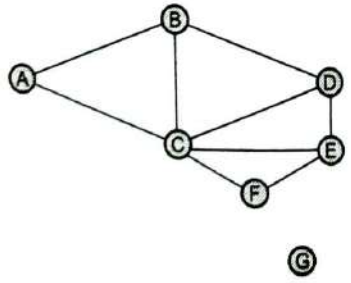


Fig. 3.5.1 Graph colouring problem

3.5.3 Examples of CSP

3.5.3.1 Map Colouring Problem

1. The problem is to colour the regions of a given map such that no 2 adjacent regions have the same colour. [Refer figure 3.5.2]
2. The regions in the map are the variables and the set of possible colours for the regions is the domain.
3. 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 = {red, green, blue}
3. **Constraints :** Adjacent regions must have different colors.

Example : N ≠ NW

Note :

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

Example Map (See Fig. 3.5.2 on next page).

3.5.3.2 Other Examples of CSP

1. N-queens puzzle.
2. Jobshop 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. Transporation scheduling.
3. Hardware configuration.
4. Spreadsheets.
5. Factory scheduling.
6. Floor planning.

3.5.4 Incremental Formulation for CSP

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

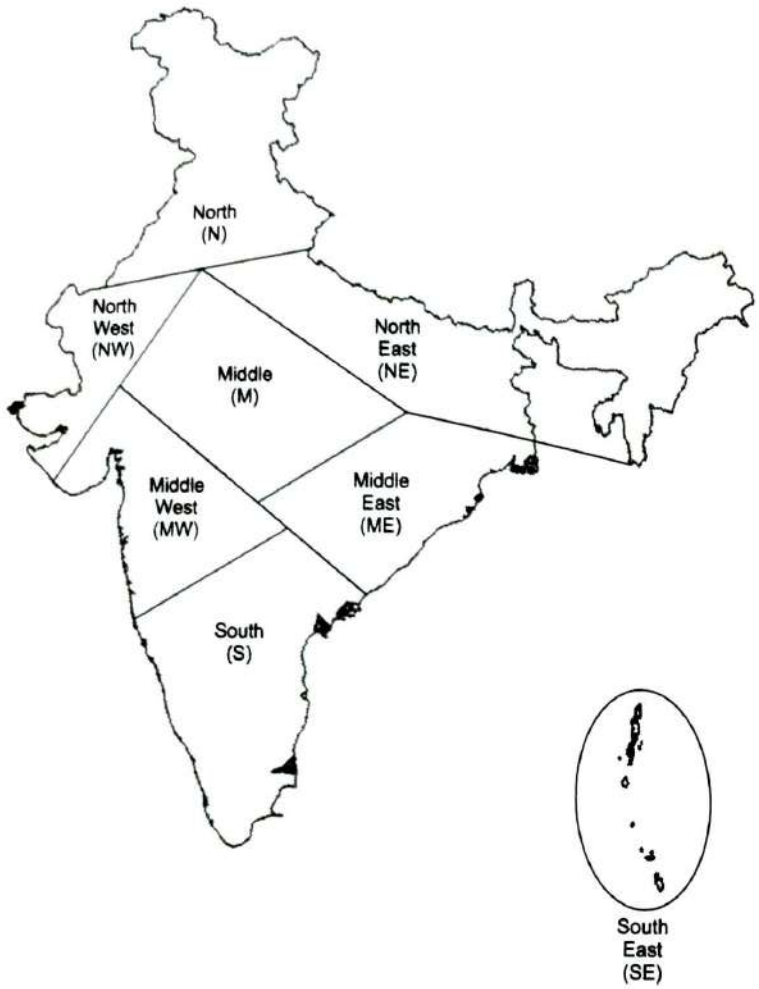


Fig. 3.5.2 Map colouring problem

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.

3.5.5 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.

3.5.6 Variations in CSPs

1. 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 $Q_1, ..., Q_8$ are the positions of each queen in colums 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.
2. 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.
3. 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.
4. Discrete variables can also have infinite domains-for example, the set of integers or the set of strings.
5. Constraints satisfaction problems with continuous domains are very common in the real world and are widely studies 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 maneuver 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.

3.5.7 Constraints in CSPs

3.5.7.1 Properties of Constraints

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

1. Constraints may specify partial infomation; constraint need not uniquely specify the values of its variables.
2. 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 ;
3. Constraints are declarative ; they specify what relationship must hold without specifying a computational procedure to enforce that relationhip.
4. 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.
5. Constraints are really independent; typically constraints in the constraint store (i.e. collection of constraints) share variables.

3.5.7.2 Types of Constraints in CSPs

1. **Unary constraint -**

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

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

2. **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 not have same colour.

3. **Higher-order constraints -**

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

For example - The cryptarithmatic problem. (A CSP problem having high order constraints).

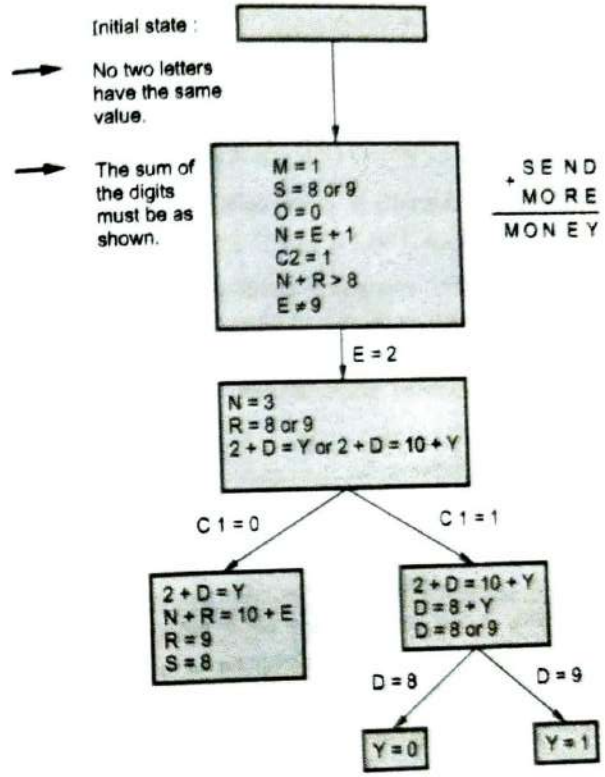


Fig. 3.5.3 The cryptarithmatic problem

3.5.8 General Algorithm for finding Solution in CSP

1. Propagate available constraints -

i) Open all objects that must be assigned values in a complete solution.

ii) 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.

2. If union of constraints discovered above defines a solution then return solution.
3. If union of constraints discovered above defines a contradiction then return failure.

4. Make a guess in order to proceed. Repeat until a solution is found or all possible solutions exhausted;
   i) Select an object with a no assigned value and try to strengthen its constraints.
   ii) 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 constrain graph has,

1. 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.)
2. Arcs as a constraints (For example - In graph-colouring problem the arc will denote that no two adjacent vertices will have same colour.)

- **Advantages of Constraint Graph**

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

3.5.9 Backtracking Search

- If we apply BFS to generic CSP problems then we can notice that the branching factor at the top level is 'nd', 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. Therfore a tree with 'n! * d^n' leaves is generated even though there are only 'd^n' possbile complete assignments.

3.5.9.1 Basic Steps in Backtracking Search

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

3.5.9.2 Backtracking Search Algorithm

1. Consider a CSP problem.
2. Apply backtracking search. If backtracking search successful returns a solution else, a failure state which return a procedure recursive-backtracking.
3. Procedure recursive-backtracking starts with empty set and takes input as CSP problem.

   If complete assignement possible or assignment done then return actual assignment.
4. Variable is assigned a specific value.
5. The relative constraint is a set which is taken as input.
6. If value is complete and consistent according to constraints then assign value to variable, add that to list.
7. Call the recursive backtracking until result or failure is reached.
8. Every time recursive-backtracking sustains result (i.e. assignment.)
9. If result is failure then remove variable with specific value from assignment. It will return failure status.

3.5.9.3 Limitations of Backtracking

1. A success function is generated with above algorithm. But backtracking is not effective for very large problems.
2. 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 : -

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

3.5.10 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.

3.5.11 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)

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

2. Degree heuristic

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

3. Least constraining value

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

3.5.12 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.**

3.5.12.1 Forward Checking (FC)

1. Forward checking is one of the potential technique which uses constraints more effectively during search.
2. It removes values in neighbouring unassigned variables domain that conflict with assigned variable.
3. Forward checking uses MRV to select assigned variable.
4. Backtracking search is performed if failure occurs.
5. Search terminates when any variable has no legal values.
6. Generic method,
   i) Assigns variable X.
   ii) Forward checking remarks unassigned variable Y connected to X.
   iii) Remove values from D, where value is inconsistent with X.

3.5.12.2 Constraint Propagation

1. A combined approach of heuristic plus forward checking gives more reliable, accurate and efficient results than a singular approch.
2. The forward checking propagates information from assigned to unassigned variables but can not avoid or detect all failure.
3. Constraint propagation repeatedly enforces constraints locally.
4. 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

1. It is fast method of constraint propagation.
2. $X \rightarrow Y$ 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 = Red, V_2 = 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 iff

   V_1 = Red and V_2 = Blue
3. As directed arcs between variables represent the domains of specified variables, they are consistent with each other.
4. Constraint propagation can be applied as preprocessing or propagation step.
   i) Before search-Preprocessing.
   ii) After search-Propagation.

5. The procedure for maintaining arc consistency, can be applied repeatedly.

Constraint Propagation using K-consistency

1. Arc consistency is not capable of detecing all inconsistencies. Partial assignments [V_1 = Red, V_2 = Red] are inconsistent.
2. K-consistency is very strong form of constraint propagation.
3. 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.

Note -

1. Consistency means that each individual variable by itself is consistent; this is also called as node consistency.
2. Consistency is the same as arc consistency.
3. Consistency means that any pair of adjacent-variables can always be extended to a third neighboring variable; this is also called as path consistency.
4. Strongly, K-consistency graph exhibit some properties mentioned below -

   i) It is K-consistent.

   ii) It is also (K - 1) consistent, (K - 2) consistent all the way down to 1 consistent.
5. This is an idealist solution which requires O (nd) time instead of O (n^2d^3).

An exponential order time is required for establishing n-consistency in the worst case.

3.5.13 Local Search for CSP

1. It is most powerful search for CSP's.
2. Backtrack search require more time in dynamic environment which can be reduced by local search.
3. 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.

3.5.13.1 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

1. 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.
2. It also works for hard-problems.
3. Local search is also used in an online setting.

   For Example - in scheduling problem like weekly outline schedule.

- **Example -** of Min conflit

  Consider 4-queen example shown in Fig 3.5.4 in 3 parts (a), (b), (c).

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

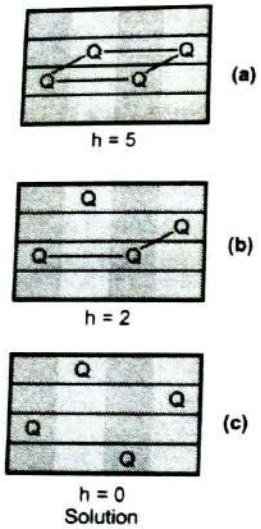


Fig. 3.5.4 Min conflict example

3.5.14 Dealing with Special Constraints

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

1. **All-diff constraints :** It enforces the rule that all the variable in state space must have distinct values (as in the cryptorithmetic problem).
2. **Atmost constraints or resource constraints :**

i) These are most important higher order constraints.

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

3.5.15 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 (discussed in section 3.5.3.1), 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 backjumping 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.

3.5.16 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 subproblems.

Subproblems can be independent or they can be connected. If the subproblems are totally independant then we have very easy way to solve the problem in totality. We can solve each subproblem independantly and then combine the solutions.

Many practical CSP subproblems are connected. The simple case is when the constraint graph forms a tree. Any two variables are connected by atmost one path.

3.5.17 Algorithm for Solving Tree-Structured CSP in Linear Time

1. 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.
2. Label the variables $X_1, \ldots, X_n$ 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,

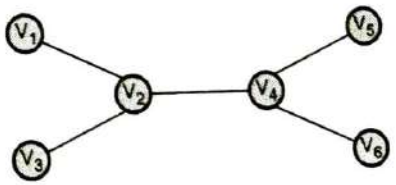


Fig. 3.5.5 Constraint graph

b) A linear ordering of the variables consistent with the tree with V_1 as the root. Refer Fig. 3.5.6.

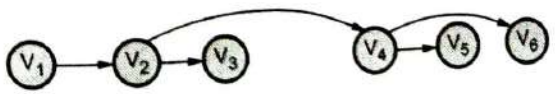


Fig. 3.5.6 Linear ordering of variables

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 far X_i, where X_i, is the parent of X_j.

Note - The complete algorithm runs in time **O (nd^2).**

3.5.18 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.

3.5.18.1 Remove the Nodes Approach (Cutset Conditioning)

1. 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.**
2. 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 inconsistant with the assignment for S.
   ii) If the remaining CSP has a solution, return it together with the assignment for S.

Note :

1. If the cycle cutset has size C, then the total runtime is O (d^c, $(n - c)\ d^2$).
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,

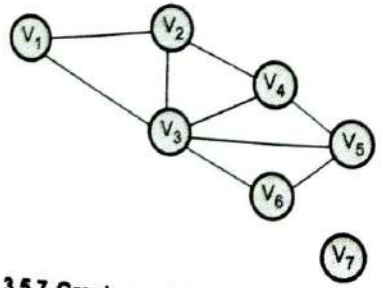


Fig. 3.5.7 Graph used for cutset conditioning

If we remove V_3 then the constraint graph is,

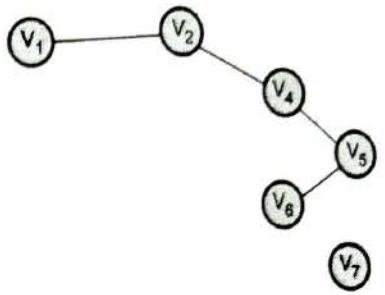


Fig. 3.5.8 Applying cutset on graph

3.5.18.2 Collapse the Node (Tree Decomposition) Approach

1. The approach is based on constructing a tree decomposition of the constraint graph into a set of connected subproblems.
2. Each subproblem is solved independantly and the resulting solutions are then combined.
3. A tree decomposition must satisfy the following three requirements :
   a) Every variable in the original problem appears in at least one of the subproblems.
   b) 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 subproblems.
   c) If a variable appears in two subproblems in the tree, it must appear in every subproblems along the path connecting those subproblems.
4. A given constraint graph admits many tree decompositions. In choosing a decomposition, the aim is to make the subproblems as small as possible.
5. The tree width of a tree decomposition of a graph is one less than the size of the largest subproblem; the tree width of the graph itself is defined to be the minimum tree width among all its tree decompositions.
6. 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 -

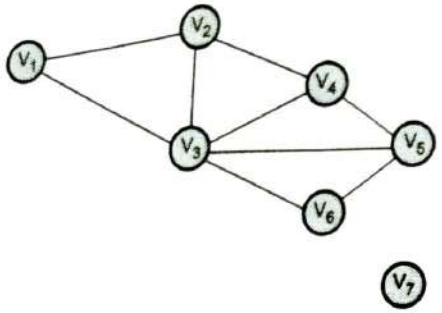


Fig. 3.5.9 Graph used for tree decomposition

A tree decomposition of above graph is,

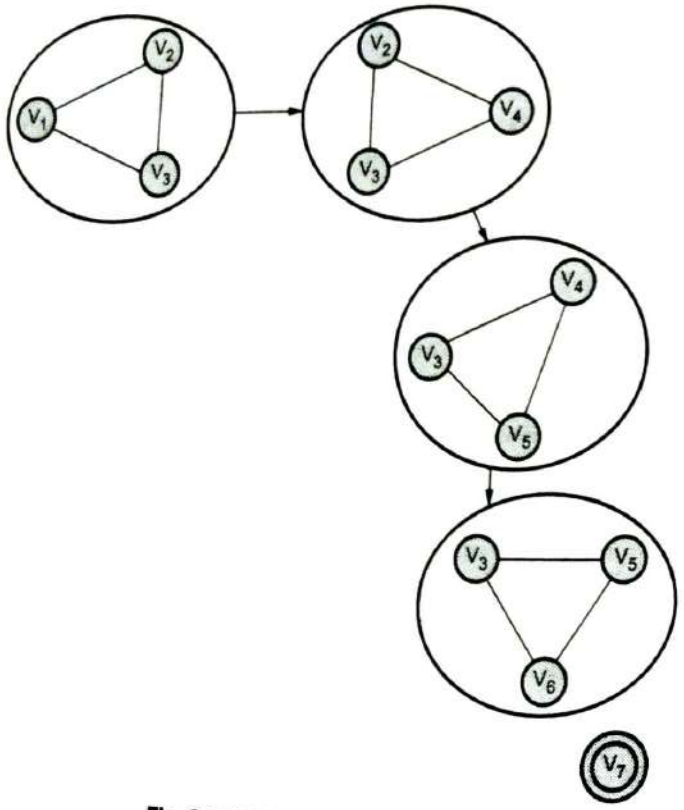


Fig. 3.5.10 Tree decomposition

3.6 Means-Ends Analysis

GTU : Winter-12, 18, Summer-14, 17, 20

- Most of the search strategies either reason forward or backward however, often a mixture of the two directions is appropriate. Such mixed strategy would make it possible to solve the major parts of problem first and solve the smaller problems that arise when combining them together. Such a technique is called "Means - Ends Analysis".
- MEA (Means-Ends Analysis) is a problem solving strategy first introduced in GPS (General Problem Solver) [Newell and Simon, 1963]. The search process over the problem space combines aspects of both forward and backward reasoning in that both the condition and action portions of rules are looked at when considering which rule to apply.
- The means-ends analysis process centers around finding the difference between current state and goal state. The problem space of means - ends analysis has an initial state and one or more goal state, a set of operators with a set of preconditions. Their application and the difference functions computes the difference between two state s(i) and s(j). A problem is solved using means - ends analysis using following steps :
  1. Computing the current state s1 to a goal state s2 and computing their difference D12.
  2. Satisfy the preconditions for some recommended operator OP, which is selected, to reduce the difference D12.
  3. The operator OP is applied if possible. If not, the current state is solved. A goal is created and means-ends analysis is applied recursively to reduce the sub goal.
  4. If the sub goal is solved then the state is restored and work resumed on the original problem.

Means-ends analysis is useful for many human planning activities. Consider the example of planning for an office worker. Suppose we have a different table of three rules :

1. If in our current state we are hungry , and in our goal state we are not hungry , then either the "Visit hotel" or "Visit canteen " operator is recommended.
2. If our current state we do not have money, and if in your goal state we have money, then the "Visit our bank" operator or the "Visit secretary" operator is recommended.
3. If our current state we do not know where something is , need in our goal state we do know, then either the "Visit office enquiry", "Visit secretary" or "Visit co worker " Operator is recommended.

- Differences between the current and goal states are used to propose operators which reduce the differences. The correspondence between operators and differences may be provided as knowledge in the system (in GPS this was known as a Table of Connections) or may be determined through some inspection of the operators if the operator action is penetrable. This later case, which is true of **STRIPS-like** operators, allows task-independent correlation of differences to the operators which reduce them. When knowledge is available concerning the importance of differences, the most important difference is selected first to further improve the average performance of MEA over other brute-force search strategies. However, even without the ordering of differences according to importance, MEA improves over other search heuristics (again in the average case) by focusing the problem solving on the actual differences between the current state and that of the goal.
- In operator sub-goaling, backward chaining is used in which first the operators are selected and then sub goals are set up to establish the preconditions of the operators. If the operator does not produce the goal state, then second sub problem is produced that can reach to goal. If the difference was chosen correctly and if the operator applied is effective at reducing the difference then the two sub problems would be easier to solve than one original problem. The MEA process is then recursively applied.

The MEA algorithm can be summarized as below :

1. Until the goal is reached or no more procedures are available do the steps from 2 to 4.
2. Describe the current state, the goal state and the differences between the two.
3. Use the difference to describe the procedure that would be expected to get nearer to goal.
4. Use the procedure and update current state.
5. If goal is reached then algorithm halts with the success else it halts with failure.

Example 3.6.1 *Consider the following initial and goal configuration for 8-puzzle problem. Draw the search tree. Apply A* algorithm to reach from initial state to goal state and show the solution. Consider Manhattan distance as a heuristic function (i.e. sum of the distance that the tiles are out of place).* **GTU : Summer-17, Marks 7**

Initial state

1	2	3
7	8	4
6		2

Goal state

1	2	3
8		4
7	6	5

Solution : Given :

Initial state

1	2	3
7	8	4
6		5

Goal state

1	2	3
8		4
7	6	5

Hence the goal state (solution)

Applying A* algorithm to reach to goal state,

Heuristic function used : = Manhattan distance
i.e. [Sum of the distance that the tiles are out of place].

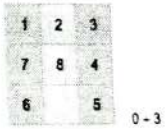
Let : For each node,

$$\hat{f}(n) = \hat{g}(n) + h(n)$$

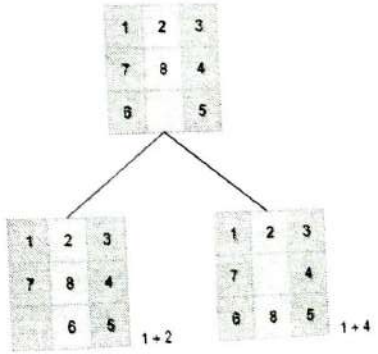
Where,

$\hat{g}(n)$ is an estimate of the 'depth' of 'n' in the graph and h(n) is an heuristic evaluation of node 'n'.

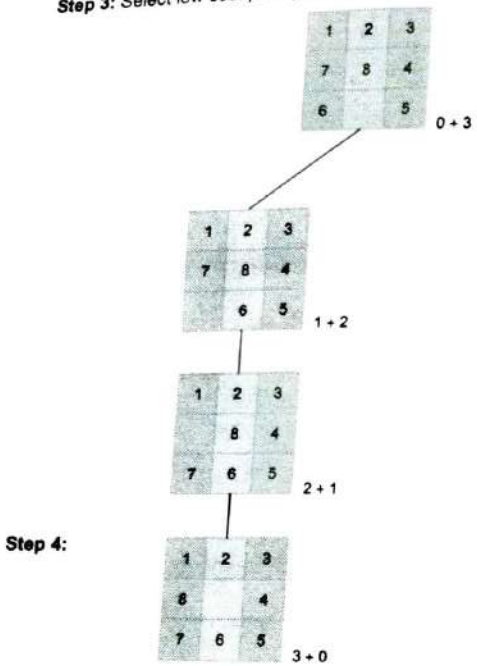
Step 1:



Step 2:



Step 3: Select low cost path (lowest f(n))



Answer in Brief

1. *When is the class of problem said to be intractable ?* **(Refer section 3.1)**
2. *What is the power of heuristic search ? Why does one go for heuristic search ?* **(Refer section 3.1.10)**
3. *What are the advantages of heuristic function ?* **(Refer section 3.1)**
4. *State the reason when hill climbing often get stuck.* **(Refer section 3.2.2)**
5. *When a heuristic function h is said to be admissible ? Give an admissible heuristic function for TSP.* **(Refer section 3.1.10)**
6. *What do you mean by local maxima with respect to search technique ?* **(Refer section 3.2.2)**
7. *What is heuristic search ?* **(Refer section 3.1.10)**
8. *Explain heuristic search with an example.* **(Refer section 3.1.10)**
9. *How to minimize total estimated solution cost using A* search with an example ?* **(Refer section 3.1.6)**
10. *Explain the A* search and give the proof of optimality of A*.* **(Refer section 3.1.6)**
11. *Describe hill climbing, random restart hill climbing and simulated annealing algorithms. Compare their merits and demerits.* **(Refer section 3.2)**

12. *How does hill climbing ensure greedy local search ? What are the problems of hill climbing ?* **(Refer section 3.2)**
13. *How does genetic algorithm come up with optimal solution ?* **(Refer section 3.2.6)**
14. *What is heuristic ? For each of the following type of problem give good heuristics function. i) Block world problem ii) Missionaries and cannibals.* **(Refer section 3.1.10)**
15. *What is heuristics ? A* search uses a combined heuristic to select the best path to follow through the state space toward the goal. Define the two heuristics used (h(n) and g(n)).* **(Refer section 3.1.10)**
16. *Best first search used both OPEN list and a CLOSED list. Describe the purpose of each for the Best-First algorithm. Explain with suitable example.* **(Refer section 3.1)**
17. *Hill climbing is a standard iterotive improvement algorithm similar to greedy best-firsts search. What are the primary problems with hill climbing ?* **(Refer section 3.2)**
18. *What is heuristics ? Explain any heuristics search method. Justify how heuristics function helps in achieving goal state.* **(Refer section 3.1.10)**
19. *What is CSP ?* **(Refer section 3.5)**
20. *Apply constrain satisfaction algorithm to a cryptarithmetic problem given below*

$$\begin{array}{r} SEND \\ +\ MORE \\ \hline MONEY \end{array}$$

(Refer section 3.5)

21. *How CSPs are defined ? Represent map colouring as CSP. Use red, green, blue to colour the map.* **(Refer section 3.5)**
22. *Give the brief summary of backtracking search for CSP.* **(Refer section 3.5)**

3.7 University Questions with Answers

Summer - 12

Q.1 *What is Hill Climbing ? Explain Simple Hill Climbing and Steepest Ascent Hill Climbing.* **(Refer section 3.2.2)** **[7]**

Q.2 *Solve the following cryptarithmetic problem.*

$$\begin{array}{r} S\ E\ N\ D \\ +\ \\ \underline{M\ O\ R\ E} \\ M\ O\ N\ E\ Y \end{array}$$

(Refer section 3.5) **[7]**

Winter - 12

Q.3 *Explain A* algorithm.* **(Refer section 3.1.6)** **[7]**

Q.4 *Explain steepest ascent Hill climbing algorithm.* **(Refer section 3.2.2)** **[7]**

4 Knowledge Representation and Related Issues

Syllabus

Representations And Mappings, Approaches To Knowledge Representation, Representation.

Contents

4.1 Representation and Mappings

GTU : Winter-14,18,19, Summer-16,18,20

4.1.1 Introduction

- Search-based problem solving programs require some knowledge to be implemented. Knowledge can be a particular states or path toward solution, rules, etc. Before being used this knowledge must be represented in a particular way with a certain format. Knowledge Representation (KR) is an important issue in computer science in general and in AI in particular. "The dominant paradigm for building intelligent systems since the early 1970s has been based on the premise that intelligence presupposes knowledge". Generally knowledge is represented in the system's knowledge base, which consists of data structures and programs. In addition, the intelligent system is expected to have a program called an inference engine that implements the reasoning patterns necessary for the task at hand. Thus current AI theory and practice dictate that intelligent systems be knowledge based consistent with this simple knowledge base plus inference engine architecture. This emphasis on knowledge has led to suggestions that AI can be arguably called applied epistemology".

4.1.2 Issues in Knowledge Representation

- Are any attributes of objects so basic that they occur in almost every problem domain ? If there are such attributes then we need to make sure that they are handled appropriately in each of the mechanism we propose. If such attributes exists, what are they ? There are several issues that must be considered when representing various kinds of real-world knowledge.

Important Attributes

- Are there any attributes that occur in many different types of problem ?

- There are two *instance* and *isa* and each is important because each supports property inheritance.
- There are two important attributes that are of general significance such as ISA and instances. These attributes are important because they support property inheritance. Relationship among attributes must be considered carefully which is depicting more knowledge.

❑ Relationships

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

band(John Zorn,Naked City)

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

Another representation is band = Naked City

- band-members = John Zorn, Bill Frissell, Fred Frith, Joey Barron,

❑ Granularity

- At what level should the knowledge be represented and what are the primitives. Choosing the Granularity of Representation Primitives are fundamental concepts such as holding, seeing, playing and as English is a very rich language with over half a million words it is clear we will find difficulty in deciding upon which words to choose as our primitives in a series of situations.

If Tom feeds a dog then it could become :

feeds(tom, dog)

If Tom gives the dog a bone like :

gives(tom, dog,bone) Are these the same ?

In any sense does giving an object food constitute feeding ?

If give(x, food) → feed(x) then we are making progress.

But we need to add certain inferential rules.

In the famous program on relationships Louise is Bill's cousin How do we represent this ? louise = daughter (brother or sister (father or mother(bill))) Suppose it is Chris then we do not know if it is Chris as a male or female and then son applies as well.

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

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

- The finest level of knowledge that is granularity is another issue to handle. Granularity is level of knowledge that needs to be represented. For this, one should have complete primitive (basic) understanding of the knowledge to be represented in the system.
- Other significant issues those need to be handled are Inverses, Existence in an hierarchy, Technique for reasoning about values, Single-valued attributes. Major attributes are required to be identified. The set of objects whose knowledge is required to be stored should be clearly identified.

4.1.3 The Techniques of Representation and Mappings

AI can be used to solve a complex problems encountered within. Nevertheless large amount of knowledge as well as some means of manipulating that knowledge is required so as to create solutions for new problems. In the representation there are two different entities that must be considered :

- Facts : truths in some relevant world. These are things that we want to represent.
- Representation of facts in some chosen formalism. These are things that can actually be manipulated.

Structuring of these entities can be done in two levels :

- The *knowledge level* at which facts are described.
- The *symbol level* at which representation of some objects at the knowledge-level are defined in terms of symbols that can be manipulated by programs.

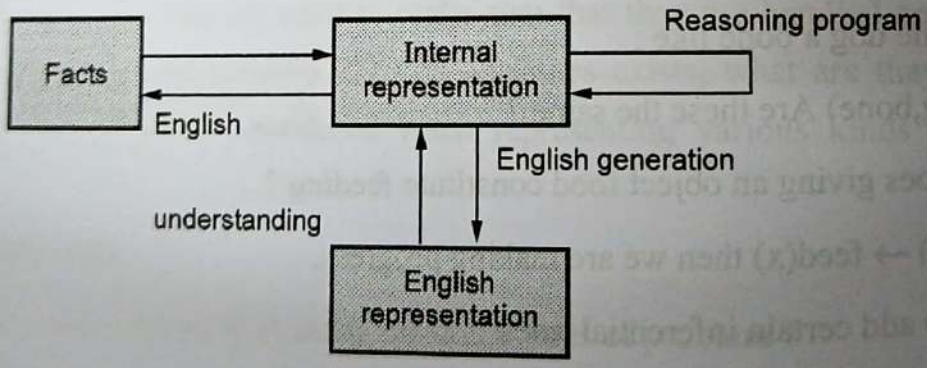


Fig. 4.1.1 Mappings between facts and representation

Our main goal is to focus on facts, representation as well as the two-way mappings that must exist between the two as shown in the Fig. 4.1.1 above. The links in the figure are called *representation mappings*. In representation mappings, there are :

- Forward *representation* which maps from facts to representation.
- Backward *representation* which maps the other way.

One representation of facts concerns with natural language (particularly English) sentences is that, regardless of the representation for facts that we use in a program, we may also need to be concerned with an English representation of those facts that in order to facilitate getting information into and out of the system. We must also have mapping functions from English sentences to the representation which we are actually going to use and from it back to sentences as shown in the Fig. 4.1.1. For example we can use mathematical logic as the representation formalism. Consider the English sentences below.

Tommy is a dog. This fact can also be represented in logic as follows : *Dog(Tommy)*

Suppose also we have a logical representation of the fact : *all dogs have tails* as explained below. Using the deductive mechanisms of the logic, we may generate the new representation object. Using an appropriate backward mapping function we could then generate the English sentence : *Tommy has a tail* Or we can make use of this representation of new fact to cause us to take some appropriate action or to derive representation of additional facts.

Consider example of the Multilated Checkerboard Problem. "Consider a normal checker board from which two squares, in opposite corners, have been removed. The task is to cover all the remaining squares exactly with donimoes, each of which covers two squares. No overlapping, either of dominoes on top of each other or of dominoes over the boundary of the multilated board are allowed. Can this task be done ?".

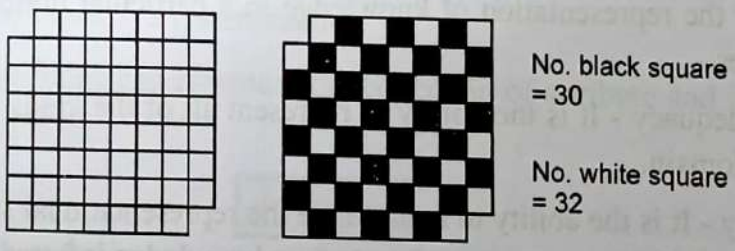


Fig. 4.1.2 A multilated checker board

A example follows :

- Checkerboard total contains 32 white squares and 30 black squares.
- When every domino cover two neighboring squares, a black one and a white one, then first thirty dominos cover 30 black squares and 30 white squares, and leaving two white square and zero black domino.
- These two black squares can not be adjusted and can not cover remaining domino.
- It is impossible to cover all 62 squares with 31 one dominos.

An observation which can be made in the computation is that the number of black squares correspond to the number of dominoes in the partial covering. The same is true for the number of white fields, which enforces the number of black squares to coincide with the white squares, when investigated the inter play between covered squares on the board and dominoes contained in the partial covering.

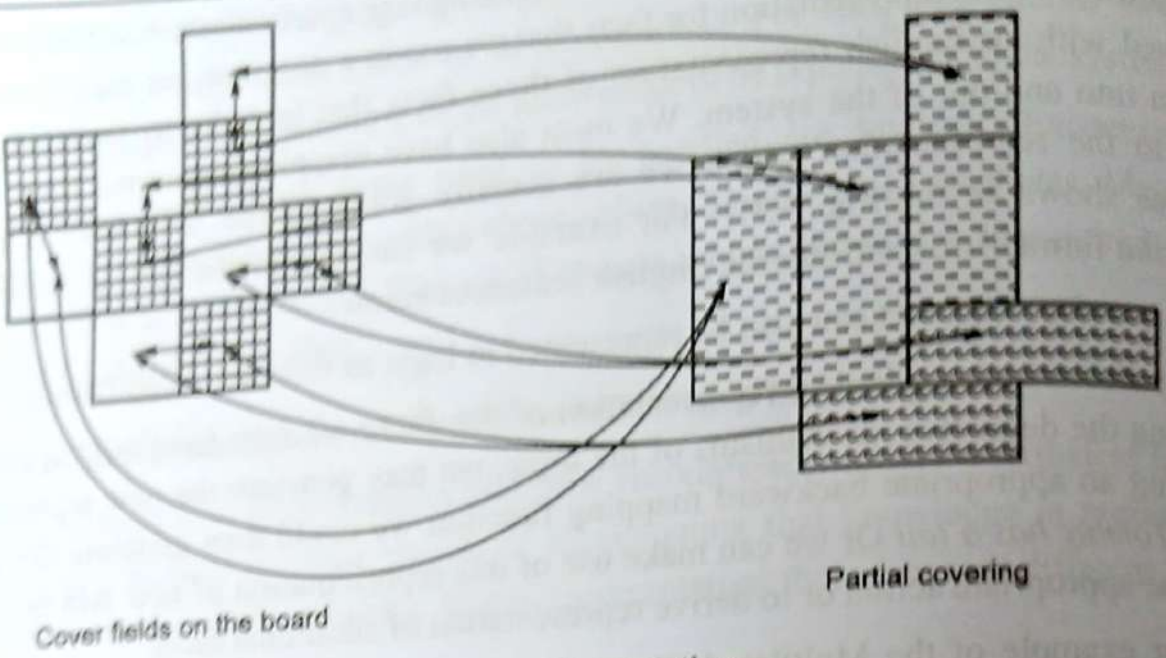


Fig. 4.1.3 Observation

4.2 Approaches to Knowledge Representation

GTU : Summer - 15, 17, 18, 20, Winter - 18

A good system for the representation of knowledge in a particular domain should possess the following properties -

- Representational adequacy - It is the ability to represent all of the kinds of knowledge that are needed in that domain.
- Inferential adequacy - It is the ability to manipulate the representational structures in such a way as to derive new structures corresponding to new knowledge inferred from old.
- Inferrential efficiency - It is the ability to incorporate into the knowledge structure additional information, that can be used to focus the attention of the inference mechanisms in the most promising direction.
- Acquisitional efficiency - Acquiring new information easily.

❑ Two types of approaches to knowledge representation :

1) Simple relational knowledge
2) Inheritable knowledge

1) Simple relational knowledge

- This is the simplest way of storing fact which uses relational method, when every and each fact about a set of objects is set out sequentially and automatically in column.
- This type of representation is small procedure for inference.
- It is used to define inference engines.

For example

Player	Weight	Age	Play cricket
Monu	70	30	Right H.
Sonu	65	25	Right H.
Bablee	50	29	Left H.
Soni	45	29	Right H.
Moni	42	25	Left H.

Player_info ('Monu', 70, 30, right H)

2) Inheritable knowledge

- Relational knowledge is made up of object associativity like co-relation associated values attribute.
- All data should be organised into a hierarchy of classes.
- Inherit values from being all members of class.
- Class must be arranged in a generalization.
- Every individual frame can represent the collection of attribute and its value associated with a individual node.

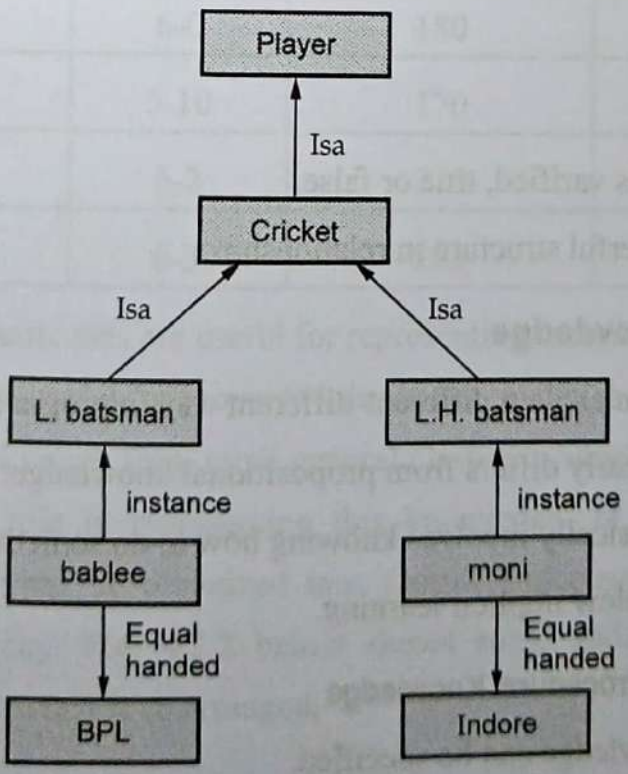


Fig. 4.2.1 Inheritable knowledge

❑ **Example**

Properties of inheritance hierarchy

1)  to be point from object and its value.

2) Boxed : to be object and value of attribute any object.

3) It may be also be called slot-and filter structure.

Algorithm retrieve

To retrieve a value for attribute of an instance object.

1) Find object in the knowledge base.

2) If there is a value for the attribute, report that value.

3) Otherwise look value of instance, if not then fail, otherwise go to the node and value for the attribute, if one is found, report it. Otherwise, do until there is n search using ISA, found for the attribute.

4.2.1 Inferential Knowledge

When inheritance property is very useful form of inference, represent the knowled formal logic.

All cat have tails $\forall x : \text{dog}(x) \rightarrow \text{has a tail}(n)$

❑ **Set of rules**

1) Define require fact.

2) Additional statement is varified, true or false.

3) Logic provides a powerful structure in relationships.

4.2.2 Procedural Knowledge

- Procedural knowledge can explain different-different way in program.
- Procedural knowledge clearly differs from propositional knowledge.
- Procedural knowledge basically involves knowing how to do something.
- Procedural knowledge follow implicit learning.

4.2.2.1 Advantages of Procedural Knowledge

1) Property specific knowledge can be specified.

2) Extended logical inference is possible.

4.2.2.2 Disadvantages of Procedural Knowledge

1) Consistency : all deduction are not always correct.

2) Completeness : all cases are not easy to represent.

There are multiple techniques for knowledge representation. Different representation formalisms are,

- Rules
- Logic
- Natural language
- Database systems
- Semantic nets
- Frames

❑ Many programs rely on more than one technique

- Database system - They are used in representing Simple Relation Knowledge which is in declarative facts and can be said as a set of relations of the same sort within database systems. Fig. 4.2.2 shows an example of such systems.

Player	Height	Weight	Bats-Thrown
Ram	6-0	180	Right-Right
Shyam	5-10	170	Right-Right
Veer	6-2	215	Left-Left
Tarun	6-3	205	Left-Right

- Semantic nets - Semantic nets are useful for representing inheritable knowledge. Inheritable knowledge is the most useful for *property inheritance*, in which elements of specific classes inherits attributes and values from more general classes in which they are included. Frames also do play a big role in representing this knowledge. In order to support property inheritance, objects must be organized into classes and classes must be arranged in a generalization hierarchy. Fig. 4.2.2 below shows some additional baseball knowledge inserted into a structure that is so arranged.

❑ Example :

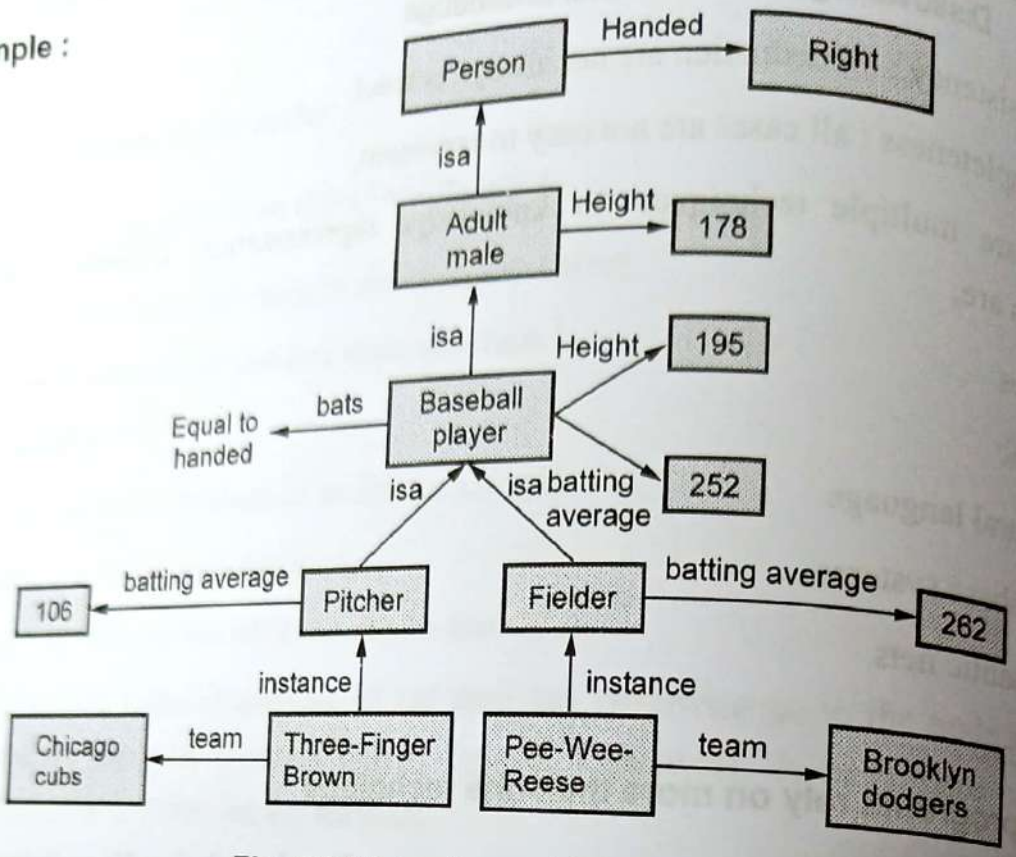


Fig. 4.2.2 Inheritance hierarchy

Isa hierarchy is normally used in semantic nets/frames. Lines represent attributes. Bo nodes represent objects and values of attributes of objects. Correct deduction from Fig. 4. could be : height of Three-Finger Brown is 195 cm. An incorrect deduction could be : he of Three-Finger Brown is 178 cm. The structure shown in the Fig. 4.2.2 is a *slot-and-fi* structure. It may be also called a *semantic network* or a collection of frames.

❑ **Predicate logic**

- Predicate logic is used to represent inferential knowledge.
- Logic provides powerful structure in which to describe relationships among values.
- It can be combined with some other powerful description language with an ISA hierarchy

❑ **Production rules**

- Production rules are useful in representing procedural knowledge.
- Procedural knowledge is form of operational knowledge which specifies what to do when
- Previously it was done using programming language such as LISP.
- However it was hard to reasoning with this method hence in AI programs procedur knowledge is represented using production rules.

❑ Inheritable knowledge

Relational knowledge is made up of objects consisting of

- Attributes
- Corresponding associated values.

We extend the base more by allowing inference mechanisms :

- Property inheritance
  - Elements inherit values from being members of a class.
  - Data must be organised into a hierarchy of classes (Fig. 4.2.3).
- Boxed nodes -- objects and values of attributes of objects.
- Values can be objects with attributes and so on.
- Arrows -- point from object to its value.
- This structure is known as a slot and filler structure, semantic network or a collection of frames.

The algorithm to retrieve a value for an attribute of an instance object :

1. Find the object in the knowledge base.
2. If there is a value for the attribute report it.

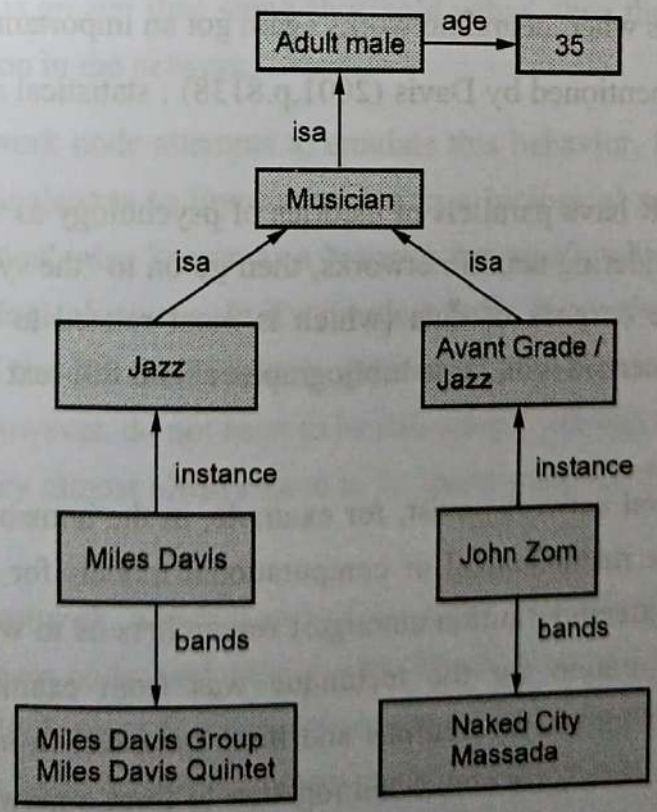


Fig. 4.2.3 Property inheritance hierarchy

3. Otherwise look for a value of instance if none fail.
4. Otherwise go to that node and find a value for the attribute and then report it.
5. Otherwise search through using isa until a value is found for the attribute.

- Knowledge Representation (KR) is an important issue in computer science in general and in AI in particular. "The dominant paradigm for building intelligent systems since the early 1970s has been based on the premise that intelligence presupposes knowledge. Generally, knowledge is represented in the system's knowledge base, which consists of data structures and programs. In addition, the intelligent system is expected to have a program called an inference engine that implements the reasoning patterns necessary for the task at hand. Thus current AI theory and practice dictate that intelligent systems be knowledge based, consistent with this simple knowledge base plus inference engine architecture. This emphasis on knowledge has led to suggestions that AI can be arguably called applied epistemology".
- The approach described above may be termed the symbol-manipulation approach. Historically, however, AI grew out of work in which another approach, neural networks (or connectionism or parallel distributed processing or non-symbolic representations) played the major role, but this approach was outplayed by the symbol-manipulation approach until the 80s when neural networks again got an important role.
- A final approach is mentioned by Davis (2001,p.8138) : statistical analysis of large corpora of data.
- The approaches to KR have parallels in theories of psychology as well as in epistemology. We will start by considering neural networks, then go on to "the symbolic" approaches and finally consider large corpora of data (which is most related to library and information science, which is concerned with large bibliographical and full-text databases).

❑ Neural networks

- While biological neural networks exist, for example, in the human brain, Artificial Neural Networks (ANN), are mathematical or computational models for information processing. There is no precise agreed definition amongest researchers as to what a neural network is, but the original inspiration for the technique was from examination of bioelectrical networks in the brain formed by neurons and their synapses. In a neural network model, simple nodes (or "neurons") are connected together to form a network of nodes, hence the term "neural network".

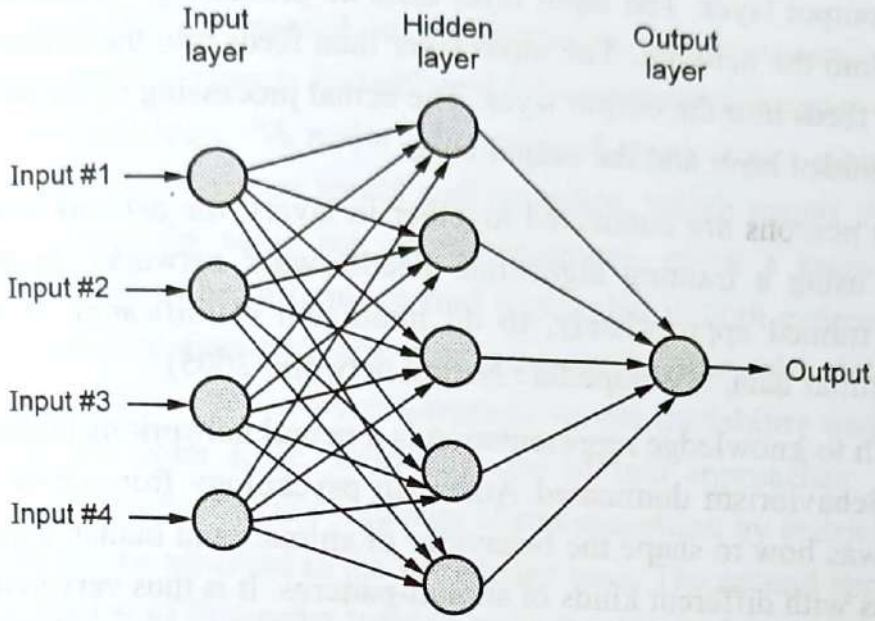


Fig. 4.2.4 A model of a neural net

- In a typical neural network, each node operates on a principle similar to a biological neuron. In a biological neuron, each incoming synapse of a neuron has a weight associated with it. When the weight of each synapse, times it's input, is summed up for all incoming synapses and that sum is greater than some 'threshold value', then the neuron fires, sending a value to another neuron in the network.
- The typical neural network node attempts to emulate this behavior. Each node has a set of input lines which are analagous to input synapses in a biological neuron. Each node also has an 'activation function' (also known as a 'transfer function'), which tells the node when to fire, similar to a biological neuron. In it's simplest form, this activation function can just be to generate a '1' if the summed input is greater than some value or a '0' otherwise. Activation functions, however, do not have to be this simple - in fact to create networks that can do useful work, they almost always have to be more complex, for at least some of the nodes in the network.
- A feedforward neural network, which is one of the more common neural network types, is composed of a set of these nodes and connections. These nodes are arranged in layers. The connections are typically formed by connecting each of the nodes in a given layer to all of the neurons in the next layer. In this way every node in a given layer is connected to every other node in the next layer.

- Typically there are at least three layers in a feedforward network : an input layer, a hidden layer and an output layer. The input layer does no processing : it is simply where the input vector is fed into the network. The input layer then feeds into the hidden layer. The hidden layer, in turn, feeds into the output layer. The actual processing in the network occurs in the nodes of the hidden layer and the output layer.
- When enough neurons are connected together in layers, the network can be 'trained' to do useful things using a training algorithm. Feedforward networks, in particular, are very useful, when trained appropriately, to do intelligent classification or identification tasks on unfamiliar data." (Wikipedia : Neural network, 2005).
- As an approach to knowledge representation is a neural network much like behaviourism psychology. Behaviorism dominated American psychology from about 1913 to 1970. Its main interest was how to shape the behaviour of animals and human beings by confronting such organisms with different kinds of stimuli-patterns. It is thus very much an input-output approach (or stimuli-response approach). They tried to avoid mental terms (e.g. memory) and to replace them with terms referring to relations between stimuli and response (e.g. replace "memory" with "delayed response"). Although most behaviourists neglected brain structures and processes and preferred to look at the brain as a "black box", some behaviourists were interested in brain models, and the idea of neural networks was put forward for the first time by the behaviourist Donald O. Hebb in 1949.
- Both the computer technology of neural nets and behaviourism are closely connected to epistemological ideas developed in particular by classical British empiricism. The basic idea may be that knowledge is represented in the brain as a result of stimuli-processes, where learning based on repetitions of similar stimuli follows the laws of association. From a point of view the most important issue is that knowledge representation is provided by somebody who is in control of the learning process. It is his or her view of what represents wanted behaviour that indirectly manage what is considered true, relevant and important knowledge. This knowledge is not formulated and provided directly, but is implemented in the system or the organism by manipulating the stimulation of the system or organism (simplified : by feedback which involves rewards and/or punishment).

❑ Symbol representation

- There are several approaches to knowledge representation in AI, which can be seen as subcategories of the symbol-representation approach. They all share the conditions that knowledge is explicated by the use of some kind of symbolic language and is installed in the system "manually", piece by piece. The four most important kinds of symbolic systems may be a) logic based systems b) semantic networks and c) frame-based systems.

a) Logic based representations

- Knowledge may be represented in computers by programmers writing declarative sentences such as "Socrates is human" and "if somebody is human, then she is mortal" using mathematical logic. "A major advantage of many logics adopted for knowledge representation is that they are sound and complete, which means that derivability and provability lead to the same set of consequences, given a knowledge base. It has however turned out to be difficult to find logics that is both expressively adequate for knowledge representation and also computationally tractable. "Attempts to find an acceptable compromise to the expressiveness versus tractability trade-off generally use variations of first-order logic, following one of two approaches. The first approach limits the expressiveness of the language of representation by restricting the form of the formulas that can be admitted in the knowledge base. The second approach redefines the provability relation of first-order logic to make it computational tractable.
- Relational databases . . . widely used to represent "simple" facts, such as people's addresses or salaries, constitute a good example of the first approach (. . .)." (Kramer and Mylopoulos, 1992, p. 745-746). The second approach may for example make a slight change in the semantics of existential quantification which makes large representations computational tractable, but this has a remarkable impact on the provability relation.
- Logic based systems may also use **procedural representations**. "Declarative representations treat the intended meaning of a knowledge base as a foundation that imposes constraints on knowledge base operations. Procedural representations, on the other hand, reverses this dependency by identifying the meaning of a knowledge base with its use"(Kramer and Mylopoulos, 1992, p. 746).

b) Semantic networks

- Semantic networks are knowledge representation schemes involving nodes and links between nodes. The nodes represent objects or concepts and the links represent relations between nodes. The links are directed and labeled. Semantic nets were originally motivated by cognitive models of human memory.
- According to Kramer and Mylopoulos (1992, p. 747-748) their popularity and success can best be understood as a convenient compromise between the declarative and the procedural extremes, while "others have argued that semantic networks offer a fundamentally different representational paradigm that is object centered in the sense that it is based on object descriptions rather than arbitrary propositions and focuses on knowledge organization.

- WordNet is an example of a semantic net. The semantic web is a concept that re a major research program with semantic networks. For many persons this i semantic web represents the kind of knowledge organization with the most p prospects.

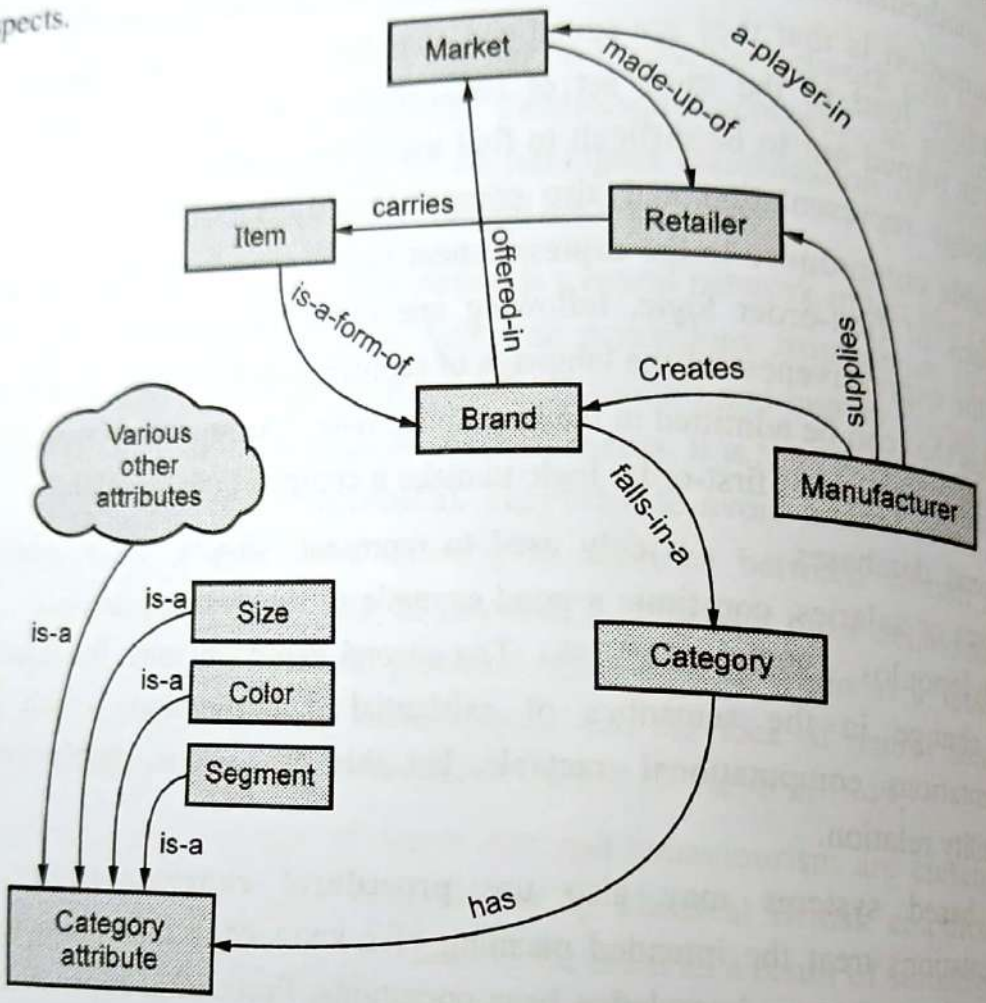


Fig. 4.2.5 Semantic nets

c) Frame-based representations

- Frame-based systems are knowledge representation systems that use frames, originally introduced by Minsky (1975), as their primary means to represent knowledge. A frame is a structure for representing a concept or situation "restaurant" or "being in a restaurant". Attached to a frame are several kinds information, for instance, definitional and descriptive information and how to frame. Frames are supposed to capture the essence of concepts or stereo situations, for example going out for dinner, by clustering all relevant informa these situations together. This means, in particular, that a great deal of procedu expressed knowledge should be part of the frames. Collections of such frames are organized in frame systems in which the frames are interconnected.

- Obviously, frame-based systems are in many ways similar to object-oriented programming languages; indeed, the two theories interacted strongly in their development.
- The chief advantages of frame-based architectures are expressivity, flexibility and ease of use. The chief disadvantages is lack of precision and lack of a well defined model of inference. The architecture provides a wealth of features and options for both representation and inference, but only a weak underlying model. Hence, in a complex case, it is difficult to predict how these features will interact or to explain unexpected interactions, which makes debugging and updating difficult.
- From a psychological point of view has a tendency to overuse frames as explanations been critized : "I am not going to argue against the `existence' (whatever that may be) of organised knowledge structures. What I will do is place doubts on the explanatory value of concepts as frames, conventions, scripts and so on. . . Even if there are structures like frames and scripts, they are relatively easy for people to override. People can still use arbitrary knowledge of the world to understand sentences and scenes : you cannot exclude any part of the knowledge base in advance, using some general prestructuring of that knowledge. Therefore, the content of such structures as frames and scripts must themselves be both analyzable and subject to reasoning by their users, which puts us back almost where we started. What we have gained is a summary of the aggregate regularities frequently or typically, exhibited. The structures themselves tell us nothing about people's cognitive capacities, only about what are probably quite ephemeral habits of thought which people can change. In terms of Billig (1987) frames and scrips lack any kind of `witcraft'. Frames, scripts and related concepts summarize some of the patterns that emerge when people don't bother to think. " (Vliet, 1992)

❑ General epistemic aspects of KR in computer science

- "The KR [Knowledge Representation] architectures we have considered above [logic-based systems, semantic networks and frame-based systems], together with many other proposals of a more or less similar flavour, such as production systems, constitute what may be called the `classical,' or (with some question begging) the `knowledge-based' approach to AI. Knowledge representation, in this view, involves large, complex structures of symbols, defined and assembled by hand. This approach to AI essentially derives from a line of philosophical thought running from Descartes through Leibniz, Frege, and Russell. In the late 1980s and 1990s, however, as a result of the inherent difficulty of this line of research, and of the limited progress that has been made, this approach to AI has been challenged by two alternative methodologies : neural networks, and statistical analysis of large corpora". (Davis, 2001, p. 8137-8138).

- The symbolic forms of knowledge representation thus correspond to cognitivism in psychology and to rationalism in epistemology while neural networks correspond to behaviourism in psychology and to empiricism in epistemology. They may both be said to ignore the subjective side of knowledge representation. In the symbolic form of KR the person in control of the programming tasks are defining and assembling the knowledge. Nothing is said about whether different subjects (representing different traditions or paradigms) would or should define and assemble different kinds of knowledge. In neural networks the person in control of the stimulation is determining what the organism or the system should learn. Nothing is said about how persons' criteria may be connected to subjective views and to socio-cultural factors. In both cases it is assumed without any kind of examination, that the knowledge representation is "objective". What kinds of perspectives that are missing may be uncovered by the following quotation : "More recently, Clancey (1991) argues that the knowledge level has relativistic properties. A knowledge-level description is of an agent in its environment. It is an observer's theory, not representations possessed by the agent being studied" (Clancey, 1992, 743). Yes! The description of an agent's or a system's knowledge level (or generalized : a description of its knowledge) has relative properties and implies the theory of the agent or system from specific perspectives. This is a basic point in the pragmatic understanding of knowledge but it has yet to be fully implemented in theories of knowledge representation.
- In spite of the recognition of AI as applied epistemology there has not been much systematically investigation between on the one hand epistemological theories and on the other hand theories of knowledge representation. This is odd, because epistemology is the theory of knowledge and any theory of knowledge representation must be based on a theory of knowledge.
- In overviews of knowledge representation in AI (such as Davis, 2001, and Kramer and Mylopoulos, 1992) only empiricism and rationalism has been reflected as approaches. There is an obvious need to expand such overviews by a broader coverage of different epistemological positions.
- Hermeneutics has been regarded in the field of AI, first and foremost by Winograd & Flores (1987). Additional contributions include Mallery, Hurwitz and Duffy (1992); Chalmers (1999) and Fonseca and Martin (2005). There seems still to bee a need for a more direct application of historicist/hermeneutical/pragmatic approaches to knowledge representation.
- In the next section we shall very briefly suggest such an approach to knowledge representation. The main point will be put on the subjectivity of the person doing the representation in line with the thoughts introduced in the first part of this article.

❑ Analysis of large corpora

- Among alternatives to both neural nets and symbolic KRs E. Davis briefly mentions text corpora :
- "The statistical approach to AI involves taking very large corpora of data and analyzing them in great depth using statistical techniques. These statistics can then be used to guide new tasks. The resulting data, as compared to the knowledge-based approach, are extremely shallow in terms of their semantic content, since the categories extracted must be easily derived from the data, but they can be immensely detailed and precise in terms of statistical relations. Moreover, techniques - such as maximum entropy analysis - exist that allow a collection of statistical indicators, each individually quite weak, to be combined effectively into strong collective evidence. From the point of view of knowledge representation, the most interesting data corpora are online libraries of text. Libraries of pure text exist online containing billions of words; libraries of extensively annotated texts exist containing hundreds of thousands to millions of words, depending on the type of annotation. Now, in 2001, statistical methods of natural language analysis are, in general, comparable in quality to carefully hand-crafted natural language analyzers; however, they can be created for a new language or a new domain at a small fraction of the cost in human labor" (Davis, 2001, p. 8138).
- Large corpora of data may be approached by methods related to empiricism, which seems to be what Davis is suggesting. There is an important difference, however, between traditional empiricist approaches to knowledge representation and "text corpora" approaches. In the traditional approach is represented what is considered knowledge by the person doing the representation. There is only one voice present. In large corpora of texts many voices are present (what kind of voices varies according to how the text corpus is selected, e.g. if it consists of newspapers or scholarly papers).
- Large corpora of texts consist of documents each of which is itself a system of arguments and knowledge claims. We are now in the realm of Library and Information Science (LIS) rather than in computer science in a narrow sense. What are represented in LIS are representations of documents representing knowledge (thus meta-representations). If, for example, the text corpus is an academic corpus from the same domain as the person doing the representation (e.g. computer science) then different suggestions and voices on how best to perform the task at hand is present in the very material to be (meta) represented. Different paradigms in KR contain arguments in favour of specific ways to do the representation.

- In other words : The texts to be organized are voices, which probably will contain different implications for how this knowledge should be organized (and by the way also implications for how texts should be selected in the first hand). This argument may be expanded also to cases in which the corpus is not in the domain of knowledge representation : Any document has implicit or explicit criteria of relevance, which are of importance for organizing those documents.
- If we consider the domain of Arts then the criteria for how best to represent arts is depending on what is considered (good) art. As discussed by Ørom (2003) different traditions in Arts have different implications for how arts should be represented. In corpora there are different voices, not just the programmers voice. The programmer may ignore these voices and provide knowledge representations based on his own voice alone, or the programmer may consider those voices and provide a knowledge representation, which represents a dialog between himself and the voices in the corpora. This way it is possible to use text corpora based on pragmatic epistemologies rather than empiricist epistemologies (see also Hjørland and Nissen Pedersen, in press).

Answer in Brief

1. *Write a note on representations and mappings.* **(Refer section 4.1)**
2. *What are various approaches to knowledge representation ?* **(Refer section 4.2)**
3. *Explain neural nets.* **(Refer section 4.1)**

4.3 University Questions with Answers

Winter - 14

Q.1 Explain the different issues in knowledge representation. **(Refer section 4.1.2)**

Summer - 15

Q.2 Explain different approaches of knowledge representation. **(Refer section 4.2)**

Summer - 16

Q.3 Explain the different issues in knowledge representation. **(Refer section 4.1)**

Summer - 17

Q.4 Explain property inheritance algorithm with example. **(Refer section 4.2)**

➥ Summer - 18

Q.5 Differentiate with example representation of " Instance" and "Isa" relationships. **(Refer section 4.2)** [7]

Q.6 Explain with example how choosing the granularity of representation and finding the right structure are crucila issues in knowledge representation ? **(Refer section 4.1.1)** [7]

➥ Winter - 18

Q.7 Discuss the different approaches to knowledge representation. **(Refer section 4.2)** [4]

➥ Winter - 19

Q.8 Explain why it is necessary to choose appropriate granularity for knowledge representation. **(Refer section 4.1)** [3]

Q.9 Briefly discuss declarative and procedural knowledge. **(Refer section 4.2)** [3]

➥ Summer - 20

Q.10 Define the following words in the context of AI :

i) Intelligence ii) Knowledge iii) Information iv) Logical reasoning. [4]

(Refer section 4.1)

Q.11 Briefly discuss declarative and procedural knowledge. **(Refer section 4.2)** [3]

❑❑❑

5 Logical Agents and Knowledge Representation using Propositional Logic

Syllabus

Logical Agents, Knowledge-based agents, The Wumpus world, Logic, Propositional logic, Propositional theorem proving, Effective propositional model checking, Agents based on propositional logic.

Contents

Propositional Logic

5.1 Characteristic of Propositional Logic

GTU : Summer-20

1. Propositional logic is declarative (pieces of syntax corresponds to facts).
2. Propositional logic allows partial/disjunctive/negated inference (unlike most data structures and databases)
3. Propositional logic is compositional.

   (Meaning of $B_{1,1} \wedge P_{1,2}$ is deried from meaning of $B_{1,1}$ and meaning of $P_{1,2}$).
4. Meaning in propositional logic is context independent (unlike natural language, where meaning depends on context).

5.2 Drawbacks of Propositional Logic

1. Propositional logic has very limited expressive power (unlike natural language).

   Example : Cannot express "pits causes breezes in adjacent squares".

   For such statement one need to write, one sentence for each square.
2. Propositional logic represents statements about the world without reflecting the structure and without modeling these entities explicitly.
3. Therefore some knowledge is hard or impossible to encode in the propositional logic.
4. Two cases that are hard to represent :

   Case i) Statements about similar objects and their relations :

   a) Statements about similar objects and relations needs to be enumerated.

   b) Because of this knowledge-base grows large.

   c) We need to represent many rules to allow inferences. The solution to this problem is introduce variables.

   Case ii) Statements referring to groups of objects :

   a) Statements referring to groups of objects require exhaustive enumeration of objects.

   b) Solution to this problem is to allow quantification in statements. [which is done in first order logic].

5.3 Syntax for Propositional Logic

Syntax : It defines allowable sentences in the model.

Propositional calculus symbols :

1. Symbols

i) They are propositional calculus symbols

P, Q, R, S .....

ii) Predicate symbols : They are used to represent a relation in a domain.

For example :

If we want to represent a sentence like, "Dipu reads book".

We will use the predicate symbol "READS". The simple predicate will be,

READS (Dipu, Book).

iii) Constant symbols : A constant symbol is used to represent objects or entities in a domain. These objects or entities may be physical objects, people, concepts or anything that is to be named.

For example : In the above formula Dipu and Book are constant symbols.

iv) Variable symbols : Variables like x or y are also symbols. They allow the users to be indefinite about which entity is to be reffered to example READ (x, y).

v) Function symbols : They denote functions in the domain of discourse.

For example : "Ram's mother is married to Ram's father", the atomic formula would be,

MARRIED [MOTHER (Ram), FATHER (Ram)] where MOTHER, FATHER and MARRIED are function symbols.

2. Truth symbols

True, False.

3. Connectives

$\wedge, \vee, \neg, \rightarrow, \equiv$

Note Propositional symbols denote propositions, or statements about the world that may be either true or false, such as "the car is red" or "water is wet".

5.3.1 Propositional Calculus Sentence

i) Every propositional symbol and truth symbol is sentence.

For example : True, P, Q, R are sentences.

ii) Atomic sentence : It is indivisible (non-composite) syntactic element. It consists of proposition symbol. Each such symbol stands for a proposition that can be true or false.

iii) Complex sentence : It is constructed from simpler sentences using logical connectives.

Note Logical sentences are also called as well-formed formulas or WFF's.

5.3.2 Connectives

Atomic formulas can be converted to well formed formulas by combining them with connectives.

a) The AND connective : ∧

The "∧" connective is used to represent compound statement like "Nimu wears pink dress".

WEARS (Nimu, Dress) ∧

COLOUR (Dress, Pink)

Where the predicate WEARS gives the relation between person and object and the predicate COLOUR gives the relation between object and colour.

Formulae built by connecting other formulae by ∧'s are called conjunctions and each of the component of the formula is called as **conjunct.**

b) The OR connective : ∨

The symbol "∨" is used to represent **inclusive** "or".

For example :

"Dipu plays badminton or tennis" can be represented by

PLAYS (Dipu, Badminton) ∨ PLAYS (Dipu, Tennis)

Formulas built by connecting other formulas by ∨'s are called disjunctions and each of the component formula is called as **disjunct.**

c) The NOT connective : ¬

The symbol "¬" is used to represent the NOT connective. It is used to negate the truth values of a formula. It changes the value of sentence from T to F and vice versa.

For example :

Dipu did not read book, can be represented as :

¬ READ (Dipu, Book).

A formula with a ¬ in front of it is called **a negation.**

d) The IF connective : ⇒

The "⇒" is used to represent "if-then" statements.

For example :

"If Nimu buys frock, then its colour is pink" can be represented as :

BUYS (Nimu, Frock) ⇒ COLOUR (Frock, Pink)

or

"If Dipu is caught in the rain then she uses umbrella".

CAUGHT IN (Dipu, Rain) ⇒ USES (Dipu, Umbrella)

A formula built by connecting two formulas with '⇒' is called as **implication**. The left hand side of an implication is called the **antecedant** and the right hand side is called the **consequent.**

e) The BICONDITIONAL connective : ⇔

P (⇔) Q, shows that it is true whenever both P ⇒ Q and Q ⇒ P are true. In English, this is often written as "P" if and only if "Q" or "P" iff "Q".

For example :

"Nimu becomes happy iff Nimu eats icecream." This sentence can be represented as,

BECOMES (Nimu, Happy) ⇔ EATS (Nimu, Icecream)

- **Truth table for five logical connectives :**

P	Q	¬ P	P ∧ Q	P ∨ Q	P ⇒ Q	P ⇔ Q
F	F	T	F	F	T	T
F	T	T	F	T	T	F
T	F	F	F	T	F	F
T	T	F	T	T	T	T

where T : True

F : False.

Following are Some of the Legal Sentences (WFF's) in Propositional Logic

1) The negation of a sentence is a sentence.

   For example :

   ¬ P and ¬ false are sentences.

2) The conjunction, 'and', of two sentences is a sentence.

   For example :

   P ∧ ¬ P is a sentences.

3) The disjunction, 'or', of two sentences is a sentence.

For example :

$P \vee \neg P$ is a sentence.

4) The implication of one sentence from another is a sentence.

For example :

$P \rightarrow Q$ is a sentence.

5) The equivalence of two sentences is a sentence.

For example :

$P \vee Q \equiv R$ is a sentence.

5.3.3 Grouping of Symbols in Propositional Logic

In propositional calculus, the symbols ( ) and [ ] are used to group the symbols into subexpressions by which their order of evaluation and meaning are controlled.

For example :

$(P \vee Q) \equiv R$ is different from $P \vee (Q \equiv R)$.

5.3.4 Forming a Legal Sentence in Propositional Logic

An expression is a sentence of propositional calculus if and only if it can be formed of legal symbols through some sequence of these rules.

$((P \wedge Q) \rightarrow R) \equiv \neg P \vee \neg Q \vee R$

is a well formed sentence in the propositional calculus because

i) P, Q and R propositions and thus sentences.

ii) $P \wedge Q$, the conjunction of two sentences.

$\therefore$ It is a sentence.

iii) $(P \wedge Q) \rightarrow R$ is the implication of a sentence for another.

$\therefore$ It is a sentence.

iv) $\neg P$ and $\neg Q$, the negations of sentences are sentences.

v) $\neg P \vee \neg Q$, the disjunction of two sentences, is a sentence.

vi) $\neg P \vee \neg Q \vee R$, the disjunction of two sentences, is a sentence.

vii) $((P \wedge Q) \rightarrow R) \equiv \neg P \vee \neg Q \vee R$

The equivalence of two sentences is a sentence.

5.3.5 Formal Grammar for Propositional Logic

A BNF (Backus Naur - Form) grammar of sentences in propositional logic, is as described below.

Sentence → Atomic sentence | Complex sentence.

Atomic sentence → True | False | Symbol.

Symbol → P | Q | R | ....

Complex sentence → ¬ Sentence

| (Sentence ∧ Sentence)

| (Sentence ∨ Sentence)

| (Sentence ⇒ Sentence)

| (Sentence ⇔ Sentence).

Notice that grammar is very strict about parentheses. Every sentence constructed with binary connectives must be closed in parentheses.

The order of precedence in propositional logic is from highest to lowest. The operators with their precedence order, are shown below,

¬ , ∧ , ∨ , ⇒ , ⇔.

Hence, the sentence,

¬ X ∨ Y ∧ Z ⇒ W

is equivalent to,

((¬ X) ∨ (Y ∧ Z)) ⇒ W.

5.3.6 Inference in the Knowledge Base

A logical inferencing means to decide whether KB $\models \alpha$ for some sentence α.

General inferencing algorithm (truth table enumeration algorithm)

1) This algo is used for deciding entailment in propositional logic.
2) It works like backtracking search algorithm.
3) It performs a recursive enumeration of a finite space of assignments to variables.
4) This algorithm is sound, because it implements directly the definition of entailment.
5) This algorithm is complete, because it works for any knowledge base and 'α', and always terminates (As there are finite models to check).
6) Time complexity of algorithm is $O(2^n)$ (assuming n symbol in all, then there are 2^n models).
7) Space complexity of algorithm is O(n), because the enumeration is depth first.

Note Every known inference algorithm for propositional logic has a worst case time complexity which is exponential in the size of the input.

Note Propositional entailment is CO-NP-complete.
(CO-NP stands for Complement - Nonpolynomial time, where as CO-NP-complete means hardest problem in CO-NP)

The functions for truth table enumeration algorithm :

Function TT Entails (KB, α) Returns True or False

Inputs : (KB, the knowledge base),
(a sentence in propositional logic α),
the query, a sentence in propositional logic.

Symbols $\leftarrow$ A list of the proposition symbols in KB and α

return TT CHECK-ALL (KB, α, symbols, [ ])

Function TT-CHECK-ALL (KB, α, symbols, model) returns true or false

```
if EMPTY (symbols) then
  if PL-TRUE (KB, model) then return
      PL-TRUE (α, model)
  else return true
else do
P ← FIRST (symbols); rest ←
      REST (symbols)
return TT-CHECK-ALL (KB, α, rest,
      EXTEND (P, true, model) and
TT-CHECK-ALL(KB, α, rest, EXTEND
      (P, false, model)
```

Note that,

1) In truth-table enumeration algorithm for deciding propositional entailment, TT stands for truth Table.
2) PL-TRUE returns true; if a sentence holds within a model.
3) The variable model represents a partial model-an assignment to only some of the variables.
4) The function call EXTEND (P, true, model) returns a new partial model in which P has the value true.

5.4 Semantics for Propositional Logic

The semantics define rules for determining the truth of a sentence with respect to a particular model. In propositional logic, a model simply fixes the truth value true or false for every propositional symbol.

The semantics for propositional logic should specify how to compute the truth value of any sentence, given a model. This is done recursively. All sentences are constructed from atomic sentences and the five connectives.

5.4.1 Interpretation of Propositions

1) An interpretation of a set of propositions is the assignment of a truth value, either T or F, to each propositional symbol.
2) The symbol true is always assigned T.
3) The symbol false is assigned F.
4) Propositions :
   i) A proposition is a statement which in English would be declarative sentence.
   ii) Every proposition is either true or false.
   iii) Propositions are sentences, either true or false but not both.
   iv) Sentence is smallest unit in propositional logic.
   v) If a proposition is true then its truth value is "True".
   vi) If a proposition is false then its truth value is "False".

5.4.2 Computation of Truth Values

1) The truth assignment of negation, $\neg$ P, where P is any propositional symbol, is F, if the assignment to P is T, and T if the assignment to P is F.
2) The truth assignment of conjunction, $\wedge$, is T only when both conjuncts have truth value T; otherwise it is F.
3) The truth assignment of disjunction, $\vee$, is F only when both disjuncts have truth value F, otherwise it is T.
4) The truth assignment of implication, $\rightarrow$ is F only when premise or symbol before the implication is T and the truth value of the consequent or symbol after the implication is F; otherwise it is T.
5) The truth assignment of equivalence, $\equiv$ is T only when both expressions have the same truth assignment for all possible interpretations; otherwise it is F.

For example :

Sentence	Truth value	Proposition (Y/N)
i) "Grass is green"	"True"	Yes
ii) "2 + 5 = 5"	"False"	Yes
ii) "Close the door"	-	No
iv) "Is it hot out side ?	-	No
v) "X > 2" where X is variable	-	No (Since X is not defined)
vi) "X = X"	-	No (It is unknown that what is 'X' and "="; "3 = 3" or "air is equal to air or "water is equal to water" has no meaning).

5.4.3 Truth Table

1) The truth assignments of compound propositions are often described by truth tables.
2) A truth table lists all possible truth value assignments to the atomic propositions of an expression and gives the truth value of the expression for each assignment.
3) Thus, a truth table enumerates all possible words of interpretation that may be given to an expression.

For example :

The truth table for $P \wedge Q$, lists truth values for each possible truth assignment of the operands. $P \wedge Q$ is true only when both P and Q are both T or ($\vee$), not ($\neg$), implies ($\rightarrow$), equivalence ($\equiv$) are defined in a similar fashion.

- **Truth Table demonstrating the equivalence of $P \rightarrow Q$ and $\neg P \vee Q$.**

P	Q	$\neg P$	$\neg P \vee Q$	$P \Rightarrow Q$	$(\neg P \vee Q) \equiv (P \Rightarrow Q)$
T	T	F	T	T	T
T	F	F	F	F	T
F	T	T	T	T	T
F	F	T	T	T	T

4) Equivalence of two sentences in propositional logic :

Two expressions in the propositional calculus are equivalent if they have the same value under all truth value assignments.

By demonstrating that two different sentences in the propositional calculus have identical truth tables, we can prove the following equivalences for propositional expressions P, Q and R.

i) $\neg (\neg P) \equiv P$ (Double-negation elimination)

ii) $(P \vee Q) \equiv (\neg P \rightarrow Q)$

iii) The contrapositive low :

$(P \rightarrow Q) \equiv (\neg Q \rightarrow \neg P)$

iv) De Morgan's Law :

$\neg (P \vee Q) \equiv (\neg P \wedge \neg Q)$ and $\neg (P \wedge Q) \equiv (\neg P \vee \neg Q)$

v) The commutative law :

$(P \wedge Q) \equiv (Q \wedge P)$ and $(P \vee Q) \equiv (Q \vee P)$

vi) The associative law :

$((P \wedge Q) \wedge R) \equiv (P \wedge (Q \wedge R))$

vii) The associative law :

$((P \vee Q) \vee R) \equiv (P \vee (Q \vee R))$

viii) The distributive law :

$P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$

ix) The distributive law :

$P \wedge (Q \vee R) \equiv (P \wedge Q) \vee (P \wedge R)$

x) Implication elimination :

$(P \rightarrow Q) \equiv (\neg P \vee Q)$

xi) Biconditional elimination :

$(P \Leftrightarrow Q) \equiv ((P \rightarrow Q) \wedge (Q \rightarrow P))$

5.4.4 Validity

A sentence is valid if it is true in all models.

For example :

(Parrot is green $\vee$ Parrot is not green) is valid, one of the two holds.

Valid sentence is called Tautology. They are necessarily true.

5.4.5 Satisfiability

A sentence is satisfiable if it is true in some model.

If a sentence S is true in model m, then we say that m satisfies S or m is a model of S.

5.4.6 A Complete Example of Knowledge-based Agent

A wumpus world agent :

Wumpus World is a classic artificial intelligence problem, which is used to demonstrate various aspects based on simulation, as well as other AI concepts.

Wumpus was an early computer game in which an agent had to explore a cave made up from a series of interconnected rooms. In one of the rooms in the cave, there was a Wumpus which would kill the agent if it entered that room. Some rooms contained pits, and the agent would die if it entered any of those rooms too. The agent had one arrow with which it could kill the Wumpus. The goal was to locate the gold that was hidden some where in the cave and return to the start without getting killed.

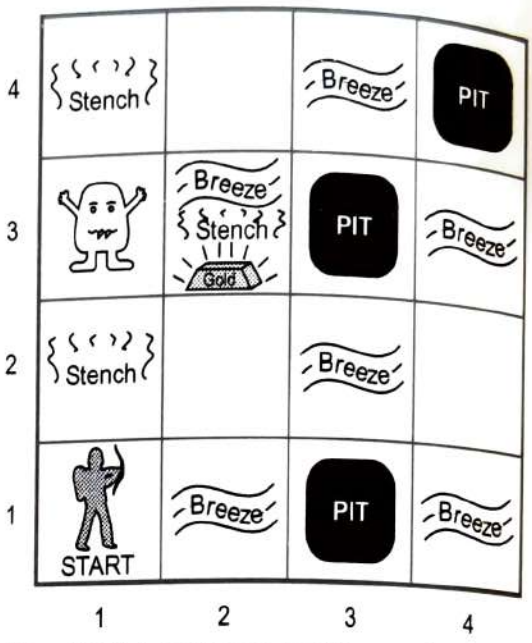


Fig. 5.4.1 A wumpus world agent and its environment

PEAS description :

- **Performance measure :**
  Gold : + 1000, **Death :** – 1000
  – 1 per step, – 10 for using the arrow.
- **Environment :**
  - Squares adjacent to Wumpus are **smelly**.
  - Squares adjacent to pit are **breezy**.
  - **Glitter** iff gold is in the same square.
  - Shooting kills Wumpus if you are facing it. It **screams**.
  - Shooting uses only arrow.
  - Grabbing picks up gold if in same square.
  - Releasing drops the gold in same square.
  - You **bump** if you walk into a wall.
- **Actuators :**
  Left turn, right turn, forward, grab, release, shoot.
- **Sensors :**
  Stench, breeze, glitter, bump, scream.

Wumpus world characterization :

1) Deterministic

   Yes - outcomes exactly specified.

2) Static

   Yes - Wumpus and Pits do not move.

3) Discrete

   Yes

4) Single - agent

   Yes - Wumpus is essentially a natural feature.

5) Fully observable

   No - only **Local** perception.

Exploring the wumpus world :

1) The knowledge base initially contains the rules of the environment.

2) Location : [1, 1]

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 OK	2,2	3,2	4,2
1,1 A OK	2,1 OK	3,1	4,1

A = Agent
B = Breeze
G = Glitter gold
OK = Soft square
P = Pit
S = Stench
V = Visited
W = Wumpus

Fig. 5.4.2 Exploring wumpus world-I

Percept : [¬ Stench, ¬ Breeze, ¬ Glitter, ¬ Bump, ¬ Scream]

Action : Move to safe cell [2, 1].

3) Location : [2, 1]

Percept : [¬ Stench, Breeze, ¬ Glitter, ¬ Bump, ¬ Scream].

Infer : Breeze indicates that there is a pit in [2, 2] or [3, 1].

Action : Return to [1, 1] to try next safe cell.

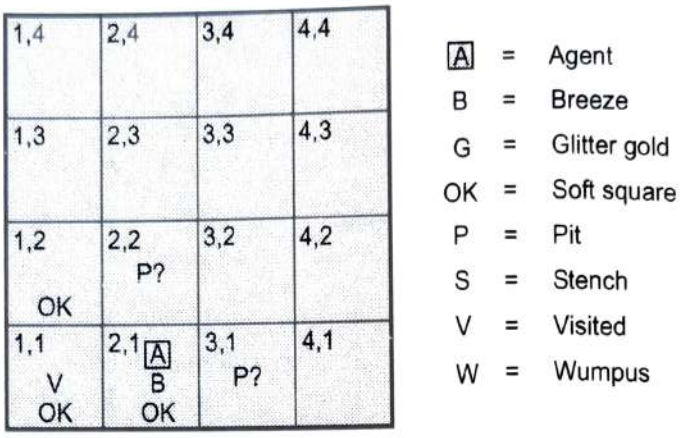


Fig. 5.4.3 Exploring wumpus world - II

4) Location : [1, 2]

Percept : [Stench, ¬Breeze, ¬ Glitter, ¬ Bump, ¬ Scream].

Infer : Wumpus in [1, 3] or [2, 2]

YET ..... not in [1, 1]

Thus .... not in [2, 2] or stench would have been detected in [2, 1]

Thus .... Wumpus is in [1, 3] [2, 2] is safe because of lack of breeze in [1, 2]

Thus .... pit in [3, 1]

Action : Move to next safe cell [2, 2].

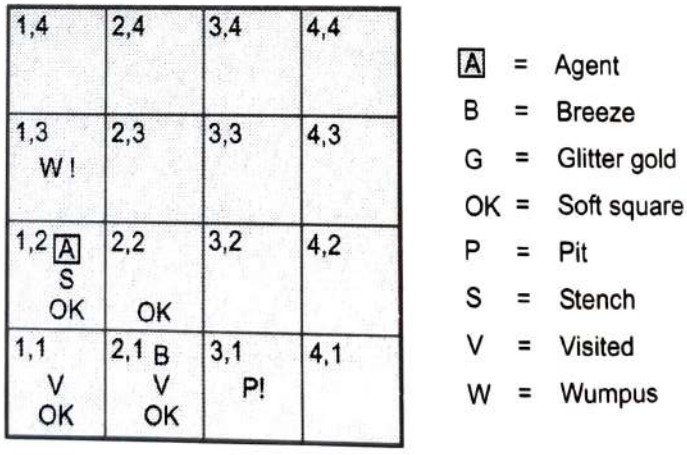


Fig. 5.4.4 Exploring wumpus world - III

5.5 Reasoning Patterns in Propositional Logic

There are standard patterns that can be applied to derive chains of conclusions that lead to the desired goal. These pattern of inference are called inference rules.

5.5.1 The Concept of Monotonicity

1) It says that the set of entailed sentences can only increase as information is added to the knowledge base.

2) Monotonicity means that inference rules can be applied whenever suitable premises are found in the knowledge base. The conclusion of the rule must follow regardless of what else is in the knowledge base.

5.5.2 Inference Rules

1) Modus Ponens

It is represented as,

$$\frac{\alpha \Rightarrow \beta, \alpha}{\beta}$$

It means that whenever any sentences of the form $\alpha \Rightarrow \beta$

and α are given then the sentence β can be inferred.

For example -

If two statements,

i) (Signal Pole Ahead $\wedge$ Signal Red) $\Rightarrow$ Stop

ii) (Signal Pole Ahead $\wedge$ Signal Red) are given then

"Stop" can be inferred.

2) And-elimination : It says that from conjunction any conjuncts can be inferred.

It is represented as,

$$\frac{\alpha \wedge \beta}{\alpha}$$

For example -

From the sentence,

(Signal Pole Ahead $\wedge$ Signal Red)

The inference,

"Single Red".

can be drawn.

3) Unit resolution [$P \vee Q, \neg Q \vdash P$] :

Unit resolution rule takes a clause - a disjunction of literals and a literal and produce a new clause.

Rule is,

$$\frac{l_1 \vee ... \vee l\,k,\ m}{l_1 \vee ... \vee l_{i-1} \vee l_{i+1} \vee ... \vee l\,k}$$

where each l is literal. l_i and m are complementary literals. [It means that one is the negation of other].

4) Resolution : The unit resolution rule can be generalized to the full resolution rule,

$$\frac{l_1 \vee ... \vee l_k, m_1 \vee ... \vee m_n}{l_1 \vee ... \vee l_{i-1} \vee l_{i+1} \vee ... \vee l_k \vee m_1 \vee ... \vee m_{j-1} \vee m_{j+1} \vee ... \vee m_n}$$

where l_i and m_j are complementary literals. If we are dealing only with clauses of length two we could write this as

$$\frac{l_1 \vee l_2, \neg l_2 \vee l_3}{l_1 \vee l_3}$$

That is, resolution takes two clauses and produces a new clause containing all the literals of the two original clauses except the two complementary literals.

It is a single inference rule, that yields a complete inference algorithm when coupled with any complete search algorithm.

Conjunctive normal form

1) The resolution rule applies only to disjunctions of literals, so it would seem to be relevant only to knowledge bases and queries consisting of such disjunctions.
2) Every sentence of propositional logic is logically equivalent to a conjunction of disjunctions of literals.
3) A sentence expressed as a conjunction of disjunctions of literals is said to be in Conjunctive Normal Form (CNF).
4) It is used for representing sentences.

Steps for converting propositional logic sentences into CNF

1) Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$:

   For example -

   Consider following statement,

   $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$

   After eliminating $\Leftrightarrow$ in above sentence,

   $(B_{1,1} \rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \rightarrow B_{1,1})$

2) Eliminate $\rightarrow$, replacing $\alpha \rightarrow \beta$ with $\neg \alpha \vee \beta$:

   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg (P_{1,2}) \vee P_{2,1} \vee B_{1,1})$

3) CNF requires $\neg$ to appear only in literals, so we "move $\neg$ inwards" by repeated application of the following standard equivalences : -

$\neg (\neg \alpha) \equiv \alpha$ (clouble - negation elimination)

$\neg (\alpha \wedge \beta) \equiv (\neg \alpha \vee \neg \beta)$ (De Morgan)

$\neg (\alpha \vee \beta) \equiv (\neg \alpha \wedge \neg \beta)$ (De Morgan)

In the example, we require just one application of the last rule.

$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg (P_{1,2} \wedge P_{2,1}) \vee B_{1,1})$

4) Now we have a sentence containing nested $\wedge$ and $\vee$ operators applied to literals. We apply the distributivity law, distributing $\vee$ over $\wedge$ wherever possible.

$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \wedge (\neg P_{2,1} \vee B_{1,1}) \vee B_{1,1})$

Above sentence is now in CNF.

Resolution rule forms the basis for a family of complete inference procedures.

- **Refutation completeness :**

It means that resolution can always be used to either confirm or refute a sentence but it cannot be used to enumerate true sentences.

Resolution algorithm

1) It takes input as knowledge base [a sentence in propositional logic] and α (it is query - which is a sentence in propositional logic).
2) Its output is true or false means that knowledge base do resolves α. It shows that knowledge base $\models \alpha$, for this it will be shown that $(KB \wedge \neg \alpha)$.
3) Steps are as follows,
   i) $(KB \wedge \neg \alpha)$ is first converted in to CNF.
   ii) Resolution rule is applied to the resulting clauses.
   iii) Each pair that contains complementary literals is resolved to produce new clause.
   iv) This new clause is added to set if it is not already present.
   v) Step (ii) to (iv) are repeated until one of the two following situations occurs.

a) There are no new clauses that can be added, in which case α does not entail β

OR

b) An application of the resolution rule derives the empty clause, in which case α entails β.

Completeness of resolution

The algorithm of resolution is complete. It means that it will surely produces a result checking all clauses in KB and all the clauses derivable from them. It finds closure set of clauses set which has all derivable clauses from existing clauses set. It checks the input query against the closure set.

Ground resolution theorem

It states that – "If a set of clauses is unsatisfiable, then the resolution closure of those clauses contains the empty clause".

The ground resolution theorem is called as completeness theorem for resolution.

5.6 Forward and Backward Chaining

5.6.1 The Horn Clause

1) Knowledge base contains simple restricted clauses called as Horn clauses. A Horn clause is a disjunction of literals of which at most one is positive.

2) Horn clause :

- It has atmost one positive literal.
- Horn clause with exactly one position literal are called definite literal.
- The positive literal is called HEAD of the clause.
- The negative literal is called as BODY of the clause.
- A definite clause with no negative literals is called as fact.
- Horn clause with no positive literals can be written as an implication whose conclusion is the literal false. Such horn clause can be used to represent integrity constraints (conditions) in the databases.

3) Inferencing with HORN clause - For inferencing with Horn clause forward and backward chaining methods are used.

5.6.2 Forward Chaining Method

The concept :

1) It is general concept of data-driven reasoning in which the focus of attention starts with known data.
2) It can be used within an agent to derive conclusions from incoming percepts.
3) It is used for determining whether a single proposition symbol Q (the query) is entailed by the knowledge base of Horn clauses.
4) It initially starts from known facts (positive literals) in the knowledge base.

5) If all the premises of an implication are known then its conclusion is added to the set of known facts.

For example :

If $L_{1,1}$ and breeze are known and $(L_{1,1} \wedge \text{Breeze}) \Rightarrow B_{1,1}$ is in the KB then $B_{1,1}$ can be added.

6) This process continues until the query Q is added or until no further inferences can be made.

Forward chaining algorithm :

```
Function PL-FC-ENTAILS ? (KB, q)
      returns true or false
Inputs : KB, the knowledge base, a set of propositional Horn clauses, q, the query, a
   proposition symbol.
Local variables :  Count, a table, indexed by clause, initially the number of premises.
                   inferred, a table, indexed by symbol, each entry initially false.
                   agenda, a list of symbols, initially the symbols known to be true in KB.
while agenda is not empty do
   p ← POP (agenda)
unless inferred [p] do
   inferred [p] ← true
For each Horn clause (in whose premise p appears do decrement count [c])
   if count [c] = o then do
      if HEAD [c] = q then return true
      PUSH (HEAD [c], agenda)
return false.
```

- **Properties of forward chaining algorithm**

1) Forward chaining algorithm is sound
   - It means that every inference is essentially an application of modus ponens.
2) Forward chaining is complete.
   - It means that every entailed atomic sentence will be derived. Algorithm constructs inferred table for final state. In this table each symbol inferred during process has true value and false for all other symbols.
3) For example :

   Consider a simple knowledge base of Horn clause, as given below.

   1. $A \Rightarrow Q$
   2. $B \wedge C \Rightarrow A$

3. $D \wedge B \Rightarrow C$
4. $E \wedge A \Rightarrow B$
5. $E \wedge D \Rightarrow B$
6. E
7. D

AND - OR graph for above Horn clause knowledge base

1) Multiple links joined by arc indicate a conjunction.
2) Multiple links without an arc indicate a disjunction.
3) The links should be proved.
4) Inforce propagates up the graph.
5) Before proceeding ahead, from conjunction point, the propagation waits until all the conjuncts are known.

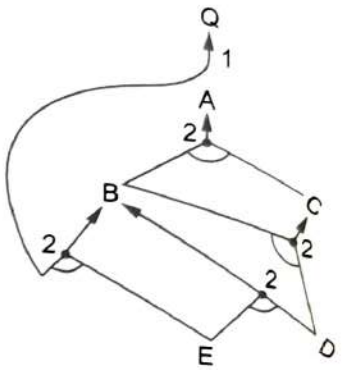


Fig. 5.6.1 AND-OR graph for Horn clause KB

- **The order of clauses considered by forward chaining algorithm for inferencing query Q :**
  1) E and D are given true.
  2) E and D implies B also B and A implies B.
  3) D and B implies C.
  4) B and C implies A.
  5) A implies Q.
  6) Therefore Q is true.

5.6.3 The Backward Chaining Algorithm

The concept :

1) It works backwards from the query i.e if query Q is known to be true then no work is needed.
2) Algorithm finds those implications in the knowledge base that conclude Q.

   If all the premises of one of those implications can be proved true (by backward chaining) then Q is true.
3) Backward chaining works down the graph starting at Q until it reaches a set of known facts that forms the basis for a proof.
4) Backward chaining is a form of goal-directed reasoning.

5) It is useful for answering questions such as "what action should I do now" ?

6) Cost of backward chaining is equal to or less than linear in the size of knowledge base, [As the process touches only relevant facts].

The example - [Referring to same knowledge base considered in forward chaining]

7) The order of clauses considered by backward chaining algorithm for inferecing query Q (in backward style) ---

1. Q is implied by A.
2. A is implied by B and C.
3. B is implied by E and A as well as E and D.
4. C is implied by E and D.
5. E and D are given true in knowledge base.
6. Therefore from Q we found implications sequence that conclude Q.

Therefore Q is true.

5.7 Effective Propositional Inference

For better inferences we need efficient algorithm which are based on model checking. Algorithm approach can be backtracking search or hill-climbing search. The algorithms we are going to discuss are for checking satisfiability.

5.7.1 A Complete Backtracking Algorithm [Davis Putnam Logmann-Loveland Algorithm-(DPLL)]

The concept :

1) It takes input as a sentence in conjunctive normal form, which is a set of clauses.

2) It do recursive depth first enumeration of possible models.

3) It determines if input propositional logic sentence (in CNF) is satisfiable.

DPLL- A improved algorithm

This algorithm is improvement over earlier general inferencing algorithm for proposition logic (The TT - Entails algorithm). There are three improvements over truth table enumeration (TT - entails algorithm) as described below,

1) Early termination :

i) A clause is true if any literal is true.

ii) A sentence is false if any clause is false.

2) Pure symbol heuristic :

i) **Pure symbol :** Pure symbol always appears with the same "sign" in all clauses.

Example :

In the three clauses $(A \vee \neg B)$, $(\neg B \vee \neg C)$, $(C \vee A)$, A and B are pure, C is impure.

ii) Make a pure symbol literal true.

3) Unit clause heuristic :

i) Unit clause : It is a clause which have only one literal in the clause.

ii) The only literal in a unit clause must be true.

Example :

If the model contains B = false, then $(B \vee \neg C)$ becomes a unit clause because it is equivalent to $(\text{false} \vee \neg C)$ or just $\neg C$.

The DPLL algorithm for checking satisfiability of a sentence in propositional logic

```
Function DPLL-SATISFIABLE (S) returns true or false
Inputs : S, a sentence in propositional logic.
Clause ← The set of clauses in the CNF representation of S.
symbols ← A list of the proposition symbols in S.
return DPLL (clauses, symbols, [ ])
```

DPLL algorithm (the searching procedure)

```
Function DPLL (clauses, symbols, model)
    returns true or false
if every clause in clauses is
    true in model then return true
if some clause in clauses is false
    in model then return false
P, value ← FIND-PURE-SYMBOL (symbols, clauses, model)
if P is non-null then return DPLL
    (clauses, symbols, -P, EXTEND (P, value, model)
P value ← FIND-UNIT-CLAUSE (clauses, model)
if P is non-null then return DPLL
    (clauses, symbols-P, EXTEND (D, value, model)
P ← FIRST (symbols); rest ← REST (symbols)
    return DPLL (clauses, rest,
    EXTEND (P,true, model)) or DPLL (clauses, rest, EXTEND (P, false, model)).
```

Notes i) FIND-PURE-SYMBOL is a function which returns pure symbol (with value) or null.

ii) FIND-UNIT-CLAUSE is a function which returns a unit clause (with value) or null.

5.8 Local Search Algorithms for Inferencing in Propositional Logic

The local search algorithm like Hill-climbing and simulated-annealing can be applied directly to satisfiability problems. They work properly when correct evaluation function is provided to them. The task of algorithm is to find an assignment that satisfies every clause. Therefore, an evaluation function that counts the no. of unsatisfied clauses will be a good evaluation function choice.

5.8.1 Local Search Algorithms Characteristics

1) These algorithm take steps in the space of complete assignments, flipping the truth table value of one symbol at a time.
2) The space usually contains many local minima, to escape from which various forms of randomness are required.
3) The algorithm is expected to achieve balance between greediness and randomness.

5.8.2 Local Search Algorithm for Checking Satisfiability

The WALKSAT algorithm

[It is a **"random walk"** algorithm implementation for satisfiability].

Characteristics of WALKSAT algorithm :

1) It is local search algorithm.
2) Evaluation function :

   The evaluation function is the min-conflict heuristic of minimizing the number of unsatisfied clauses.
3) It balances between greediness and randomness.
4) It is more useful when we expect a solution to exist. for e.g. 8 queen problem.
5) It is incomplete algorithm. It return satisfying model when it succeed but when it fails it has problem. When algorithm fails to satisfy the sentence it may enter into infinite loop because of infinite values of max-flips.
6) Local search algorithm like WALKSAT, cannot always detect unsatisfiability, which is very necessary for deciding entailment.

For example :

An agent cannot reliably use local search to prove that a square is safe in the Wumpus world.

5.8.3 Steps in WALKSAT Algorithm

```
Function WALKSAT (Clauses, p, max, flips) returns a satisfying model or failure.
    Inputs : Clauses, a set of clauses in propositional logic. p, the probability of choosing to
    do a "random walk" move, typically around 0.5 max-flips, number of flips allowed
        before giving up.
    Model ← A random assignment of true/false to the symbols in clauses.
      For i=1 to max-flips do
    if model satisfies clauses then return model
    with probability p flip  the value in
      model of a randomly selected
      symbol from clause
    else flip whichever symbol in clause
      maximizes the number of satisfied clauses
return failure.
```

5.8.4 Hard Satisfiability Problems

Easy problems can be solved using any old algorithm but we need algorithm that can solve hard problems.

If we look at satifiability problems in CNF then we can have a under constrained (fixed and minimum constraints with some initial assignments to variables) problem with few clauses.

For example :

Following is 3-CNF sentence [in 3-CNF each clause contains three randomly generated distinct symbols] with five symbols and five clauses.

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

$\wedge (\neg C \vee \neg B \vee E) \wedge (E \vee \neg D \vee B) \wedge (B \vee E \vee \neg C)$

Here 16 of the total 32 possible assignments are model of this sentence therefore on an average it would just take two random guesses to find a model.

It can be stated that,

if m = Number of clauses

n = Number of symbols.

Then we can calculate probability of satisfiability as the ratio m/n,

That is,

$$\text{Probability of satisfiability} = \frac{\text{Number of clauses}}{\text{Number of symbols}}$$

As we can see that, for above 3 CNF sentence this probability,

(5 clauses/5 symbols =1) is exactly 1.

For small values of m and n this probability will be near to 1.

For large values of m and n this probability will be near to 0.

When the m/n = 4.3 the probability drops down at mid level and remains constant. It **means that at this stage it is "nearly satisfiable" or as well "nearly unsatisfiable". This is called as critical point while satisfying clauses.** If we are solving a problem where in critical points do come then it is a **hard problem.**

➤ **Note that :**

1) Problems near critical points (hard problems) are mush more difficult than other random problems.
2) DPLL works effectively on harder problems

   It takes averagly few thousand steps compared with $2^{50} \approx 10^{15}$ for truth table enumeration.
3) WALKSAT is much faster than DPLL for all ranges of problems.
4) Practically, for general purpose problem solutions we can combine min-conficts heuristic and random walk behaviour.

5.9 Knowledge based Agents

5.9.1 Inference based Agent in Wumpus World - Agent based on Propositional Logic

It uses inference algorithms and knowledge base like a general knowledge based agent.

Wumpus world

1) This agent reasons logically about the location of pits (P), Wumpus (W), and safe squares (S).
2) It begins with a knowledge base that states the "physics" of the Wumpus world.
3) The agent begins its exploration from a square known to be safe.
   - $R_1 : \neg P_{1,1}$
   - $R_2 : \neg W_{1,1}$

4) For every square [x, y], if the agent perceives breeze....

$-R_3 - R_{18} : B_{x,y} \Leftrightarrow (P_{x,y+1} \vee P_{x,y-1} \vee P_{x+1,y} \vee P_{x-1,y})$

5) For every square [x, y], if the agent perceives stench....

$- R_{19} - R_{34} : B_{x,y} \Leftrightarrow (P_{x,y+1} \vee P_{x,y-1} \vee P_{x+1,y} \vee P_{x-1,y})$

6) How do we express there is exactly one Wumpus ?

We can represent it in 2 sentences,

a) There is at least one Wumpus $R_{35} : W_{1,1} \vee W_{1,2} \vee .... \vee W_{4,4}$.

b) For any two squares, one must be Wumpus-free.

i) This should be stated for every pair of squares.

ii) [n(n-1)]/2 number of sentences.

$-R_{36} : \neg W_{1,1} \vee \neg W_{1,2}$

$-R_{37} : \neg W_{1,1} \vee \neg W_{1,3}$

:

:

$-R_{155}$:

7) We have used 64 distinct symbols in the knowledge base.

Wumpus-World-Agent program

1) ASK-Entailment computation :

TT-Entails would have to enumerate 2^{64} rows.

DPLL/WALKSAT performs better. DPLL works because it uses unit propagation heuristic. WALKSAT suffers from incompleteness.

2) For a large Wumpus world, the number of sentences in the knowledge base will be huge.

3) We could not capture the property "breeze in a square indicates pit in at least one of directly adjacent square".

Location, orientation in wumpus world

1) Can we add propositions like,

$L_{1,1} \wedge$ Facing Right $\wedge$ Forward $\rightarrow L_{2,1}$.

2) Knowledge base will entail both $L_{1,1}$ and $L_{2,1}$ by using inference rules.

3) What we intended to capture is this

$L_{1,1}^{(t)} \wedge$ Facing Right $\wedge$ Forward $\rightarrow L_{2,1}^{(t+1)}$

Facing Right $\wedge$ Turn Left $^{(t)} \rightarrow$ Facing Up $^{(t+1)}$

4) For every times step, one such statement ???

Assuming the problem is solvable in 100 times steps, then 10 thousands of additional sentences are needed.

5.9.2 Circuit based Agent in Wumpus World

These are agents which evaluate logical expression directly in the form of circuits.

Characteristics of circuit based agent

1) Reflex agent with state.
2) Percepts are inputs to sequential circuit.

   It has network of gates and registers.
3) Outputs are registers corresponding to actions.
4) Circuits are evaluated in a data flow model.
5) Value stored at each proposition symbol gives the truth value of the corresponding symbol at the current time t.
6) State estimation, is the general task of keeping track of environment state given a stream of percepts.
7) For logic based systems : Maintain a representation of the set of all logically possible world states, given axioms and percepts.
8) Basic trick : Successor-state axioms define truth of proposition at t+1 from propositions at t.
9) For example :

$$\text{Alive}^t \Leftrightarrow \text{Scream}^t \wedge \text{Alive}^{t-1}$$

$$L^t_{1,1} \Leftrightarrow (L^{t-1}_{1,1} \wedge (\neg\, \text{Forward}^{t-1} \vee \text{Bump}^t)) \vee$$

$$(L^{t-1}_{1,2} \wedge (\text{Facing Down}^{t-1} \wedge \text{Forward}^{t-1})) \vee (L^{t-1}_{2,1} \wedge (\text{Facing Left}^{t-1} \wedge \text{Forward}^{t-1}))$$

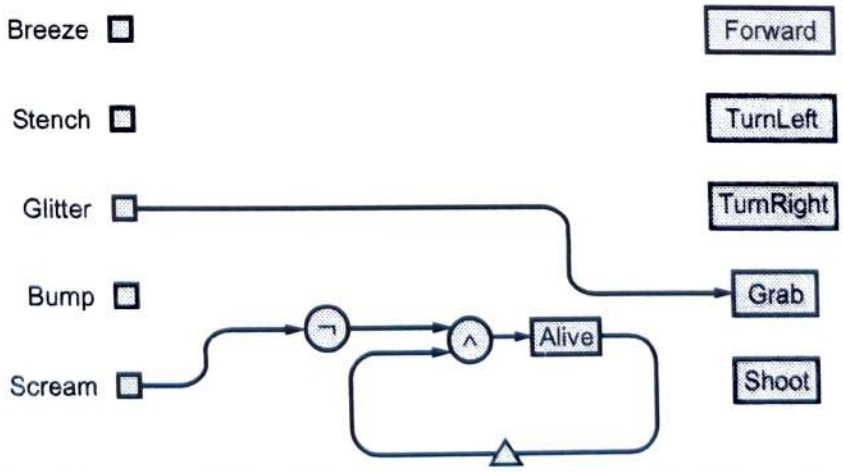


Fig. 5.9.1 Part of a circuit based agent for the wumpus world. Figure depicts circuit for grabbing the gold and the circuit for determining whether the wumpus is alive

Note
- Registers are shown as rectangles.
- One-step delays are shown as small rectangle.

Circuit based agent's location

1) One register for each L_{xy}
2) Agent is in [1, 1] at t if
   - The agent was in [1, 1] at t^{-1} and has no moved, tried but bump.
   - It was at [1, 2] facing down and moved forward.
   - It was at [2, 1] facing left and move forward.

$$L^{t}_{1,1} < > L^{t-1}_{1,1} \wedge (\neg \text{Forward}^{t-1} \vee \text{Bump}^{t}))$$

$$\vee (L^{t-1}_{1,2} \wedge (\text{Facing Down}^{t-1} \wedge \text{Forward}^{t-1}))$$

$$\vee (L^{t-1}_{2,1} \wedge (\text{Facing Left}^{t-1} \wedge \text{Forward}^{t-1}))$$

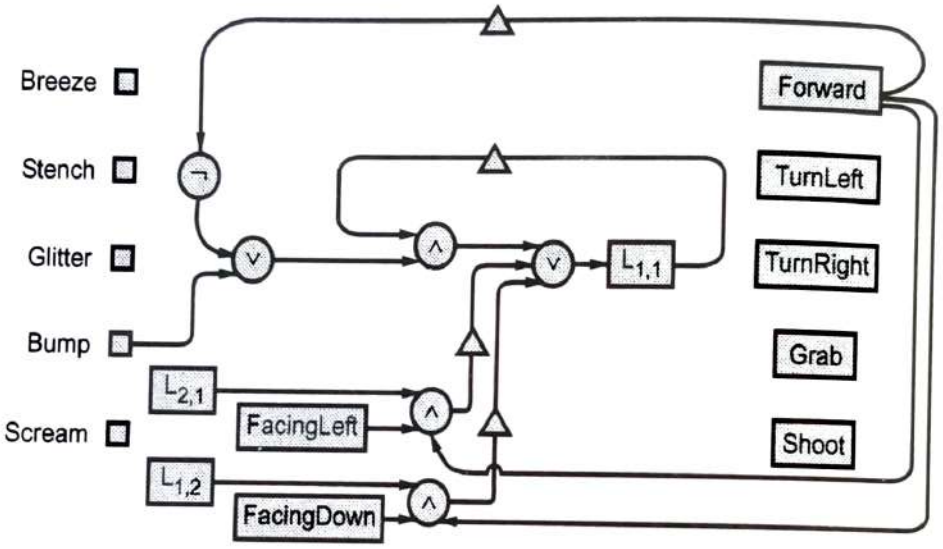


Fig. 5.9.2 The circuit for determining if the agent is at [1, 1]. Each location and orientation register has a similar circuit attached

A problem related to circuit base agent

What will be the initial truth value contained in $B_{4,4}$.

- The agent cannot set a truth value for it.
- Need to represent unknown, environment.
- Use of knowledge propositions

  $K(B_{4,4})$, $K(\neg B_{4,4})$

5.9.3 Comparison between Inference Based Agent (IBA) and Circuit Based Agent (CBA)

1) Conciseness

IBA - It require seperate copies of its knowledge base for every time step.

CBA - It do not require seperate copies of its knowledge base for every time step.

Both agents requires physics (expressed as sentences or circuits) for every square and therefore they are not suitable for large scale environment.

2) Computational efficiency

IBA - It takes exponential time for inferencing. (Time grows exponentially with respect to number of symbols)

CBA - It takes linear time for inferencing which is dependent on circuit size.

3) Completeness

IBA :

1) It is complete but with huge time requirements for completeness.
2) It also stores previous states/percept therefore memory requirement is also more.

CBA :

1) It is incomplete for various reasons :
   a) Acyclicity restriction in circuit itself which is practically not possibles.
   b) For a complete circuit CBA will take exponential time with respect to circuit size.
2) It forgets all previous states therefore can not draw conclusions based on previous states/percepts.

4) Ease of construction

IBA - As environment and conditions are simple and clear. It is easy to represent in propositional logic language.

CBA - The environment considered is limited therefore a circuit is small acyclic and semicomplete. Hence describing it is easy when relation between percepts and action is simple and straightforward. It is easy to describe and construct knowledgebase for such agents.

Answer in Brief

1. *What is propositional logic ? Explain the knowledge representation using propositional logic.* **(Refer section 5.1)**
2. *Give the five logical connectives used to construct complex sentences and give the formal grammar of propositional logic.* **(Refer section 5.3)**
3. *Explain propositional logic.* **(Refer section 5.1)**
4. *What are the limitations in using propositional logic to represent the knowledge base ?*

Ans. : Propositional logic has following limitations to represent the knowledge base.

1) It has limited expressive power.

2) It cannot directly represents properties of individuals or relations between individuals.

3) Generalizations, patterns, regularities cannot easily be represented.

4) Many rules (axioms) are required to write so as to allow inferencing.

5. *Give the five logical connectives used to construct complex sentences and give the formal grammar of propositional logic.* **(Refer section 5.1)**

6. *Write the algorithm for deciding entailment in propositional logic.* **(Refer section 5.1)**

5.10 University Question with Answer

Summer - 20

Q.1 *Define propositional logic.* **(Refer section 5.1)** [1]

❑❑❑

6 First Order Logic - Predicate Logic

Syllabus

First Order Logic : *Representation Revisited, Syntax and Semantics of First Order logic, Using First Order logic.*

Inference in First Order Logic : *Propositional Versus First Order Inference, Unification, Forward Chaining, Backward Chaining, Resolution.*

Contents

6.1 First Order Logic - [Predicate Logic]

GTU : Winter-18, Summer-20

6.1.1 Introduction to First Order Logic

It is a expressive language which has well defined syntax and semantics. It uses three things to represent the world model :

1) Objects :

People, houses, numbers, theories, colours, games, wars etc. These are nouns and nounphrases in natural languages.

2) Relations :

Red, round, prime, brother, bigger than, inside, owns, comes between etc. These are associations among objects or properties of objects. Relations can be unary such as green or n-ary such as brother_of.

3) Function :

Father_of, best_friend, one_more_than etc.

It is a relation in which there is only one "value" for a given input.

For example : "Two plus five equals seven".

Objects - Two, five, seven.

Relation - equals.

Function - plus.

("Two plus five" is the name for the object that is obtained by applying the function plus to the objects two and five).

For example : "Multitalented King Krishna Ruled Dwarika."

Objects - Krishna, Dwarika.

Relation - Ruled.

Functions - King, Multitalented.

6.1.2 Properties of First Order Logic

1. It has ability to represent facts about some or all of the objects in the universe.
2. It enables to represent law and rules extracted from real world.
3. It is useful language representation in mathematics, philosophy and AI related fields.
4. It represents facts in more realistic manner rather than just the true or false statement.

5. First order logic makes ontological commitment.

   Ontological commitment means what assumptions language makes about the nature of reality. First order logic assumes that real world consists of objects and certain relationship they hold or not among them.

6.1.3 First Order Logic-Extension to Propositional Logic

First order logic extends propositional logic in two directions :

1) It provides an inner structure for sentences. They are viewed as expressing relations between objects or individuals.
2) It provides a means to express, and reason with, generalizations. It makes it possible to say that a certain property holds for objects, or for some objects, or for no object.

6.1.4 Variations of First Order Logic

1. Temporal logic :

   It assumes that facts hold at some particular times. These times also have orders.

2. High order logic :

   It views the relations and functions referred to by first order logic as objects in themselves.

6.1.5 Characteristics of Logic

1) Epistemological commitments :

   It is ability to show, the possible states of knowledge that it allows with respect to each fact.

   From AI agent point of view, any sentence can be true, false or has no opinion states depending on the knowledge base.

2) Probability theory :

   It is ability to assign a degree of probability to the sentence ranging from 0 to 1. It can be a belief probability for agent.

   A high probability sentence would be a better choice for AI agent when it needs to think.

6.1.6 Formal Languages and their Ontological and Epistemological Commitments

Language	Ontological commitment (what exists in the world)	Epistemological commitment (What an agent belives about facts)
Propositional logic	Facts	true/false/unknown
First-order logic	Facts, objects, relations	true/false/unknown
Temporal logic	Facts, objects, relations, times	true/false/unknown
Probability theory	Facts	degree of belief $\in [0,1]$
Fuzzy logic	Facts with degree of truth $\in [0, 1]$	known interval value

6.1.7 Higher-Order Logic

- FOPL is called first-order because it allows quantifiers to range over objects (terms) but not properties, relations or functions applied to those objects.
- Second-order logic allows quantifiers to range over predicates and functions as well :

  "X" Y [(x=y) U ("p p(x) U" p(y))]

  Says that two objects are equal if and only if they have exactly the same properties.

  "f" g [(f=g) U ("X f(x) = g(x))]

  Says that two functions are equal if and only if they have the same value for all possible arguments.
- Third-order would allow quantifying over predicates of predicates, etc.
  For example : A second-order predicate would be symetric (p) stating that a binary predicate p represents a symmetric relation.
- Higher order logic views the relations and functions referred by first-order logic as objects in themselves.
- Higher order logic makes further ontological commitments.

6.1.8 Syntax for First Order Logic

1. Model :

Model of logical language are the formal structures that constitute the possible world under consideration.

In first order language, models have objects in them.

2. Domain of a model :

It is a non-empty set of objects in model. These objects are also referred to as domain elements.

3. Relation :

It is association among objects. It is represented as a tuple. It is a set of tuples of objects that are related.

Note *Tuple is collection of object in some specific order, written inside angle brackets.*

4. Function :

Is a special type of relation in that a given object must be related to exactly one object in this way.

5. Symbols :

Smallest syntatic element in first order logic are symbols, that can stand for objects, relations, functions.

- **Following are the first order logic's symbols :**

1. **Truth symbols :**

   Truth symbols true and false. These are reserved symbols.

2. **Constant symbols :**

   Constant symbols are symbol expressions having the first character lowercase, they stand for objects.

   Example : Cindrella, A, B, Ram, Red, etc.

3. **Variable symbols :**

   These are symbol expressions beginning with an uppercase character and designate unspecified objects.

   Example : X, Y, Z, etc.

4. **Predicate symbols :**

   Predicate symbols are symbols beginning with a lowercase letter and they stand for relations.

   Example : Stepsister, likes, color, etc.

5. **Function symbols :**

   Function symbols are symbol expressions having the first character lowercase and they stand for functions. Functions have an attached arity indicating the number of elements of the domain mapped onto each element of the range.

A function expression consists of a function constant of arity n, followed by 'n' terms $t_1, t_2, t_3 .... t_n$ enclosed in parentheses and separated by commas.

Example : RightLegShoe, FatherOF, ColorOF etc.

- **A example depicting various synatic terms used in first order logic :**

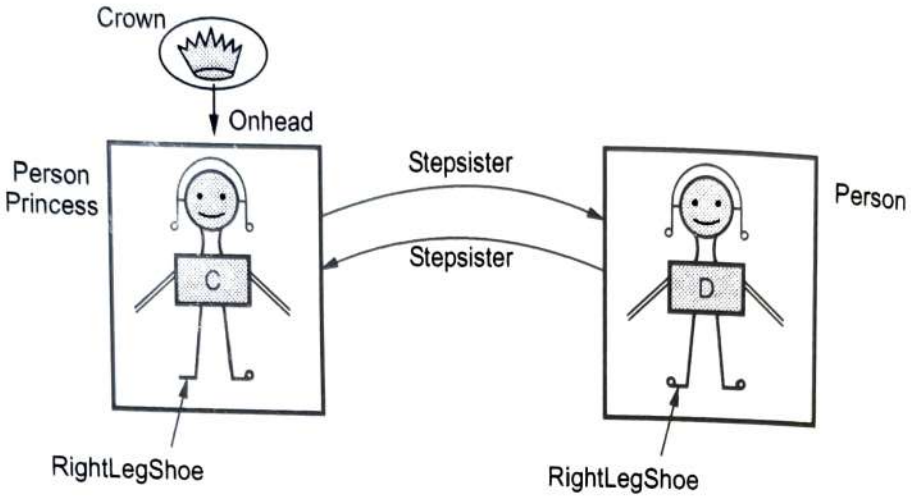


Fig. 6.1.1 Cindrella and Drizella are sisters

Objects :

1. Person princess Cindrella.
2. Person Drizella.
3. Crown.
4. Right Leg shoe of Cindrella.
5. Right Leg shoe of Drizella.

Relation :

1. "Onhead" < the Crown, Princess Cindrella >
2. "Stepsister" < Cindrella, Drizella >
3. "Person" < Cindrella >
4. Person < Drizella >
5. Princess < Cindrella >

Function :

[Every person wears shoe in right leg]

1. < Cindrella the Princess > → Rightlegshoe.
2. < Drizella > → Rightlegshoe

6.1.9 Sentence in First Order Logic

1) Terms

i. It is a logical expression that refer to an object or individuals.

ii. Terms are built using constant, variable, and function symbols.

iii. Constant symbols are terms.

iv. Functions are terms.

For example : LeftLegShoe (Cindrella), cat, X, times (2,3), blue, Mother (Ram), Seeta.

2) Atomic sentences

i. An atomic sentence is formed from a predicate symbol followed by a parenthesized list of terms.

For example : Stepsister (Cindrella, Drizella)

ii. Atomic sentences can have complex terms as the arguments.

For example : Married (Father (Cindrella), Mother(Drizella))

iii. Atomic sentences are also called atomic expressions, atoms or propositions.

For example : Equal (plus (two, three), five) is an atomic sentence.

3) Predicates

i. Predicates have a value of true or false.

ii. A predicate can take arguments, which are terms.

iii. A predicate with one argument expresses a property of an object.

For example : Student (Dipu)

iv. A predicate with two or more arguments expresses a relation between objects.

For example : Likes (Dipu, Books), Likes (Dipu, School-of (Dipu)).

v. A predicate with no arguments is a simple proposition, as in propositional logic.

4) Complex sentences

i. Atomic sentences can be connected to each other to form complex sentence.

Logical connectives, $\wedge$, $\vee$, $\neg$, $\rightarrow$ can be used to connect atomic sentences.

For example :

$\neg$ Princess (Drizella) $\rightarrow$ Princess (Cindrella)

ii. (foo (two, two, plus (two, three))) $\rightarrow$ (equal (plus (three, two), five) $\equiv$ true) is a sentence because all its components are sentences, appropriatly connected by logical operators.

- **Various sentences in first order logic formed using connectives :**
  1) If S is a sentence, then so is its negation, $\neg S$.
  2) If S_1, and S_2 are sentences, then so is their conjunction, $S_1 \wedge S_2$.
  3) If S_1 and S_2 are sentences, then so is their disjunction, $S_1 \vee S_2$.
  4) If S_1 and S_2 are sentences, then so is their implication, $S_1 \rightarrow S_2$.
  5) If S_1 and S_2 are sentences, then so is their equivalence, $S_1 \equiv S_2$.

6.1.10 Semantic of First Order Logic

Once constant symbols, relations and functions are decided, one need the interpretations to relate various symbols in first order logic. There are many ways to interpret the relationships as describe below : -

Interpretation

Let the domain D be a nonempty set.

An interpretation over D, is an assignment of the entities of D to each of the, constant, variable, predicate and function symbols of a predicate calculus expression such that :

1) Each constant is assigned an element of D.
2) Each variable is assigned to a non-empty subset of D; these are the allowable substitutions for that variable.
3) Each function 'f' of arity 'm' is defined on m arguments of D and defines a mapping from D^m into D.
4) Each predicate 'p' of arity 'n' is defined on 'n' arguments from D and defines a mapping from D^n into {T, F}.

- Given an interpretation, the meaning of an expression is a truth value assignment over the interpretation.

Truth value of first order logic expression

Assume an expression E and an interpretation I for E over a non-empty domain D. The truth value for E is determined by : -

1) The value of a constant is the element of D it is assigned to by I.
2) The value of a variable is the set of elements of D it is assigned to by I.
3) The value of a function expression is that element of D obtained by evaluating the function for the parameter values assigned by the interpretation.
4) The value of truth symbol "true" is T and "false" is F.
5) The value of an atomic sentence is either T or F, as determined by the interpretation I.

6) The value of the negation of a sentence is T if the value of the sentence is F and it is F if the value of the sentence is T.

7) The value of the conjunction of two sentences is T if the value of both sentences is T and it is F otherwise.

8) The truth value of expressions using $\vee$, $\rightarrow$, and $\equiv$ is determined from the value of their operands same as in propositional logic.

6.1.11 Quantifiers

They are used for expressing properties of entire collection of objects, rather than just a single object. We need complex sentences to represent huge amount of data. This will really help to represent real world data which has complex association among the objects.

There are two quantifiers in first order logic : -

1) **Universal Quantifier**
2) **Existential Quantifier**

6.1.11.1 Universal Quantifier (∀)

i) It is represented using symbol (∀)

∀ is pronounced as "for all".

ii) The sentence formed using universal quantifier use a variable.

- **Variable :**
  1. It is term itself.
  2. It is represented using lowercase alphabets like x, y, z.
  3. It can take up value from allowable objects, relation or function set.
  4. A term without variable is called as ground term.

iii) The sentence

∀ x P

Say that P is true for every object x, where P is a logical expression.

For example :

a) "All Princess are Person" can be represented as,

$\forall$ x Princess (x) $\Rightarrow$ Person (x)

b) "If the universe of discourse is people, then this means that everyone is happy" can be represented as,

$\forall$ x Happy (x)

iv) '∀' can make statement about every object.

v) **'⇒' is natural connective to use with '∀'.**

vi) **Common mistake to avoid :** Do not use ∧ as the main connective with ∀.

For example :

∀ x At (x, IIT) ∧ Smart (x)

It means that "Everyone is at IIT and everyone is smart."

It is mistake because you wanted to say "everyone at IIT is smart."

6.1.11.2 Existential Quantifier (∃)

i) These are used to make statement about some objects and not just for all.

ii) The objects are not named.

iii) The sentence, ∃ x P says that P is true for atleast one object x.

For example :

a) "Princess Cindrella has crown on her head" can be expressed as,

∃x Crown (x) ∧ OnHead (x, Cindrella)

[Crown is a crown and it is on Cindrella's head]

b) "If the universe of discourse is people, then this means there is at least one happy person", can be expressed as,

∃ x Happy (x)

iv) '∃ x' it is pronounced as **"There exists an x such that...."**

OR

"For some x....".

v) **'∧' is natural connective to be used with '∃'.**

vi) **Common mistake to avoid :** Do not use ⇒ as the main connective with ∃.

For example :

∃ x At (x, IIT) ⇒ Smart (x) is true if there is anyone who is not at IIT.

- **First order logic sentences using quantifier :**
  1) If X is a variable and S is a sentence, then ∀ X S is a sentence.
  2) If X is a variable and S is a sentence, then ∃ X S is a sentence.

6.1.11.3 Truth Values of First Order Logic Expressions Containing Quantifiers

For a variable X and a sentence S containing X :

1) The value of ∀ X S is True if S is True for all assignments to X under I, and it is False otherwise.

2) The value of ∃ X S is True if there is an assignment to X in the interpretation under which S is True; otherwise it is False.

6.1.11.4 The Nested Quantifiers

Using multiple (nested quantifier) we can express complex sentences. We can do nesting of same or different quantifiers.

- [Simple nesting using same quantifier ∀]
  For example : "Sisters are siblings" it can be written as,
  $\forall x\ \forall y$ Sisters $(x,y) \Rightarrow$ Siblings (x,y).
- [Consecutive quantifier of the same type can be written as one quantifier with several variables]
  For example : "Siblinghood" is symmetric relationship then we have,
  $\forall x,y$ Sibling $(x,y) \Leftrightarrow$ Sibling (y,x).
- [Mixing of two different quantifiers]
  For example : -
  a) Everybody loves somebody.
  $\forall x\ \exists y$ Loves (x,y)
  b) There is someone who is loved by everybody.
  $\exists x\ \forall y$ Loves (y,x)

Note ***The order of quantifier is important.*** *Change in the order effectively changes the meaning of predicate.*

In above example :
$\forall y\ \exists x$ Loves (y,x)

Means that, for every y there is somebody to Love (!!!) (?)

6.1.11.5 Relationship between ∀ and ∃

Two quantifiers are connected with each other through "negation",

Saying : "All girls like Rose" means that

"There is no girl who does not like Rose".

$\forall x$ Like (x, Rose) is equivalent to

$\neg \exists x \neg$ Like (x, Rose).

- **DeMorgan's rule for quantifiers**

1) $\forall x \neg P \equiv \neg \exists x\ P$
2) $\neg P \wedge \neg Q \equiv \neg (P \vee Q)$
3) $\neg \forall x\ P \equiv \exists x \neg P$
4) $\neg (P \wedge Q) \equiv \neg P \vee \neg Q$
5) $\forall x\ P \equiv \neg \exists x \neg P$
6) $P \wedge Q \equiv \neg (\neg P \vee \neg Q)$

7) $\exists x\ P \equiv \neg\ \forall x \neg\ P$

8) $P \vee Q \equiv \neg\ (\neg P \wedge \neg\ Q)$

6.1.11.6 Equality

1) In first order logic equality symbol is used to make sentences that can associate two more atomic sentences.
2) It can be used to state fact about given function.
3) It can be used with negation to express that two terms are not the same object.

For example : Stepsister (Cindrella) = X

To say "Cindrella has at least 2 step-sisters"

we can write $\exists$ x,y Stepsister (x, Cinderella) $\wedge$ Stepsister (y, Cindrella) $\wedge \neg$ (x=y)

6.1.11.7 Properties of Quantifiers

1) Quantifiers of same type commute

i) $\forall$ x $\forall$ y is same as $\forall$ y $\forall$ x

ii) $\exists$ x $\exists$ y is same as $\exists$ y $\exists$ x.

For example :

a) Statement 1) $\forall x \forall y$ father (x,y) $\rightarrow$ Parent (x,y)

Statement 2) $\forall y \forall x$ father (x,y) $\rightarrow$ Parent (x,y)

Note that, Both statement 1 and statement 2 are same.

b) Statement 1) $\exists x \exists y$ Likes (x,y) $\rightarrow$ FriendOF (x,y)

Statement 2) $\exists y \exists x$ Likes (x,y) $\rightarrow$ FriendOF (x,y)

Note that, Both statement 1 and statement 2 are same.

2) Quantifiers of different type do not commute

$\exists x \forall y$ **is not the same as** $\forall y \exists x$

For example :

a) Statement 1 :

$\exists x \forall y$ Loves (x,y)

"There is a person who loves everyone in the world".

Statement 2 :

$\forall y \exists x$ Loves (x,y)

"Everyone in the world is loved by at least one person".

Note that,

Statement 1 and Statement 2 are not same.

b) Statement 1 :

$\forall x \exists y$ Mother (x,y)

"Everyone has a mother" (correct)

Statement 2 :

$\exists y \forall x$ Mother (x,y)

"There is a person who is the mother of everyone". (Wrong)

3) Quantifier Duality

$\forall x$ Likes (x, IceCream) is the same as

$\neg \exists x \neg$ Likes (x, IceCream).

$\exists x$ Likes (x, Brocoli) is the same as

$\neg \forall x \neg$ Likes (x, Brocoli)

6.1.11.8 The Examples Bank

Here are few varieties of first order logic sentences : -

1) $\underbrace{\underbrace{\text{StepSister}}_{\text{Predicate}} \; \underbrace{(\underbrace{\text{Drizella}}_{\text{Constant}}}_{\text{Term}}, \underbrace{\underbrace{\text{Cindrella}}_{\text{Constant}})}_{\text{Term}}}_{\text{Atomic sentence}}$

2) $\underbrace{\underbrace{>}_{\text{Predicate}} \; [(\underbrace{\underbrace{\text{Length}}_{\text{Function}} \; (\underbrace{\underbrace{\text{Leftshoeof}}_{\text{FunctionOF}} \; \underbrace{(\text{Cindrella})}_{\text{Constant}}}_{\text{Term}}}_{\text{Term}}, \text{Length(Leftshoeof (Drizella)))}]}_{\text{Atomic sentence}}$

3) $\underbrace{\underbrace{\underbrace{\text{Sibling}}_{\text{Predicate}} \; (\underbrace{\text{Cindrella}}_{\text{Term}}, \underbrace{\text{Drizella}}_{\text{Term}})}_{\text{Atomic sentence}} \Rightarrow \underbrace{\text{Sibling(Drizella, Cindrella)}}_{\text{Atomic sentence}}}_{\text{Complex sentence}}$

4) Everyone studying in India is smart.

$\underbrace{\forall x}_{\text{Variable}} \; \underbrace{(\text{StudiesIn } (x, \text{India}) \Rightarrow \text{Smart } (x))}_{\text{Sentences}}$

5) Someone studying in India is smart.

$\exists x$ (StudiesIn (x, India) $\wedge$ Smart (x))

6) "Ram has an umbrella"can be written as,

$\exists y$ (Has (Ram, y) $\wedge$ IsUmbrella (y))

7) "Anything that has an umbrella is not wet".

$\forall x [(\exists y (Has (x,y) \wedge IsUmbrella (y))) \Rightarrow \neg (IsWet (x))]$

8) "Any person who has an umbrella is not wet".

$\forall x [IsPerson (x) \Rightarrow ((\exists y (Has (x,y) \wedge IsUmbrella (y))) \Rightarrow \neg (IsWet (x)))]$

9) Ram has at least two umbrellas.

$\exists x [\exists y (Has (Ram, x) \wedge IsUmbrella (x) \wedge Has (Ram, y) \wedge IsUmbrella (y) \wedge \neg (x=y)]$

10) Everyone Has umbrella.

$\forall x \exists y [Has (x,y) \wedge IsUmbrella (y)]$

11) If it doesn't rain on Monday, Nimu will go to the school.

$\neg$ Weather (rain, Monday) $\rightarrow$ Go (Nimu, School)

12) Sweety is a Doberman pinscher and a good dog.

GoodDog (Sweety) $\wedge$ Isa (Sweety, Doberman)

13) All basketball players are tall.

$\forall x$ (Basketball_Player (x) $\rightarrow$ Tall (x))

14) Some people like cricket.

$\exists x$ (Person (x) $\wedge$ Likes (x, cricket))

15) If wishes were horses, beggars would ride.

Equal (wishes, horses) $\rightarrow$ Ride (Beggars)

16) Nobody likes taxes.

$\neg \exists$ x Likes (x, taxes)

17) Sisters are siblings.

$\forall$ x, y [Sisters (x,y) $\Rightarrow$ Siblings (x,y)]

Note *Sibling is symmetric relationship.*

18) Siblings is symmetric.

$\forall$ x, y (Sibling (x, y) $\Leftrightarrow$ Sibling (y, x)]

19) One's mother is one's female parent.

$\forall$ x, y (Mother (x, y) $\Leftrightarrow$ (Female (x) $\wedge$ Parent (x, y)))

20) A first cousine is a child of a parent's sibling.

$\forall$ x, y [FirstCousin (x, y) $\leftrightarrow \exists$ p,

ps (Parent (p,x) $\wedge$ Sibling (ps, p) $\wedge$ Parent (ps, y))]

First order logic semantic example

"A Blocks World".

1. We can model a blocks world, for a control algorithm written for a robotic arm.
2. The diagramatic representation is as shown below :

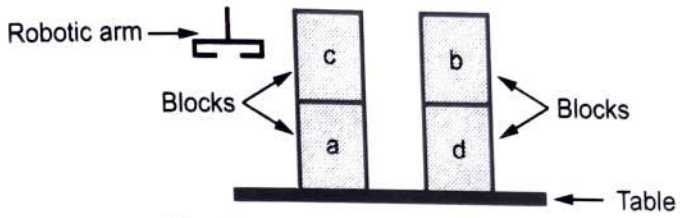


Fig. 6.1.2 The blocks world

First order logic expressions for above situation of block world :

1) On (c, a)
2) On (b, d)
3) On Table (a)
4) On Table (d)
5) Clear (c) [Clear says that there is no block above argument block]
6) Clear (b)
7) hand_empty.

6.1.12 Using First Order Logic for Knowledge Base

1) Sentences are added to knowledge base using **TELL** interface.

   Such sentences are called **"assertions"**.

   For example : TELL (KB, Princess (Cindrella))

2) i) When we want to query knowledgebase then ASK interface is used.
   ii) Questions asked to knowledgebase are called as queries or goals.

   For example : ASK (KB, Person (Cindrella)) will return true.

   iii) As an answer to a query knowledge base can return true or false.
   iv) We can use quantifiers in query. But one need to **bind** or **substitute** variables in sentence so as to get fruitful output.

   For example : ASK (KB, $\exists$x Person (x))

   It will return true but **"Who is Person" is more informative** than just that **"Somebody is person"**.

6.1.13 The Kinship Domain

This is a type of relationship which represent real life family relationships like FatherOF, MotherOF, SonOF, etc.

For example :

Rajiv is SonOF Indira.

Rajiv's grandfather is FatherOF Indira.

In Kinship domain relationships -

i) Objects - are people.

ii) We can have unary predicates like Male, Female.

iii) Relations are parenthood, Marriage.

These are represented using binary predicates like Parent, Siblings, Aunt, Uncle, Grandfathers, etc.

For example :

1) One's wife is one's female spouse $\forall$ w, h wife (w, h) $\Leftrightarrow$ (female (w) $\wedge$ Spouse (w,h))

2) Male and female are disjoint categories $\forall$ x Male (x) $\Leftrightarrow \neg$ Female (x)

3) Parent and child is inverse relation $\forall$ p, c Parent (p, c) $\Leftrightarrow$ Child (c, p)

iv) Sentences in Kinship domain are called as axioms (rules) or definitions.

v) Some axioms can be theorem i.e. they are derived from other axioms

$\forall$ x, y Sibling (x, y) $\Leftrightarrow$ Sibling (y,x).

6.1.14 Number, Sets and Lists

Number, sets, list are the means with which a knowledge base can be easy to build, because numbers, sets, list have symbols, predicates and functions that can be utilized for representing knowledge; A brief idea is given in following section :

- **Numbers :**

Using numbers one can represent large theory statements, which can be useful for AI agent.

For example : To describe natural number theory

1) We need predicate NatNum. It will be true for all natural numbers.

2) We need constant symbol 0 (zero)

3) We need successor function symbol S.

4) There is axiom called as Peano Axiom that define natural number and addition.

5) We can define any natural number using following predicates,

NatNum (0)

$\forall$ n NatNum (n) $\Rightarrow$ NatNum (S(n))

i.e. we can derive from O any natural number using successor function i.e three can be derived from O as,

$$S(\underbrace{S(\underbrace{S(O)}_{1})}_{2})}_{3}$$

6) We can have constraints on function like

i) $\forall$ n O $\neq$ S(n)

i.e. O can not be a successor of any 'n' as 'O' is first natural number.

ii) $\forall$ m, n $\quad m \neq n \Rightarrow S(m) \neq S(n)$

7) We can define addition as

$\forall$ m NatNum (m) $\Rightarrow$ + (m, 0) = m

[It means that adding 0 to any natural number gives that number].

$\forall$ m, n NatNum (m) $\wedge$ NatNum (n) $\Rightarrow$ + (S(m), n) = S (+(m, n))

8) Ordinary mathematics have **infix** notations for expression like m+n (operator is between operands)

9) In first order logic we use notation +mn which is called **prefix** (where operator comes first).

- **Sets :**

Set is collection of unordered distinct elements. We can use number theory to represent set theory. Set theory is also useful for AI agent, which can be used for storing data.

For example :

i) We can use unary predicate set which is true for set.

ii) Binary predicate

a) $x \in S$ (x is member of set S)

b) $S_1 \subseteq S_2$ (S_1 is subset of S_2)

Lists :

List is a collection of ordered elements and a element can appear repetedly in the list.

[Lisp is a List programming language which makes use of lists and predicates and function on those list]

For example :

- Nil is constant list with no elements.
- Append - Add element at last in list.
- First - Return first element of list are functions.
- Find is a predicate which search element in list.

6.1.15 Synchronic and Diachronic Sentences

1) The sentences dealing with time are synchronic sentences. They relate properties of a world state to other properties of the same world state.
2) The sentences that allow reasoning "across time" are called as diachronic sentences. That is previous state combined with current action should provide reasoning to determine its current location.

6.1.16 Synchronic Rules for Inferencing

6.1.16.1 Diagnostic Rules

1) From observed effect they generate hidden causes.
2) They help to deduce hidden facts in the world.

For example : Consider Wumpus world.

Representing Wumpus world environment

1) Various objects in Wumpus world would be squares, pits, the Wumpus, etc.
2) Each square should be named. The fact about adjacent squares should be specified.
3) Pit can be represented using unary predicate, that is true for squares containing pit.
4) A single Wumpus in wumpus world is represented using constant Wumpus. As Wumpus lives in one square its location can be described using functions like Home (Wumpus) etc.
5) By having knowledge about places and their properties agent can infer where is pit and where is Wumpus.
6) The agent will keep on changing its location over time which can be described using function like,

   At (Agent, s, t) means that agent is square 's' at time 't'.

Diagnostic rule for finding Pit is,

A) "If square is breezy some adjacent square must contain pit", which is written as,

$\forall$ s Breezy (s) $\Rightarrow \exists$ r Adjacent (r, s) $\wedge$ Pit (r).

B) "If square is not breezy, no adjacent square contains pit", which is written as,

$\forall$ s $\neg$ Breezy (s) $\Rightarrow \neg \exists$ r Adjacent (r, s) $\wedge$ Pit (r)

C) Combining A and B clause we get biconditional sentence,

$\forall$ s Breezy (s) $\Leftrightarrow \exists$ r Adjacent (r, s) $\wedge$ Pit (r)

6.1.16.2 Causal Rules

1) These rules reflect the assumed direction of causality in the world.
2) Some hidden property (fact) of the world causes certain percepts to be generated. For example : A pit causes all adjacent squares to be breezy.
   A. Which is written as

      $\forall$ r Pit (r) $\Rightarrow$ [$\forall$ s Adjacent (r, s) $\Rightarrow$ Breezy (s)].

   B. If all squares adjacent to a given square are pitless, the square will not be breezy, which is written as,

      $\forall$ s [$\forall$ r Adjacent (r, s) $\Rightarrow \neg$ Pit (r)] $\Rightarrow \neg$ Breezy (s).

   C. A and B can be combined together to form biconditional sentence. Biconditional sentences are causal in nature because from world state they can generate facts.

6.1.16.3 Model Based Reasoning Systems and Diagnostic Reasoning Systems

1) Systems that reason with causal rules are called model-based reasoning systems.

   For example : In Medical Systems where disease needs to be investigated one can start from diseases process to reach at the conclusion about infected disease. Symptoms are used as facts about current state rather than the direct reason for inference.

2) Systems that directly makes association between facts and conclusion are diagnostic reasoning systems.

   For example : In finding infection of the diseases a direct association is made between the symptoms and conclusion about disease.

3) The rule of thumb is -

   Agent designer should concentrate on getting knowledge right without worrying about inferencing procedure.

   More correct knowledge leads to better inferencing.

Any complete logical inferencing algorithm will infer the strongest possible description of the world state provided that,

i) It receives all possible available percepts.

ii) If the axioms are correctly and completely described exactly the way world works and the way the percepts are produced in the world.

6.2 The Wumpus World Represented using First Order Logic

6.2.1 Wumpus World Revisited

1) Agent receives percept with five elements shown as follows : -

   Percept ([Stench, Breeze, Glitter, None, None] , 5)

   Here percept is binary predicate and stench etc. are constants stored in the list.

2) The knowledge base must contain both the percept and the time at which it occured. We will use integer for time steps.

3) The actions in Wumpus world can be represented using logical terms.

   Turn (Right), Turn (Left), Forward, Shoot, Grab, Release, Climb.

   To choose best action from action list the query can be constructed as,

   $\exists$ a BestAction (a, 5).

6.2.2 Reasoning in Wumpus World using First Order Logic

1) We need percept data for reasoning.

2) Raw percept data implies certain facts about the current state.

   For example :

   $\forall$ t, s, g, m, c Percept ([s, Breeze, g, m, c], t) $\Rightarrow$ Breeze (t)

   $\forall$ t, s, b, m, c Percept ([s, b, Glitter, m, c], t) $\Rightarrow$ Glitter (t).

3) These rules exhibit **a trivial form of the reasoning process** called **perception.**

4) Simple reflex behavior can also be implemented using quantified implication sentences.

   For example : $\forall$ t Glitter (t) $\Rightarrow$ BestAction (Grab, t)

5) Given all above axioms the 'BestAction' query would return the best action as 'Grab'.

6.3 Knowledge Engineering in First Order Logic

The process of constructing knowledge base is called as knowledge base engineering. These knowledge base engineering projects vary widely in content, scope and difficulty. A specially designated person called as knowledge engineer will do task of knowledge engineering. He is a person who,

1) Investigates domain.
2) Understands which concepts are important in that domain.
3) Creates format for representing objects and relations.

6.3.1 We have Two Types of Knowledge Base

1) General purpose knowledge base : Which is intended to support queries about full range of human knowledge. In this, we can expect any kind of query which knowledge base will have to infer.
2) Special purpose knowledge base : Which has restricted domain [which can be a certain problem specific domain].

   Here expected queries are known in advance.

6.3.2 Steps in Knowledgebase Engineering Process

1) Identify the task :

i. The knowledgebase engineer must find the range of questions that knowledgebase will support. He should also find the facts that would be available for each specific problem instance.

ii. Once the task is identified it will determine what knowledge must be represented in order to relate problem instances to answers.

iii. Note that identifying the task is similar to PEAS designing.

iv. For example : In Wumpus world knowledgebase engineering process, knowledge engineering should decide ⇒ Whether knowledgebase need to be able to choose actions or it is required to answer questions only about the contents of the environment ?

2) Assemble the relevant knowledge :

i. This is the process of knowledge acquisition.

ii. At this stage knowledge is gathered.

iii. This step helps to understand how the domain actually works.

iv. This process is carried out by an expert in the underlying domain. (either knowledge base engineer himself or knowledgebase engineer can take expert's help).

v. For example : In Wumpus world, the set of rules, the related actions are understood in this phase of knowledgebase engineering process.

vi. In real world domain it is difficult to study and gather knowledge as environment is complex.

For example : In VLSI design one needs to consider stray capacitances and skin effects.

3) Decide vocabulary of predicates, functions, and constants :

i. This step move towards formal representation of knowledge assembled in step 2.

ii. Domain-level concepts are translated into logic level names. The process of translation will involve lots of questions about domain like, what is this ?, What it likes ?, What it do ?, How it changes states etc.

iii. The translated version of problem domain knowledge will contain predicates, functions, constants, terms. All these things together form the vocabulary of problem-domain.

iv. Vocabulary of problem-domain is called as **Ontology**. The ontology determines what kinds of things exists, but does not determine specific properties and relationship about them.

4) Encode knowledge about the domain :

i. In this step knowledge engineer formally list down the axioms for all the vocabulary terms. It is noting of the meaning of the terms, enabling the expert to check the content.

ii. If some terms are found missing or irrelevant then the mistakes are corrected using step 3 again.

iii. If additional terms are found they are added to ontology. Invalid terms are removed from ontology. Thus the ontology is updated.

5) Encode a description of the specific problem instance : -

i. If ontology is well defined encoding the description becomes easy.

ii. This step involves writing simple atomic sentences about instances of various concepts that are part of ontology.

For example : For a logical agent various problem instances can be supplied by its sensors.

For disembodied knowledge base the instances can be generated by expected input.

6) Fire queries to inference procedure and get outputs : -

i. In this step the trials (demos) of querying the knowledge base are conducted.

ii. The inference procedure is applied on axioms and problem specific facts to get the results.

7) Debug the knowledgebase :

i. If step 6 succeds, its the reward of the first 5 steps.

ii. But if one is getting unexpected results of queries then knowledge base is debugged for errors.

iii. There are various reasons why inferencing can fail. The axioms can be missing, axioms can be weak to infer related query, axioms is wrong, are some of the reasons for failure.

For example :

The axiom $\forall$ x Numoflegs (x, 4) $\Rightarrow$ Mammal (x)

The above statement is false for reptiles. Here some other necessary axioms are missing.

iv. Step 6 and 7 will be carried out repeatedly to make 100 % error free knowledge base.

6.3.3 Knowledge Engineering in First Order Logic for the Electronic Circuits Domain

• Electronic circuit domain example - One-bit full adder

1. Identify the task : Does the circuit actually add properly ? (circuit verification)
2. Assemble the relevant knowledge :

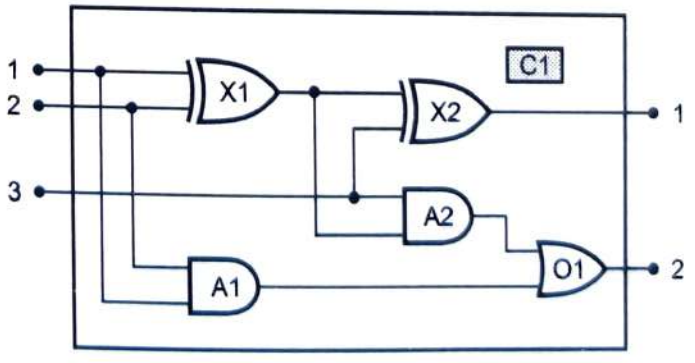


Fig. 6.3.1 A digital circuit C1 designed for one bit full adder

It is composed of wires and gates, types of gates (AND, OR, XOR, NOT). Irrelevant - Size, Shape, Color, Cost of gates.

3. Decide on a vocabulary / encoding :

   Alternatives -

   Type (X_1) = XOR

   Type $(X_1$, XOR)

   XOR (X_1)

   ....

4. Encode general knowledge of the domain :

   $\forall\ t_1\ ,\ t_2$ Connected $(t_1, t_2) \Rightarrow$ Signal (t_1) = Signal (t_2)

   $\forall$ t Signal (t) = 1 $\vee$ Signal (t) = 0 $1 \neq 0$

   $\forall\ t_1, t_2$ Connected $(t_1, t_2) \Rightarrow$ Connected (t_2, t_1)

   $\forall$ g Type (g) = OR $\Rightarrow$ (Signal (out (1, g)) = 1 $\Leftrightarrow \exists$ n Signal (In (n, g)) = 1)

   $\forall$ g Type (g) = AND $\Rightarrow$ (Signal (out (1, g)) = 0 $\Leftrightarrow \exists$n Signal (In (n, g)) = 0)

   $\forall$ g Type (g) = XOR $\Rightarrow$ (Signal (out (1, g)) =1 $\Leftrightarrow$ Signal (In (1, g)) $\neq$ Signal (In(2, g))

   $\forall$ g Type (g) = NOT $\Rightarrow$ Signal (out (1, g)) $\neq$ Signal (In (1, g)).

5. Encode the specific problem instance :

   Type (X_1) = XOR Type (X_2) = XOR

   Type (A_1) = AND Type (A_2) = AND

   Type (O_1) = OR.

   connected (Out (1, X_1), In (1, X_2))

   connected (In (1, C_1), In (1, X_1))

   connected (Out (1, X_1), In (2, A_2))

   connected (In (1, C_1), In (1, A_1))

   connected (Out (1, A_2), In (1, O_1))

   connected (In (2, C_1), In (2, X_1))

   connected (Out (1, A_1), In (2, O_1)

   connected (In (2, C_1), In (2, A_1))

   connected (Out (1, X_2), Out (1, C_1))

   connected (In (3, C_1), In (2, X_2))

   connected (Out (1, O_1), Out (2, C_1))

   connected (In (3, C_1), In (1, A_2))

6. Pose the queries to the inference procedure.

   What are the possible sets of values of all the terminals for the adder circuit ?

$\exists i_1, i_2, i_3, o_1, o_2$ Signal (In (1, C – 1)) =

$i_1 \wedge$ Signal (In (2, C_1)) = $i_2 \wedge$ Signal (In (3, C_1)) =

$i_3 \wedge$ Signal (Out (1, c_1) = $o_1 \wedge$ Signal (Out (2, C_1) = o_2

7. Debug the knowledge base -
   May be you have omitted assertions like $1 \neq 0$, etc. ?

6.4 Inference in First Order Logic

GTU : Summer-18, Winter-18

We have seen how to make sound and complete inference for Propositional logic. These results can be extended to obtain algorithms that can answer any question (if it has answer) stated in first order logic.

6.4.1 Inferencing in First Order Logic

The semantics of first order logic provides a basis for a formal theory of logical inferences. These new derived inferences are correct in the sense that they are consistent with all previous interpretations of the original set of expressions.

The inference rules provide a computationally feasible way to determine when an expression, a component of an interpretation, logically follows for that interpretation.

The concept "**logically follows**" provides basis for proofs of the soundness and correctness of inference rules.

An interpretation that makes a sentence true is said to satisfy that sentence. An interpretation that satisfies every member of a set of expressions is said to satisfy the set.

An expression X logically follows from a set of first order logic expressions S if every interpretation that satisfies S also satisfies X. The function of logical inference is to produce new sentence that logically follow a given set of first order logic sentences.

The inference rule is essentially a mechanical means of producing new first order logic sentence from other sentences.

When every sentence X produced by an inference rule operating on a set S of logical expressions, logically follows from S, the inference rule is said to be **sound**.

If the inference rule is able to produce every expression that logically follows from S, then it is said to be **complete**.

6.4.2 Definition of Various Terms used in Inference Theory

1. **Satisfy, Model** :

- For a first order logic expression X and an interpretation I,
  - If X has a value of T under I and a particular variable assignment, then I is said to satisfy X.
  - If I satisfies X for all variable assignments, then I is a model of X.

2. **Inconsistent** :

- For a first order logic expression X and an interpretation I,
  - X is satisfiable if and only if there exist an interpretation and variable assignment that satisfy it; otherwise it is unsatisfiable.
  - A set of expressions is satisfiable if and only if there exist an interpretation and variable assignment that satisfy every element.

  If a set of expressions is not satisfiable, it is said to be inconsistent.

3. **Valid** :

- For a first order logic expression X and an interpretation I, If X has a value T for all possible interpretations, X is said to be valid.

4. **Proof Procedure** :

- A proof procedure is combination of an inference rule and an algorithm for applying that rule to a set of logical expressions to generate new sentences.

6.4.3 Inference Rules

1) **Modus ponens** :

If the sentence **P** and **P -> Q** are known to be true, then modus ponens lets us infer **Q**.

For example : If we have statment, " If it is raining then the ground will be wet" and "It is raining". If P denotes " It is raining" and Q is "The ground is wet" then the first expression becomes $P \rightarrow Q$. Because it is indeed now raining (P is true), our set of axioms becomes,

$\{ P \rightarrow Q$

$P \}$

Through an application of modus ponens, the fact that "The ground is wet" (Q) may be added to the set of true expressions.

- **The generalized modus ponens :**

For atomic sentences P_i, P_i', and q, where there is a substitution Q such that SUBST $(\theta, P_i') =$ SUBST (θ, P_i'),

For all i,

$$\frac{P_1', P_2', \ldots . P_n', (P_1 \wedge P_2 \wedge \ldots . \wedge P_n \Rightarrow q)}{\text{SUBST}(\theta, q)}$$

There are n+1 premises to this rule : - The 'n' atomic sentences p_i' and the one implication. The conclusion is the result applying the substitution θ to the consequent q.

P_1' is - Princess (Cindrella) P_1 is - Princess (x)

P_2' is - Kind (y) P_2 is - Kind (x)

θ is {x/Cindrella, y/cindrella} q is GoodHearted (x)

SUBST (θ, q) is, GoodHearted (cinderella).

2) Modus tollens :

Under the inference rules modus tollens, if $\mathbf{P \rightarrow Q}$ is known to be true and **Q** is known to be false, we can infer $\neg$ **P**.

For example : Given that,

i) $\underbrace{\text{Engine Starts} \wedge \neg \text{ Flat Tire}}_{P} \Rightarrow \underbrace{\text{Car OK}}_{Q}$

ii) $\underbrace{\neg \text{ Car OK}}_{Q}$

Then by Modus tollens,

$\underbrace{\neg \text{ (Engine Starts} \wedge \neg \text{ Flat Tire)}}_{\neg P}$ which is equivalence to $\neg$ [Engine Starts $\vee$ FlatTire]

3) And elimination :

And elimination allows us to infer the truth of either of the conjuncts from the truth of a conjunctive sentence. For instance, $\mathbf{P \wedge Q}$ is true. Lets us conclude that **P** and **Q** are true.

4) And introduction :

And introduction lets us infer the truth of a conjunction from the truth of its conjuncts. For instance, if **P** and **Q** are true, then $\mathbf{P \wedge Q}$ is true.

5) Inference rules for quantifiers :

1. Universal Instantiation (UI) :

This rule states that, we can infer any sentence obtained by substituting a ground term (a term without variable) for the variable.

Formally, this rule is described with the concept of substitution.

Substitution :

SUBST (θ, S) denotes the result of applying the substitution θ to the sentence S.

With the substitution we can write UI rule as follows,

$$\frac{\forall \; v \; S}{\text{SUBST}(\{v / g\}, S)}$$

For any variable V, ground term g.

Note *UI can be applied many times to produce many different consequences.*

For example :

All beautiful princess are GoodHearted.

$\forall$ x Princess (x) $\wedge$ Beautiful (x) $\Rightarrow$ GoodHearted (x)

From above axioms we can infer following permissible sentences : -

i. Princess (Seeta) $\wedge$ Beautiful (Seeta) $\Rightarrow$ GoodHearted (Seeta)

ii. Princess (Indumati) $\wedge$ Beautiful (Indumati) $\Rightarrow$ GoodHearted (Indumati)

iii. Princess (Mother (Seeta)) $\wedge$ Beautiful (Mother (Seeta)) $\Rightarrow$ Good Hearted (Mother (Seeta))

2. Existential Instantiation (EI)

This rule states that, "For any sentence S, variable V and constant symbol k that does not appear else where in the KB, following statement holds,

$$\frac{\exists\ V, S}{SUBST(\{V\ /\ k\}, S)}$$

Basically the existential statement say that, there is some object satisfying a condition. The instantiation process is just giving a name to this object. [It should be noted that this name must not already belong to another object].

For example :

$\exists$ x Crown (x) $\wedge$ OnHead (x, Cindrella)

We can infer sentence,

Crown (H) $\wedge$ On Head (H, Cindrella) as long as H does not appear else where in the knowledge base.

Note *Existential Instatiation can be applied once and then existentially quantified sentence can be discarded.*

For example :

If we add sentence,

Arrested (Police, Thief) then we do not need,

$\exists$ x Arrested (x, Thief)

6.4.4 Reducing First Order Logic Inferences to Propositional Inferences

1) In first order logic, we can use rules of quantified sentences to infer non-quantified sentences.
2) Using this rules we can reduce first order logic inferences to Propositional logic inferences.

3) The technique : -

- Existential quantified sentence can be replaced by one instantiations. (i.e one non-quantified sentence).
- Universal instantiation sentence can be replaced by all possible instantiations.

  For example :

  $\forall$ x Princess (x) $\wedge$ Beautiful (x) $\Rightarrow$ GoodHearted (x).

  We can apply universal instantiation to this sentence, using all ground terms in knowledgebase.

For example :

We can have ground terms,

Princess (Cindrella)

Beautiful (Cindrella),

then from universal instantiation we get,

GoodHearted (Cindrella)

We can add the sentence to knowledge base and discard universal quantified sentence. This technique is called as Propositionalization. Once we get all terms in Propositional logic, we can apply all algorithms of Propositional logic to these sentences.

4) Every first order logic knowledge base and first order logic query can be Propositionalized in such a way that entailment is preserved. It is a complete decision procedure.

5) Critical Problem : When a knowledge base contains functional symbol a set of possible ground term substitution is infinite.

   For example :

   'Mother' symbol can be infinitely nested like,

   Mother (Mother (Mother (Seeta)))

6) For first order logic, if we are deriving inference, the algorithm for inferring is surely going to output true for every entailed sentence. But there is no algorithm which outputs false to every non-entailed sentence. This is termed as **semidecidable property** of first order logic for entailment.

6.4.5 Concept of Lifting

1) Lifting means giving additional facilities (that is generating new modified version with upgradation).
2) Generalized Modus Ponen is lifted version of Modus Ponen (A rule in Propositional logic).

3) Lifting of the rule is done from Propositional logic version to first order logic version.

4) Similar to Modus ponen we are going to study lifted upgrated versions [from Propositional logic to first order logic] of unification, forward chaining, Backward Chaining and resolution procedures.

6.4.6 Unification

1) It is the process of finding substitutions for lifted inference rules, which can make different logical expression to look similar (identical).

2) Unification is an procedure for determining substitutions needed to make two first order logic expressions match.

3) Unification is important component of all first order logic inference algorithms.

4) The unification algorithm takes two sentences and returns a unifier for them, if one exists.

The algorithm statement is as follows : -

Unify (p, q) = θ where,

SUBST (θ, p) = SUBST (θ, q).

For example :

Suppose we have query-whom does Manmohan meets ? i.e first order logic query is Meets (Manmohan, x)

Some answers to this query can be found by searching all the sentences in knowledge base that unify with Meets (Manmohan, x).

Following are results with 4 different sentences that might be in knowledge base.

i. Unify (Meets (Manmohan, x), Meets (Manmohan, Sonia)) = { x/Sonia}

ii. Unify (Meets (Manmohan, x), Meets (y, Sharad)) = { x/Sharad, y/Manmohan}

iii. Unify (Meets (Manmohan, x), Meets (y/Friend (y))) = { y/Manmohan, x/Friend (Manmohan)}

iv. Unify (Meet (Manmohan, x), Meets (x, Mayawati)) = Fail

The last unification failed.

Note that, Meets (x, Mayawati) means everybody meet Mayawati. It was very straight forward that Manmohan meets Mayawati. But problem arised due to clash between variable name x, which appeared in both sentences.

Solution to variable name-clash :

We can rename variable in one of the sentences where it is clashed. This is called as **Standardizing apart** one of the two sentences.

For example :

In our Meets sentence,

Meets (x, Mayawati) we rename x as z.

∴Unify (Meets (Manmohan, x), Meets (z, Mayawati)) = { x/Mayawati, z/Manmohan}

Algorithm Unify

```
Function unify (E1, E2) ;
begin
Case
both E1 and E2 are constants or the empty list :   // Recursion stops
    If E1 = E2 then return {}
    else return FAIL ;
E1 is a variable :
    If E1 occurs in E2 then return FAIL
    else return {E2/E1} ;
E2 is a variable :
    if E2 occurs in E1 then return FAIL
    else return {E1/E2}
either E1 or E2 are empty then return FAIL // the lists are of different sizes
 Otherwise : // both E1 and E2 are lists.
    begin
        HE1 : = First element of E1 ;
        HE2 : = First element of E2 ;
        SUBS1 : = Unify (HE1, HE2) ;
        if SUBS1 : = FAIL then return FAIL ;
        TE1 : = apply (SUB1, rest of E1) ;
        TE2 : = apply (SUB1, rest of E2) ;
        SUBS2 : = Unify (TE1, TE2) ;
        if SUBS2 = FAIL then return FAIL ;
        else return composition (SUBS1, SUBS2)
     end
    end                    // end case
   end
```

- **Most General Unifier : (MGU) :**

1) We are finding substitution that makes two arguments look the same.
2) There can be more than one substitutions that can make two arguments look the same.

   In our example of Meets sentence

   Unify (Meets (Manmohan, x), Meets (y,z))

   will return

   { y/Manmohan x/z}

   OR { y/Manmohan, x/Manmohan, z/Manmohan}
3) From above example we can conclude that

   First unifier gives Meets (Manmohan, z)

   Second unifier gives Meets (Manmohan, Manmohan)

   The second unifier is obtained from first by an additional substitution,

   { z/Manmohan}

   Here, we say that first unifier is more general than the second because it places fewer restrictions on the values of the variables.
4) For every such unifiable pair of expressions, there is a single most general unifier which is unique upto renaming of variables.

   In our example { y/Manmohan, x/z}
5) General steps to find Most General Unifier (MGU) and problems associated in finding MGU.

   i) The algorithm recursively explore the two expressions simultaneously "side by side", building up a unifier along the way. It fails at a stage when two corresponding points in the structure do not match.

   ii) The algorithm has one expensive step, (in terms of multiple checks). When matching a variable for a complex term, one must check whether the variable itself occurs inside the term. If it does occurs then the match fails because no consistent unifier can be constructed. This **"Occur check"** make algorithm expensive in terms of time. The time complexity becomes equivalent to quadratic in the size of the expressions being unified.

   iii) Because of huge time complexity some systems drops the **"Occur check"**. But these systems then can become unsound inference systems. Therefore many systems use different procedures for finding MGU, than the one we have discussed here.

Unification example :

A Tree diagram representing unification algorithm procedure :

- Initial steps in the unification of (Parents X (Father X) (Mother bill)) and (Parents bill (Father bill) Y).

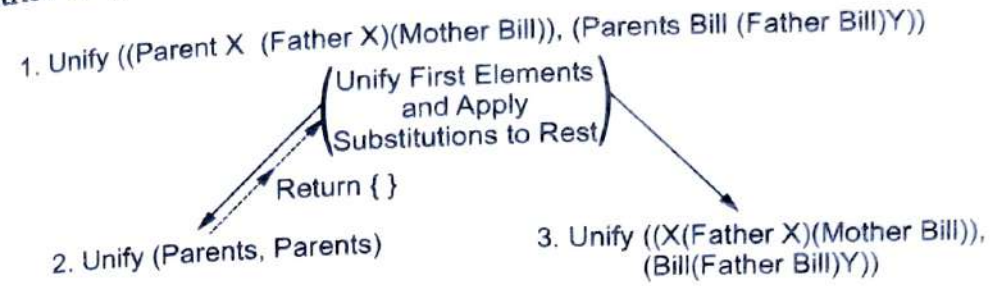


Fig. 6.4.1 Unification Process I

- Further steps in unification of (Parents X (Father X) (Mother bill) and (Parents bill (Father bill) Y)

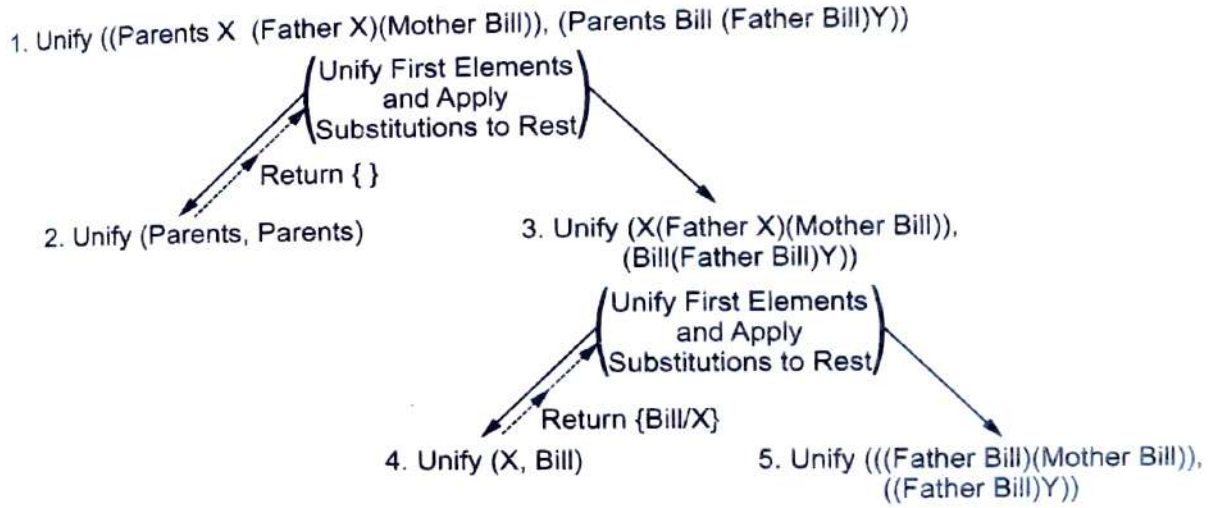


Fig. 6.4.2 Unification Process II

- Final trace of the unification of (parents X(father X) (mother bill)) and (parents bill (father bill) Y). (See Fig. 6.4.3 on next page.)

6.4.7 Storage and Retrieval of Data from Knowledge Base

1) Knowledge base has basic two functions for storing and then fetching the data for use.
   i. The TELL function is used to store data in knowledge base.
   ii. The FETCH function is used to retrieve data from knowledge base.

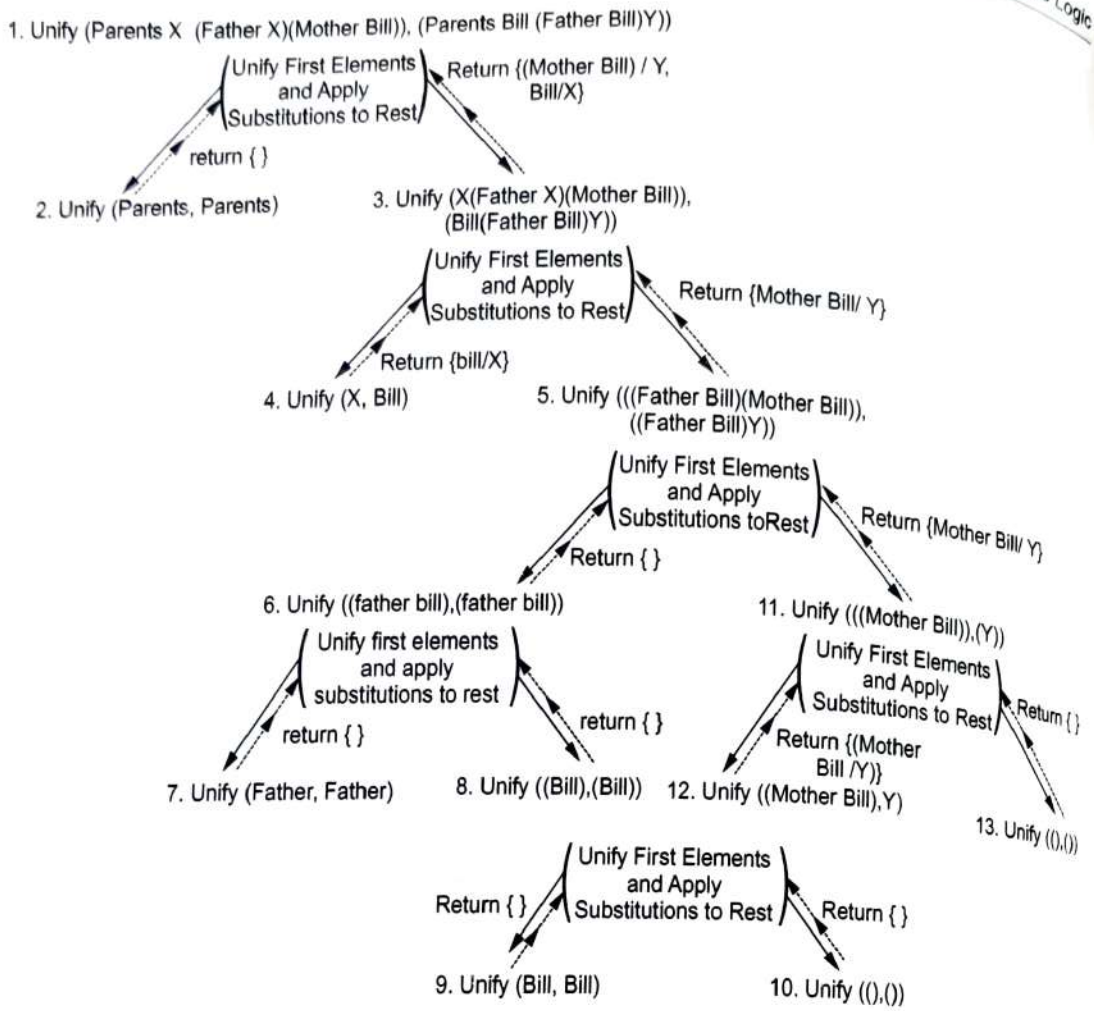


Fig. 6.4.3 Unification Process III

2) Internally TELL function is implemented as STORE(S) primitive (basic functionality), which stores a sentence S into the knowledge base.

3) Internally ASK function is implemented as FETCH(q) primitive (basic functionality), which returns all unifiers such that the query 'q' unifies with some sentence in knowledge base.

Knowledge base implemented as a long list

1) The simplest way to implement STORE and FETCH is to keep all the facts in the knowledge base in one long list.

2) The given query 'q' will then call UNIFY (q, s) for every sentence S in the list.

3) Such a process is inefficient in terms of time taken because it will unify each and every statement of knowledge base (which may unify unnecessary sentences).

4) In such case FETCH can be made efficient by unifying those sentences only which will have a chance to be unified.

For example :

There is no point in trying to unify

Meets (Manmohan, x) with the sensence,

Father (Ram, Manmohan)

5) We can avoid such unification by indexing the facts in the knowledge base.

6) A simple indexing called predicate indexing puts all the "Meets" facts in one bucket (stored collection) and all the 'Father' facts in another bucket. These buckets (stored collection) can be stored in hash table for fast access.

7) Predicate indexing is useful when there are many predicate symbols and there are few clauses for each symbol.

If there are many clauses for single symbol then the bucket of this symbol will be too large and again the search will be inefficient (more time incurred).

8) We can solve large buckets problem with the help of multiple indexing.

We can index fact both by predicate and by first or second argument of the fact by using composite (combined) key for hash table.

Then we can simply construct the key from the query and retrieve exactly those facts that unify with the query.

If we index the fact with first argument combined with predicate then it can be stored under multiple index keys.

(P which will answer various queries, unified by this predicate).

9) Given a sentence to be stored in a knowledge base it is possible to construct indices for all queries that unify with it.

For example :

The fact TEACHES (Radhakrishnan, Dipu)

Will have following sets of queries

a. TEACHES (Radhakrishnan, Dipu)

Does RadhaKrishnan teaches Dipu ?

b. TEACHES (x, Dipu)

Who teaches Dipu ?

c. TEACHES (Radhakrishnan, y)

Whom does Radhakrishnan teach ?

d. TEACHES (x, y)

Who teaches whom ?

10) The subsumption lattice :

i. The above set of queries are said to form a collection tree called as **Subsumption lattice.**

ii. Properties of subsumption lattice :

a. The child of any node in the lattice is obtained from its parent by single substitution.

b. The highest common descendent of any two nodes is the result of applying their most general unifier.

c. The portion of the lattice, above any ground fact can be constructed systematically as shown in following figure,

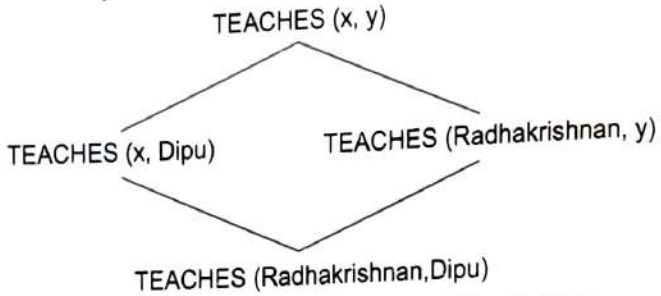


Fig. 6.4.4 Portion of subsumption lattice

A subsumption lattice whose lowest is the sentence,

TEACHES (Radhakrishnan, Dipu).

d. A sentence with repeated constants has a slightly different lattice as shown in following figure.

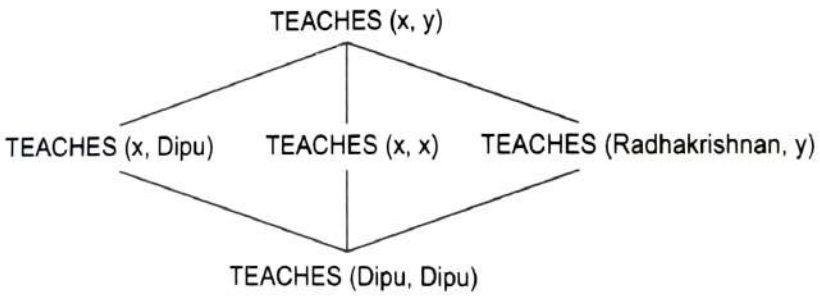


Fig. 6.4.5 Portion of subsumption lattice with repeated constants

e. Function symbols and variables in the sentence will have different lattice structure.

f. The lattice representation work well when it contains small number of nodes.

g. For a predicate with n arguments, the lattice contains O (2^n) nodes.

6.4.8 First Order Definite Clauses

1) They are disjunctions of literals of which exactly one is positive.
2) A definite clause is either atomic sentence or is an implication whose antecedants (left hand side clause) is a conjunction of positive literals and consequent (Right hand side clause) is a single positive literal.

   For example :

   Princess (x) $\wedge$ Beautiful (x) $\Rightarrow$ GoodHearted (x)

   Princess (x)

   Beautiful (x)
3) First order logic literals can include variables, in which case those variables are assumed to be universally quantified.
4) Generally, universal quantifiers are omitted while writing definite clauses.
5) Definite clauses are suitable formal forms to used with generalized modus ponen.
6) Not every knowledge base can be converted into a set of definite clause, because of **single-positive-literal condition.**
7) Set of FOL definite clauses with no function symbols is called as **Datalog knowledge base.**
8) The absence of function symbols in Datalog Knowledge base helps to make inferencing easy.

6.4.9 Inferencing Algorithm for First Order Logic

6.4.9.1 Forward Chaining (Lifted Forward Chaining)

1) Forward chaining is applied to first order logic definite clauses.
2) Definite clauses such as, situations $\Rightarrow$ Response are especially useful for systems that make inference in response to newly arrived information. Forward chaining can be more efficient than resolution theorem proving if systems are represented in definite clauses.
3) Starting from known fact, it triggers all the rules whose premises are satisfied, and add their conclusions to known facts.
4) The process repeats until query is answered (assuming there is only one answer expected) or no new facts are added (that is query remains unanswered).
5) Fact is **not considered** new if it is simply renamed version of existing fact. One sentence is renaming of another sentence if they are identical except for the names of the variable.

For example :

Likes (x, Rose) and

Likes (y, Rose)

are renaming of each other because they differ only in variable names. Both means the same "Everyone likes Rose".

6) In the algorithm a situation comes when no more new inferences are possible. At this stage, knowledge base is called as **Fixed Point** of the inference process.

First Order Logic Fixed Point (KnowledgeBases) can include universaly quantified atomic sentences.

7) Performance measurement of forward chaining : -

i. It is sound. Every inference is just an application of generalized Modus ponen which is sound.

ii. It is complete for definite clause knowledge base. It answers query if knowledge base entails it. One can prove completeness of Datalog in simple steps.

iii. If K- is the maximum arity (number of arguments) of any predicate, P- the number of predicates and n - number of constant symbols then there can be maximum (P*n)K distinct ground facts. After this, algorithm will reach fixed point.

8) Definite clauses with function symbols can generate infinitely many new facts. So extra care should be taken here to avoid infinite knowledge base growth.

9) If the query has no answer the algorithm may fail to terminate in some cases.

Forward chaining procedure :

Function FOL-FC-ASK (KB, α) returns a substitution or false

inputs : KB, the knowledge base, a set of first-order definite clauses. α, the query, at atomic sentence.

Local variables : New, the new sentences inferred on each iteration.

repeat until new is empty

new $\leftarrow$ {}

For each sentence r in KB do

{ $P, \wedge \ldots\ldots \wedge P_n \Rightarrow q) \leftarrow$ STANDARDIZE - APART (r) / * Resolve name clash */

For each θ such that SUBST (θ, $P_i \wedge \ldots. \wedge P_n$) = SUBST ($\theta$, $P_i \wedge \ldots. P_n$)

For some $P'_i \ldots. P'_n$ in KB

$q' \leftarrow$ SUBST (θ, q)
if q' is not a renaming of some sentence already in KB or new
then do
add q' to new
$\phi \leftarrow$ UNIFY (q', α)
if ϕ is not fail then return ϕ add new to KB.
return false.

Forward chaining example :

Consider the following problem :

"The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colone West, who is American".

We need to prove "West is Criminal".

The first order definite clause for above problem domain are as follows : -

1) ".... it is a crime for an American to sell weapons to hostile nations" : -

   $\subset$ American (x) $\wedge$ Weapon (y) $\wedge$ Sells (x, y, z) $\wedge$ Hostile (z) $\Rightarrow$ Criminal (x).

2) "Nono...has some missiles". The sentence $\exists$ x Owns (Nono, x) $\wedge$ Missile (x) is transformed into two definite clauses by existential elimination, introducing a new constant M_1 : $\subset$

   Owns (Nono, M_1)

   Missile (M_1)

3) "All of its missiles were sold to it by Colonel West" : $\subset$

   Missile (x) $\wedge$ Owns (Nono, x) $\Rightarrow$ Sells (West, x, Nono)

4) "We will also need to know that missiles are weapons :

   Missile (x) $\Rightarrow$ Weapon (x)

5) We must know that an enemy of America counts as "hostile" : $\subset$

   Enemy (x, America) $\Rightarrow$ Hostile (x).

6) "West, who is American .... " :$\subset$ American (West).

7) "The country Nono, an enemy of America ...." : $\subset$ Enemy (Nono, America).

Note

- *In the proof tree initial facts appear at the bottom level.*
- *Facts inferred on the first iteration appear in the middle level.*
- *Facts inferred on the second interation appear at the top level.*

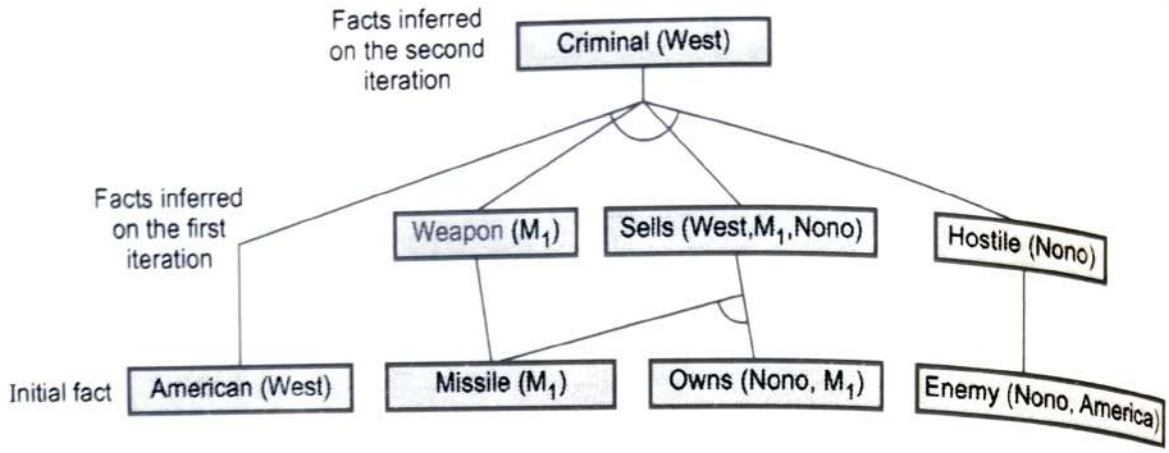


Fig. 6.4.6 A proof tree generated by forward chaining algorithm on crime example

Efficiency of Forward Chaining :

Source of complexity for first order logic forward chaining - ASK are as follows,

1) Inner loop searches all unifiers therefore Pattern matching is expensive.
2) Every rule is tested again in each iteration.
3) Algorithm may produce many facts not relevant for the goal.

1) Matching rules with known facts :

i. To apply Missile (x) ⇒ Weapon (x), find all facts which unify with Missile (x).

↳ This can be done in constant time using indices.

ii. To apply Missile (x) ∧ Owns (Nono, x) ⇒ Sells (West, x, Nono),

We can either,

↳ First find all objects Nono owns and then test if the objects are missiles or

First find all missiles and then test if they are owned by Nono.

↳ This is the problem of conjugate order.

a) Choose and order which minimizes overall costs (depends on knowledgebase).

b) Use heuristics, example, "most constraint variable" aka "Minimum Remaining Value (MRV)".

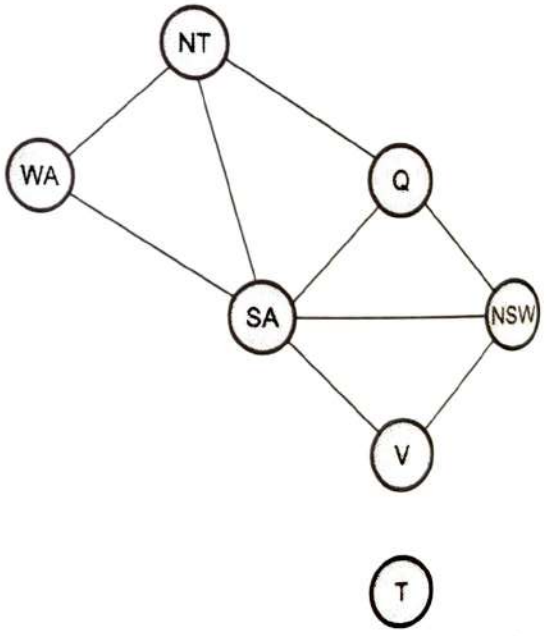


Fig. 6.4.7 Constrait graph for colouring the map

c) Pattern matching has strong relation to CSPs !

d) Every conjunct is a constraint for the contained variables.

iii. Hard matching example :

Consider the constraint graph for colouring the map [Refer chapter 3 for detail description]

- Every CSP with finite domain can be expressed as follows : -
- As a single definite clause : -

  Diff (wa, nt) $\wedge$ Diff (wa, sa) $\wedge$ Diff (nt, q)

  $\wedge$ Diff (nt, sa) $\wedge$ Diff (q, nsw) $\wedge$ Diff (q, sa)

  $\wedge$ Diff (nsw, v) $\wedge$ Diff (nsw, sa) $\wedge$ Diff (v, sa) $\Rightarrow$ Colorable ( )
- With the according facts : -

  Diff (Red, Blue) Diff (Red, Green)

  Diff (Green, Red) Diff (Green, Blue)

  Diff (Blue, Red) Diff (Blue, Green)

- Colorable ( ) is inferred iff the CSP has a solution,
- CSPs include 3SAT as a special case, hence matching is NP - hard.
- NP completeness of forward chaining in its inner loop put into following aspects perspective : -

a) Most rules in real world knowledge bases are small and simple in contrast to CSP formulation.

i) In database world :

1. Often there are limits for rule sizes and predicate arity.
2. Inference complexity is just dependent on number of facts.

b) Generate subclasses of rules which are efficient.

1. Each Datalog clause can be seen as CSP.
2. Solve the CSP, if it is a tree, it is solvable in linear time.
3. Same can be done for the rules, for example, delete SA in prior example. That is,

Diff (wa, nt) $\wedge$ Diff (nt, q) $\wedge$ Diff (q, nsw) $\wedge$ Diff (nsw, v) $\Rightarrow$ Colorable ( )

c) It avoids redundant rule matches.

2) Incremental forward chaining :

1. Simple first order logic - forward chaining - ASK would repetitively and redundantly match rules,

example,

Missile (x) ⇒ Weapon (x) during second iteration.

Observation :

No need to match rule on iteration k if a premise wasn't added on iteration (k–1)!

a. Match each rule whose premise contains a newly added positive literal.

b. Match rule its premises contain a fact P_i' which unifies with a fact P_i' derived during k – 1.

c. The algorithm

```
The algorithm for every iteration k
 for every rule r
   for each Pi in premises (r)
                for each Pi derived during k–1
          if unify (Pi, P'i) ⇒ match (r)
```

2. Database indexing allows O(1) retrival of known facts.

   Example :

   Query Missile (x) retrieves Missile (M_1)

3. Redundancy can be avoided if partial derivations are buffered :

   Rete - Algorithm : It uses a data propagation network which propagates variable bindings.

   Every node is literal from premises.

4. Rete and successors were basis for production systems like, XCON (DEC) hardware configuration and OPS-5, a general language or for cognitive architectures like ACT (Anderson, 1983) or SOAR (Laired et al., 1987)

3) Irrelevant Facts :

1. Deduction of facts is not required for a given goal (similar to forward chaining in propositional logic.)
2. Solution : -

   (i) Use of subset of rules (see propositional logic).

   (ii) From deductive database research : Use a magic set

- Only consider rules with a given variable binding : For e.g. if goal is Criminal (West), then the rule that concludes Criminal (x) can be written with extra conjuct that constraints the value of x. That is,

  Magic (x) ∧ American (x) ∧ Weapon (y) ∧ Sells (x, y, z) ∧ Hostile (z) ⇒ Criminal (x)

6.4.9.2 Backward Chaining (Lifted Backward Chaining)

1) It use generalized Modus ponen backwards to prove query q, and work backwards.

   - It checks if q is known already.

   - Otherwise it proves by backward chaining all premises of a rule concluding q.

2) The list of goals can be thought of as a "stack" waiting to be worked on; if all of them can be satisfied, then the current branch of the proof succeeds. The algorithm takes the first goal in the list and finds every clause in the knowledge base whose positive literal, or head, unifies with the goal. Each such clause creates a new recursive call in which the premise, or body, of the clause is added to the goal stack.

3) The algorithm uses composition of substitutions. COMPOSE (θ_1, θ_2) is the substitution whose effect is identical to the effect of applying each substitution in turn. That is,

   SUBST (COMPOSE (θ_1, θ_2), P) = SUBST (θ_2, SUBST (θ_1, P)).

   In the algorithm, the current variable bindings, which are stored in θ, are composed with the bindings resulting from unifying the goal with the clause head, giving a new set of current bindings for the recursive cell.

The backward chaining Procedure :

Function FOL-BC-ASK (KB, goals, θ) return a set of substitutions.

Inputs : KB, a knowledgebase goals, a list of conjucts forming a query (θ already applied)

θ, the current substitution, initially the empty substitution { }

Local variables : answers , a set of substitutions, initially empty.

if goals is empty then return {θ}

$q' \leftarrow$ SUBST (θ, FIRST (goals))

For each sentence r in knowledge base where STANDARDIZE - APART

(r) = ($P_1 \wedge ..... \wedge P_n \Rightarrow q$)

and $\theta' \leftarrow$ UNIFY (q, q') succeeds new-goals $\leftarrow$ [P_1, ..., P_n / REST (goals)]

answers $\leftarrow$ FOL-BC-ASK (KB, new-goals, COMPOSE (θ', θ)) U answers

return answers.

Properties of backward chaining :

1) It uses depth-first search for proof. It's memory requirement is linear in size of proof.

2) It is incomplete due to infinite loops. It should check if new subgoal is already an goal stack.

3) It is inefficient due to repeated subgoals (success and failure). It should cache previous results and check for new subgoal.

4) We can have two versions of it. Find any solution, OR Find all solutions.

5) It is used for logic programming : Prolog

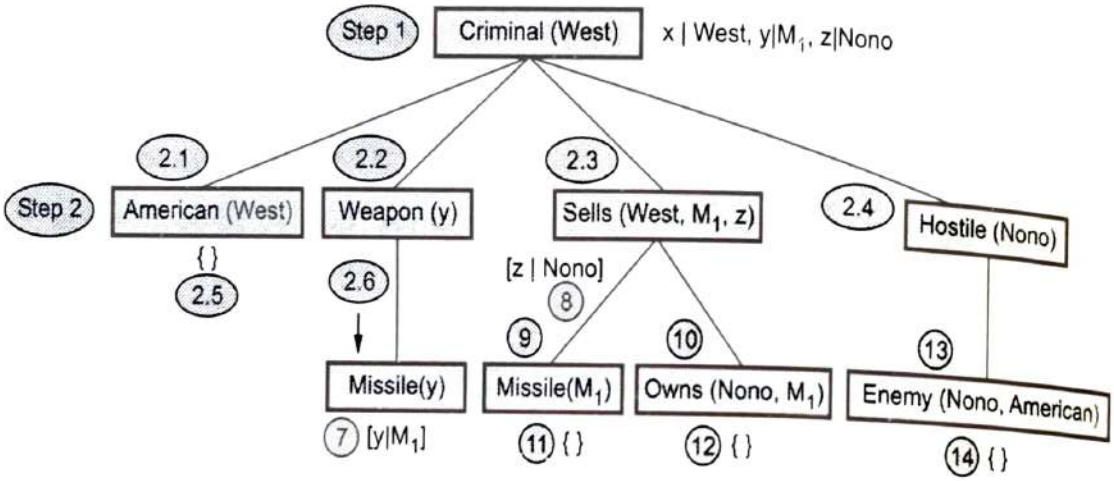


Fig. 6.4.8 Proof tree generated by backward chaining to prove that 'West is criminal'

- **Backward chaining example :**

Note *The tree should be read depth first, left to right.*

To prove criminal (west), we have to prove the four conjucts below it.

Some of the required conjucts are in knowledgebase and some conjucts are needed to be proved.

Bindings of for each successful unification are shown next to the corresponding subgoal.

Once one subgoal in a conjuction suceeds, its substitution is applied to subsequent subgoals. Thus, by the time FOL-BC-ASK gets to the last conjuct, originally. Hostile (z), z is already bound to Nono.

6.4.9.3 Forward Chaining Vs Backward Chaining

1) Forward chaining is data driven.
   - It is automatic unconscious processing.
   - Example - Object reorganition, routine decisions.
   - It may do lots of work that is irrelevant to the goal.

2) Backward chaining is goal driven.
- It is appropriate for problem solving.
- Example : Where are my keys ?,
  How do I get into a PhD programme ?
- Complexity of backward chaining can be much less than linear in size of knowledgebase.

6.4.10 Resolution

It is procedure of inferencing the given statement based on available data. The resolution procedure we are going to study is the lifted version of resolution procedure in propositional logic.

In first order logic, for resolution, it requires the sentences in Conjunctive Normal Form (CNF). That is, a conjunction of clauses, where each clauses is a disjunction of literals. Literals can contain variables, which are assumed to be universally quantified.

- General format of conjunctive normal form with exactly K literals per clause is
  $(l_{1,1} \vee .... \vee l_{1,K}) \wedge .... \wedge (l_{n,1} \vee ..... \vee l_{n,K})$
- Generally there can be any number of literals in a single clause.
  For example :
  $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$
- CNF statement with quantifiers -
  $[\forall x\ A(x) \vee \forall y\ B(y)] \wedge [\exists z\ C(z)]$
- Consider another example :
  $\forall x$ **American (x)** $\wedge$ **Weapon (y)** $\wedge$ **Sells (x, y, z)** $\wedge$ **Hostile (z)** $\Rightarrow$ **Criminal (x)**
  The above sentence in CNF becomes, $\neg$ **American (x)** $\vee \neg$ **Weapon (y)** $\vee$
  $\neg$ **Sells (x, y, z)** $\vee \neg$ **Hostile (z)** $\vee$ **Criminal (x)**
- Note that,
  CNF sentence will be unsatisfiable only when original sentences is unsatisfiable
  Therefore we will have resolution proofs by contradiction on the CNF sentences.

• The general steps for converting first order logic sentences to conjunctive normal form

1. $(\forall x)(P(x)) \Rightarrow ((\forall y)(P(y) \Rightarrow P(F(x,y))) \wedge \neg(\forall y)(Q(x,y) \Rightarrow P(y)))$(like $A \Rightarrow B \wedge C$)

2. Eliminate $\Rightarrow$

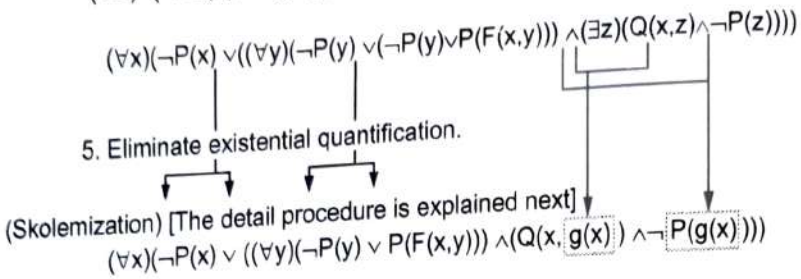
$(\forall x)(\neg P(x) \vee ((\forall x)(\neg P(y) \vee P(F(x,y))) \wedge \neg(\forall y)(\neg Q(x,y) \vee P(y)))$

3. Reduce scope of negation :

$(\forall x)\ (\neg P(x) \vee ((\forall y)\ (\neg P(y) \vee P\ (F(x, y))) \wedge (\exists y)\ (Q(x, y) \wedge \neg P(y))))$

4. Standardize variables :

$$(\forall x)\ (\neg P(x) \vee ((\forall y)\ (\neg P(y) \vee P(F(x, y))) \wedge (\exists z)\ (Q(x, z) \wedge \neg P(z))))$$



6. Drop universal quantification symbols :

$$(\neg P(x) \vee ((\neg P(y) \vee P(F(x, y))) \wedge (Q(x, g(x)) \wedge \neg P(g(x)))))$$

7. Convert to conjunction of disjunctions :

$$(\neg P(x) \vee \neg P(y) \vee P(F(x, y))) \wedge (\neg P(x) \vee Q(x, g(x))) \wedge (\neg P(x) \vee \neg P(g(x))).$$

$$(\neg P(x) \vee \neg P(y) \vee P(F(x, y))) \wedge (\neg P(x) \vee Q(x, g(x))) \wedge (\neg P(x) \vee \neg P(g(x)))P$$

8. Create separate clauses :

$$\neg P(x) \vee \neg P(y) \vee P(F(x, y))$$

$$\neg P(x) \vee Q(x, g(x))$$

$$\neg P(x) \vee \neg P(g(x)).$$

9. Standardize variables :

$$\neg P(x) \vee \neg P(y) \vee P(F(x, y))$$

$$\neg P(z) \vee Q(z, g(z))$$

$$\neg P(w) \vee \neg P(g(w)).$$

Skolemize :

1) Skolemization is the process of removing existential quantifier by elimination.
2) This methods converts a sentence with existential quantifier into a sentence without existential quantifier such that the first sentence is satisfiable if and only if the second is.
3) To eliminate an existential quantifier, replace each occurrence of its variable by a skolem function whose arguments are the variables of universal quantifier whose scope includes the scope of the existential quantifier being eliminated.
4) If the existential quantifier being eliminated is not within the scope of any universal quantifier, the skolem function has no arguments that is, it is a constant.

Skolemization example :

1. $\forall x \exists y$ (Person (x) $\wedge$ Person (y)) $\Rightarrow$ Loves (x, y)

   is converted to

   $\forall x$ (Person (x) $\wedge$ Person f(x) $\Rightarrow$ Loves (x, f(x))

   Where f(x) specifies the person that x Loves.

2. The sentence "Everyone has a brain" is represented as

   $\forall x$ Person (x) $\Rightarrow \exists y$ Brain (y) $\vee$ Has (x, y)

   If we simply substitute the constant B for the existentially qualified variable y, we get a sentence that says "Everyone has the same brain", not what we want to say!

   $\forall x$ Person (x) $\Rightarrow$ Brain (B) $\wedge$ Has (x, B)

   Using the Skolem function B(x) to represent a function that denotes the object that is x's brain, the correct skolemization of our original sentence is,

   $\forall$ x Person (x) $\Rightarrow$ Brain (B(x)) $\wedge$ Has (x, B(x)).

3. $\exists$ x Rich (x) becomes Rich (G1) where G1 is a new "Skolem constant"

   $\exists$ k $\frac{d}{dy}(K^y) = K^y$ becomes $\frac{d}{dy}(e^y) = e^y$ more tricky when $\exists$ is inside $\forall$

Example : "Everyone has a heart".

$\forall x$ Person (x) $\Rightarrow \exists y$ Heart (y) $\wedge$ Has (x, y)

Incorrect : $\forall x$ Person (x) $\Rightarrow$ Heart (H1) $\wedge$ Has (x, H1).

Correct : $\forall x$ Person (x) $\Rightarrow$ Heart (H(x)) $\wedge$ Has (x, H(x))

[If x has a y we can infer that y exists. However, its existence is contingent on x, thus y is a function of x as H(x)].

Where H is new symbol ("Skolem" function)

Skolem function arguments : All **enclosing** universally quantified variables.

Detail example :

Converting first order logic (predicate logic) sentences in to CNF :

1) The conversion procedure is almost the same as Propositional logic with the major difference arises where we need to eliminate existential quantifier.
2) We will consider sentence,

   "Everyone who loves all animals is loved by someone".

   The first order logic representation for above sentence is,

   $\forall$ x [$\forall$y Animal (y) $\Rightarrow$ Loves (x, y)] $\Rightarrow$ [$\exists$y Loves (y, x)]

3) The steps to convert above first order logic sentence into CNF are as follows,

Consider following sentence,

$\forall x\ [\forall y\ \text{Animal}\ (y) \Rightarrow \text{Loves}\ (x, y)] \Rightarrow [\exists y\ \text{Loves}\ (y, x)]$

The steps are as follows :

1. Elimination implications :

$\forall x\ [\neg\ \forall y \neg\ \text{Animal}\ (y) \vee \text{Loves}\ (x, y)] \vee [\exists y\ \text{Loves}\ (y, x)]$

2. Move $\neg$ inwards :

In addition to the usual rules for negated connectives, we need rules for negated quantifier. Thus we have,

i. $\neg\ \forall x\ P$ becomes $\exists x \neg\ P$

ii. $\neg\ \exists x\ P$ becomes $\forall x \neg\ P$.

Our sentence goes through the following transformations :

i. $\forall x\ [\ \exists y \neg\ (\neg\ \text{Animal}\ (y) \vee \text{Loves}\ (x, y))] \vee [\exists y\ \text{Loves}\ (y, x)]$.

ii. $\forall x\ [\ \exists y \neg \neg\ \text{Animal}\ (y) \wedge \neg\ \text{Loves}\ (x, y)] \vee [\exists y\ \text{Loves}\ (y, x)]$.

iii. $\forall x\ [\ \exists y\ \text{Animal}\ (y) \wedge \neg\ \text{Loves}\ (x, y)] \vee [\exists y\ \text{Loves}\ (y, x)]$.

Note *How a universal quantifier ($\forall y$) in the premise of the implication has become an existential quantifier. The sentence now reads "Either there is some animal that x doesn't love, or (if this not the case) someone loves x," Clearly, the meaning of the original sentence has been preserved.*

3. Standardize Apart Variables (Remove name clash)

For sentences like $(\forall x\ P(x)) \vee (\exists x Q(x))$ which use the same variable name twice, change the name of the variables. This avoids confusion later when we drop the quantifier. Thus we have $\forall x\ [\exists y\ \text{Animal}\ (y) \wedge \neg\ \text{Loves}\ (x, y)] \vee [\ \exists\ z\ \text{Loves}\ (z, x)]$.

4. Apply this rule to our sample sentence, we obtain

$\forall x\ [\text{Animal}\ (A) \wedge \neg\ \text{Loves}\ (x, A)] \vee \text{Loves}\ (B, x)$.

which has the wrong meaning entirely : It says that everyone either fails to love a particular animal A or is loved by some particular entity B. In fact, our original sentence allows each person to fail to love a different animal or to be loved by a different person. Thus we want the skolem entities to depend on x.

$\forall x\ [\text{Animal}\ (F(x)) \wedge \neg\ \text{Loves}\ (x, F(x))] \vee \text{Loves}\ (G(x), x)$.

Here F and G are Skolem functions.

5. Drop-universal quantifiers :

   At this point, all remaining variables must be universally quantified. More over, the sentence is equivalent to one in which all the universal quantifier have been moved to the left. We can therefore drop the universal quantifier.

   [Animal (F(x)) $\wedge \neg$ Loves (x, F(x))] $\vee$ Loves (G(x), x).

6. Distribute $\wedge$ over $\vee$:

   [Animal (F(x)) $\vee$ Loves (G(x), x)] $\wedge$ [$\neg$ Loves (x, F(x))] $\vee$ Loves (G(x), x)]

   This step may also require flattening out nested conjunctions and disjunctions.

   The sentence is now in CNF and consists of two clauses.

- **The resolution inference rule :**

1) The resolution rule for first order clauses is simply a lifted version of the propositional resolution rule.
2) Two clauses, which are assumed to be standardized apart so that they share no variables, can be resolved if they contain complementary literals.
3) First order literals are complementary if one unifies with the negation of the other.
4) The resolution inference rule is

$$\frac{l_1, V \ldots V\, l_k, \qquad m_1 V \ldots V\, m_n}{\text{SUBST}(\theta, l_1, V \ldots V\, l_{i-1}\, V,\ l_{i+1} V \ldots V\, l_k\, V_m, V \ldots V_{mj-1} V_{mj+1} V \ldots V_{m_n})}$$

   where UNIFY (l_i, $\neg$ mj) = θ for example,

   we can resolve the two clauses,

   [Animal (F(x)) $\vee$ Loves (G(x), x)] and [$\neg$ Loves (u, v) $\vee \neg$ kills (u, v)]

   by eliminating the complementary literals Loves (G(x), x) and $\neg$ Loves (u, v), with unifier.

   θ = {u/G(x), v/x}, to produce the resolvent clause.

   [Animal (F(x)) $\vee \neg$ kills (G(x), x)]

5) The rule we have just given is the binary resolution rule, because it resolves exactly two literals. The binary resolution rule by itself does not yield a complete inference procedure.
6) The full resolution rule resolves subsets of literals in each clause that are unifiable.
7) Another approach is to extend factoring. Factoring means the removal of redundant literals in first order case. Propositional factoring reduces two literals

to one if they are identical; first order factoring reduces two literals to one if they are unifiable.

Statement proving by resolution :

Resolution proves that KB $\models \alpha$ by proving KB $\wedge \neg \alpha$ unsatisfiable, i.e, by deriving the empty clause. It is a proof by contradiction.

Following are steps taken in resolution proof,

1) Convert all problem statements to first order logic.
2) Convert first order logic statements into conjunctive normal form.
3) Assert the negation of the goal.
4) Resolve clauses together until FALSE is derived.

Algorithm always chose to resolve with a clause whose positive literal unifies with the leftmost literal of the 'current' clause. This is exactly what happens in backward chaining. (See Fig. 6.4.9 on next page.)

Resolution example 1 :

Problem statement in English,

1. "Everyone who loves all animals is loved by someone".
2. "Anyone who kills an animal is loved by no one."
3. "Jack loves all animals."
4. "Either Jack or Curiosity killed the cat, who is named Tuna."

Statement to prove

"Did Curiosity kill the cat ?"

I) Expressing English Sentences

1) Express the knowledge in FOL.
2) The original sentences.
3) Some background knowledge, and
4) The negated goal G in first-order logic.
5) The original sentences in first order logic.

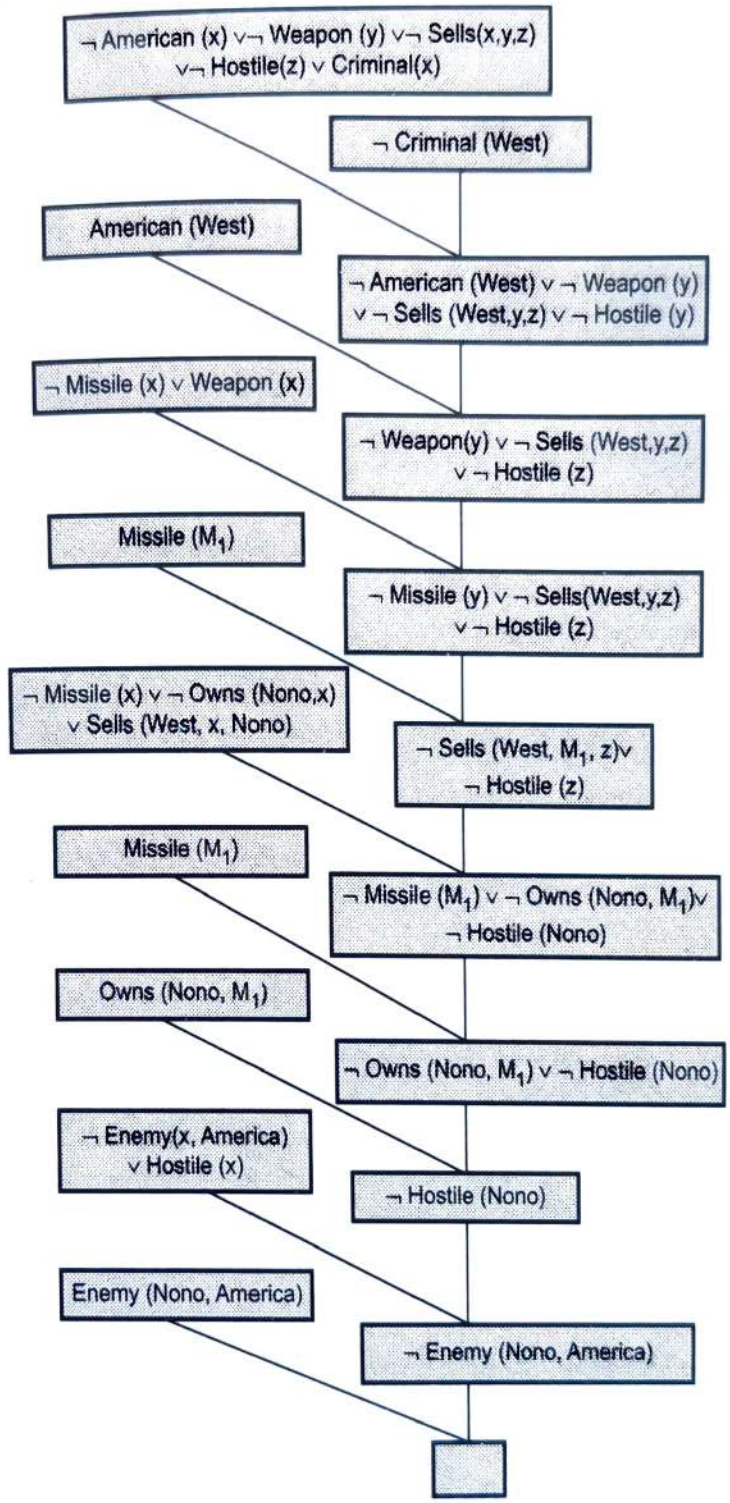


Fig. 6.4.9 A resolution proof that "West is Criminal"

English statement 1 :

A. $\forall x [\forall y \text{ Animal } (y) \Rightarrow \text{Loves } (x, y)] \Rightarrow [\exists y \text{ Loves } (y, x)]$.

English statement 2

B. $\forall x [\exists y \text{ Animal } (y) \wedge \text{kills } (x, y)] \Rightarrow [\forall z \neg \text{Loves } (z, x)]$.

English statement 3

C. $\forall x \text{ Animal } (x) \Rightarrow \text{Loves (Jack, x)}$

English statement 4

D. Kills (Jack, Tuna) $\vee$ Kills (Curiosity, Tuna).

2) Some background knowledge

E. Cat (Tuna)

F. $\forall x \text{ Cat } (x) \Rightarrow \text{Animal } (x)$

3) Negated goal G in first order logic

$\neg$ G. $\neg$ Kills (Curiosity, Tuna).

II) Apply the conversion procedure to convert each sentence to conjunctive form :

A1. Animal (F(x)) $\vee$ Loves (G(X), x)

A2. $\neg$ Loves (x, F(x)) $\vee$ Loves (G(x), x)

B. $\neg$ Animal (y) $\vee$ $\neg$ Kills (x, y) $\vee$ $\neg$ Loves (z, x)

C. $\neg$ Animal (x) $\vee$ Loves (Jack, x)

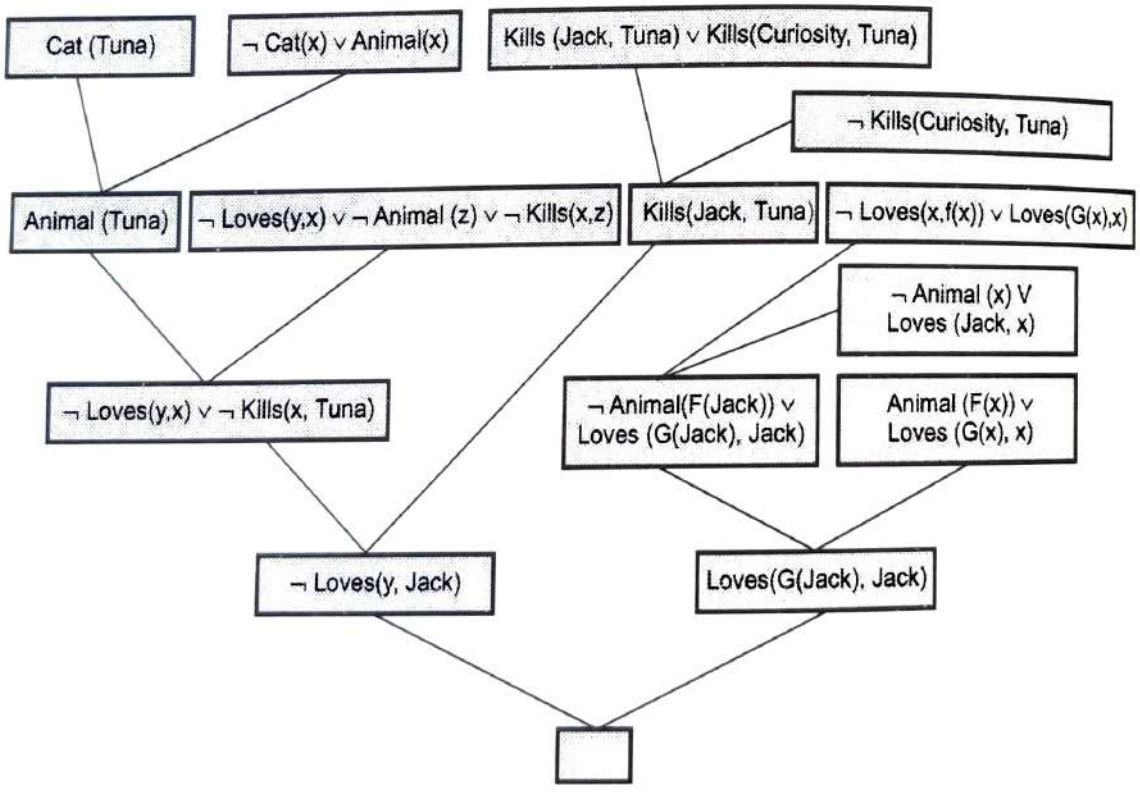


Fig. 6.4.10 Resolution Tree : "Did Curiosity kill the cat ?"

D. Kills (Jack, Tuna) ∨ Kills (Curiosity, Tuna)

E. Cat (Tuna)

F. ¬ Cat (x) ∨ Animal (x)

G. ¬ Kills (Curiosity, Tuna)

- The resolution proof for the statement "Curiosity killed the cat"

Note that, the use of factoring in the derivation of the clause Loves (G(Jack), Jack).

Resolution example 2 :

Anyone passing his/her history exams and winning the lottery is happy. But anyone who studies or is lucky can pass all his/her exams. Nimu did not study. Nimu is lucky. Anyone who is lucky wins the lottery.

To prove : Is Nimu happy ?

Step 1 : Convert to first order logic.

1) Anyone passing his history exams and winning the lottery is happy.

   ∀ x Pass (x, History) ∧ Win (x, Lottery) Happy (x)

2) But anyone who studies or is lucky can pass all his exams.

   ∀ x ∀ y Study (x) ∨ Lucky (x) Pass (x, y)

3) Nimu did not study, but Nimu is lucky.

   ¬ Study (Nimu) ∧ Lucky (Nimu)

4) Anyone who is Lucky wins the lottery.

   ∀ x Lucky (x) Win (x, Lottery)

Step 2 : Convert to CNF

i) Eliminate implications :

1. ∀ x ¬ (Pass (x, History) ∧ Win (x, Lottery)) ∨ Happy (x).
2. ∀ x ∀ y ¬ (Study (x) ∨ Lucky (x)) ∨ Pass (x, y)
3. ¬ Study (Nimu) ∧ Lucky (Nimu)
4. ∀ x ¬ Lucky (x) ∨ Win (x, Lottery)

Move ¬ inward

1. ∀ x ¬ Pass (x, History) ∨ ¬ Win (x, Lottery)) ∨ Happy (x).
2. ∀ x ∀ y (¬ Study (x) ∧ ¬ Lucky (x)) ∨ Pass (x, y).
3. ¬ Study (Nimu) ∧ Lucky (Nimu).
4. ∀ x ¬ Lucky (x) ∨ Win (x, Lottery).

ii) Standardize variables :

No action needed.

iii) Move quantifier left :

No action needed except drop quantifier.

iv) Skolemize :

No action needed.

v) Distribute $\wedge$ over $\vee$

1. $\neg$ Pass (x, History) $\vee \neg$ Win (x, Lottery)) $\vee$ Happy (x).
2. ($\neg$ Study (x) $\vee$ Pass (x, y)) $\wedge$ ($\neg$ Lucky (x) $\vee$ Pass (x, y)).
3. $\neg$ Study (Nimu) $\wedge$ Lucky (Nimu).
4. $\neg$ Lucky (x) $\vee$ Win (x, Lottery).

vi) Flatten nested conjunctions and disjunctions : **no action necessary.**

vii) State as a set of disjunction of literals

1. $\neg$ Pass (x, History) $\vee \neg$ Win (x, Lottery)) $\vee$ Happy (x).
2. a) $\neg$ Study (x) $\vee$ Pass (x, y)

   b) Lucky (x) $\vee$ Pass (x, y)
3. a) $\neg$ Study (Nimu)

   b) Lucky (Nimu)
4. $\neg$ Lucky (x) $\vee$ Win (x, Lottery).

viii) Standardize Variables apart

1. $\neg$ Pass (x1, History) $\vee \neg$ Win (x1, Lottery) $\vee$ Happy (x1).
2. a) $\neg$ Study (x2) $\vee$ Pass (x2, y1)

   b) $\neg$ Lucky (x3) $\vee$ Pass (x3, y2)
3. a) $\neg$ Study (Nimu)

   b) Lucky (Nimu)
4. $\neg$ Lucky (x4) $\vee$ Win (x4, Lottery)

Now, this is in conjunctive normal form (CNF).

Step 3 : Resolution proof procedure

- Assert negation of goal :
  - In this case the goal is to prove **Happy (Nimu).**
  - Add the negation of the goal clause to the knowledge base $\neg$ **Happy (Nimu).**
- Resolve clauses together until FALSE is derived.

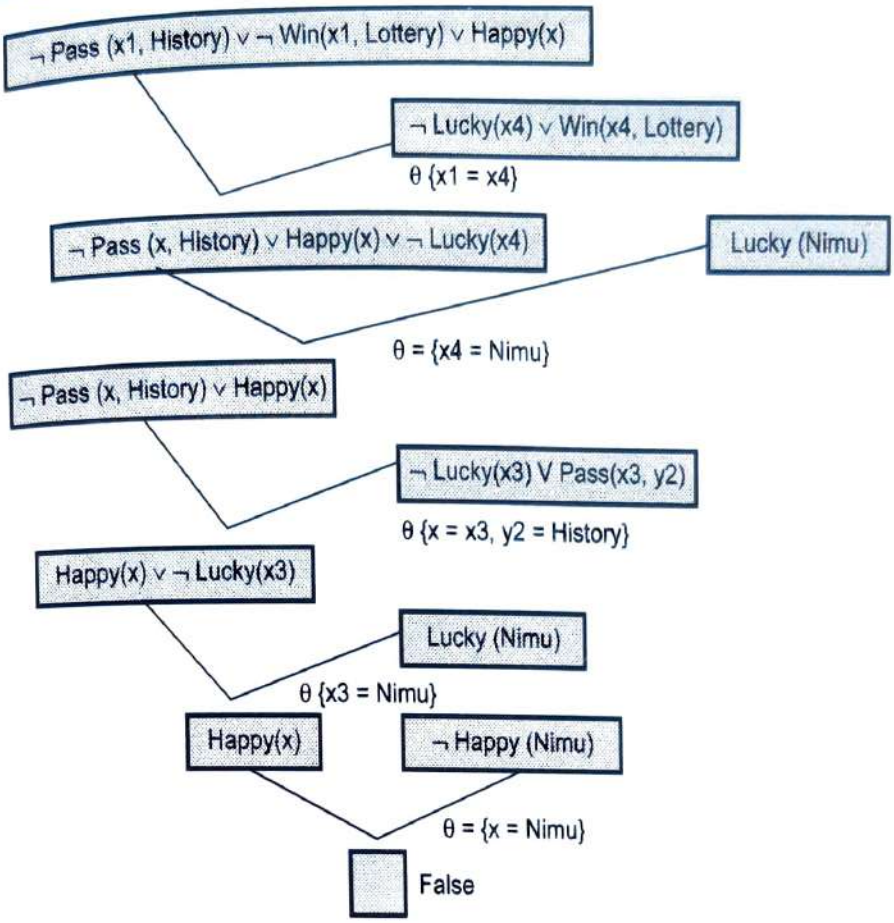


Fig. 6.4.11 Resolution proof tree for the goal-Happy(Nimu)

Resolution example 3 :

- Consider following knowledgebase

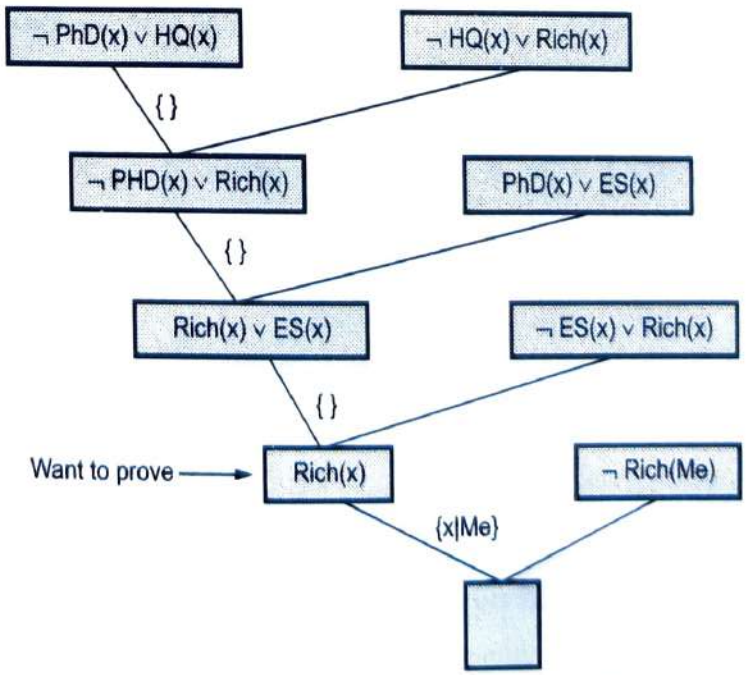


Fig. 6.4.12 Resolution proof tree for the goal Rich (me)

(i) ¬ PhD (x) ∨ HighlyQualified (x)

(ii) PhD (x) ∨ EarlyEarnings (x)

(iii) ¬ HighlyQualified (x) ∨ Rich (x)

(iv) ¬ Early Early Earnings (x) ∨ Rich (x)

- To prove **Rich (me), add ¬ Rich (me)** to the knowledge base

For resolution proof tree refer Fig. 6.4.12.

Resolution example 4 :

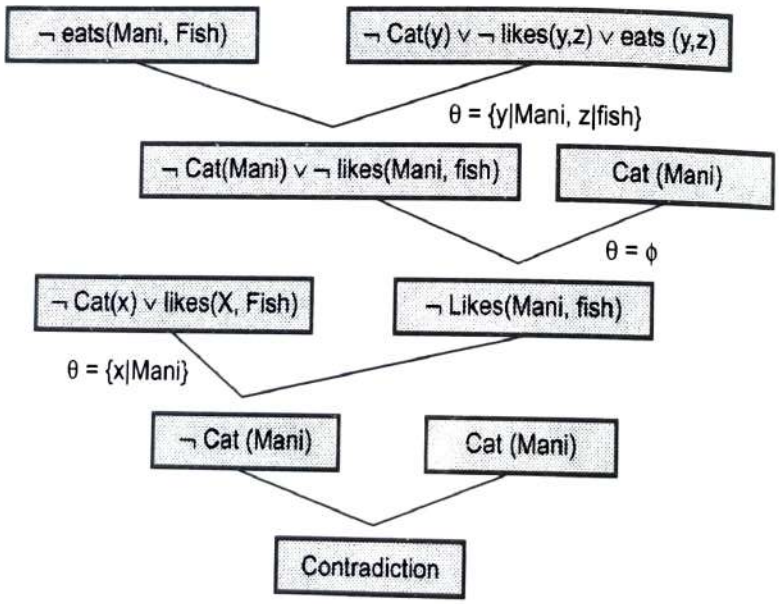
- Consider following knowledge base.

(i) Cats like fish ¬ Cat (x) ∨ likes (x, fish)

(ii) Cats eat everything they like ¬ Cat (y) ∨ ¬ likes (y, z) ∨ eats (y, z).

(iii) Mani is a cat Cat (Mani)

- To prove "Mani eats fish", **eats (Mano, fish) add ¬ eats (Mani, fish)** to the knowledge base.
- **Resolution proof tree**



Negation of goal : ¬ eats (Mani, fish)

Fig. 6.4.13 Resolution proof tree for the goal eats (mani, fish)

Resolution example 5

⇒ Consider following english sentences,

1) Everyone who loves all animals is loved by someone.

2) Anyone who kills an animal is loved by no one.

3) Jack loves all animals.

4) Either Jack or John killed the cat named Tuna.

⇒ Represent above sentences in FOL and prove that **"John killed the cat".**

Step 1 : We need to express

i) The original sentences, ii) Some background knowledge, and

iii) The negated goal G in first order logic.

Step 1 works as follows,

i) The original sentences in first order Logic.

English statement 1 :

A. $\forall x [\forall y \text{ Animal } (y) \Rightarrow \text{Loves } (x, y)] \Rightarrow [\exists y \text{ Loves } (y, x)]$.

English statement 2 :

B. $\forall x [\exists y \text{ Animal } (y) \wedge \text{kills } (x, y)] \Rightarrow [\forall z \neg \text{Loves } (z, x)]$.

English statement 3 :

C. $\forall x \text{ Animal } (x) \Rightarrow \text{Loves (Jack, x)}$.

English statement 4 :

D. kills (Jack, Tuna) $\vee$ kills (John, Tuna)

ii) Some backward knowledge

E. Cat (Tuna)

F. $\forall x \text{ Cat } (x) \Rightarrow \text{Animal } (x)$

iii) Negated goal G in first order logic.

$\neg$ G. $\neg$ kills (John, Tuna)

Step 2 : Apply the conversion procedure to convert each sentence to conjunctive normal form :

A1. Animal (F(x)) $\vee$ Loves (G(x), x)

A2. $\neg$ Loves (x, F(x)) $\vee$ Loves (G(x), x)

B. $\neg$ Animal (y) $\vee \neg$ kills (x, y) $\vee \neg$ Loves (z, x)

C. $\neg$ Animal (x) $\vee$ Loves (Jack, x)

D. kills (Jack, Tuna) $\vee$ kills (John, Tuna)

E. Cat (Tuna)

F. $\neg$ Cat (x) $\vee$ Animal (x)

$\neg$ G. $\neg$ kills (John, Tuna)

Step 3 : The resolution proof tree for the statement "John killed the cat".

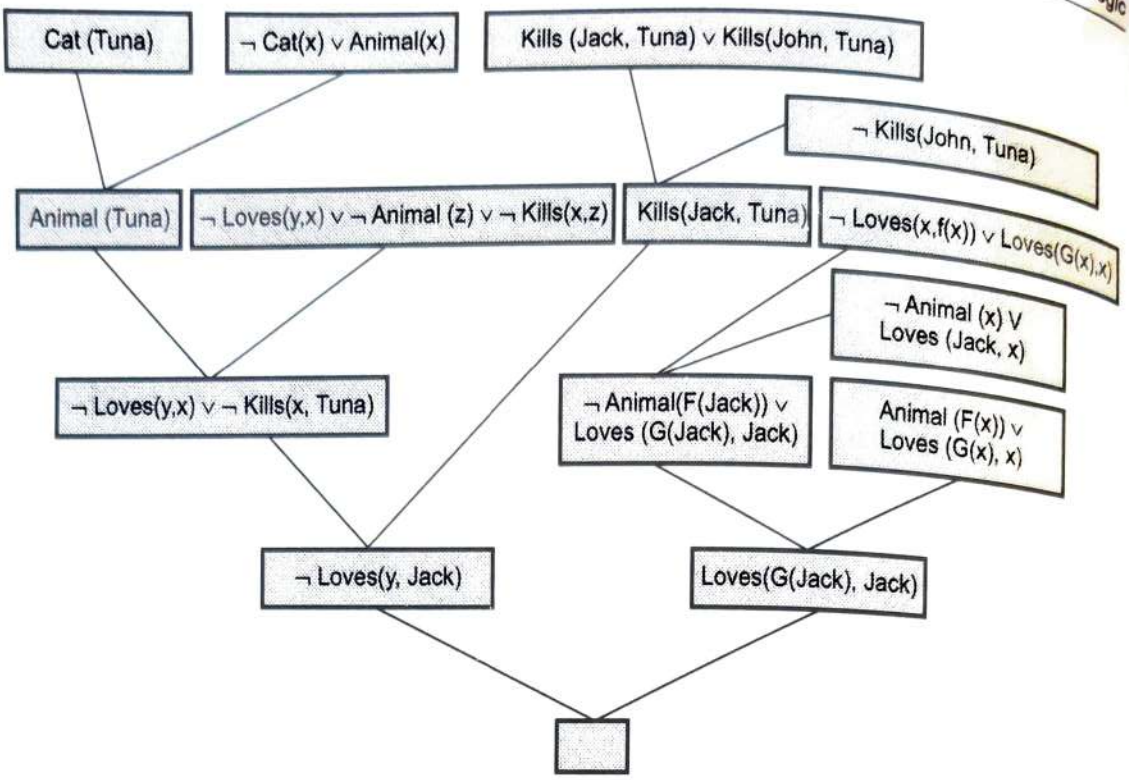


Fig. 6.4.14 Resolution Tree : "Did John kill the cat ?"

- **Completeness of resolution :**

1) Resolution is **refutation complete**, it means that if a set of sentences is unsatisfiable, then resolution will always be able to derive a contradiction.

2) Resolution cannot be used to generate all logical consequences of set of sentences, but it can be used to establish that a given sentence is entailed by the set of sentences. Hence, it can be used to find all answers to a given question, using the negated-goal method.

3) The statement of proof is,

   If S is an unsatisfiable set of clauses, then the application of a finite number of resolution steps to S will yield a contradiction.

A proof that proves, "Resolution proofs for entailment is complete". The basic structure of the proof :

Proof proceeds as follows :

1. First, we observe that if S is unsatisfiable, then there exists a particular set of ground instances of the clauses of S such that this set is also unsatisfiable (Herbrand's theorem).
2. We then appeal to the ground resolution theorem, which states that propositional resolution is complete for ground sentences.

3. We then use a lifting lemma - Lifting lemma states that for every resolution proof on a set of ground formulae, there exists a corresponding proof based on the non ground formulae that originate the ground ones.

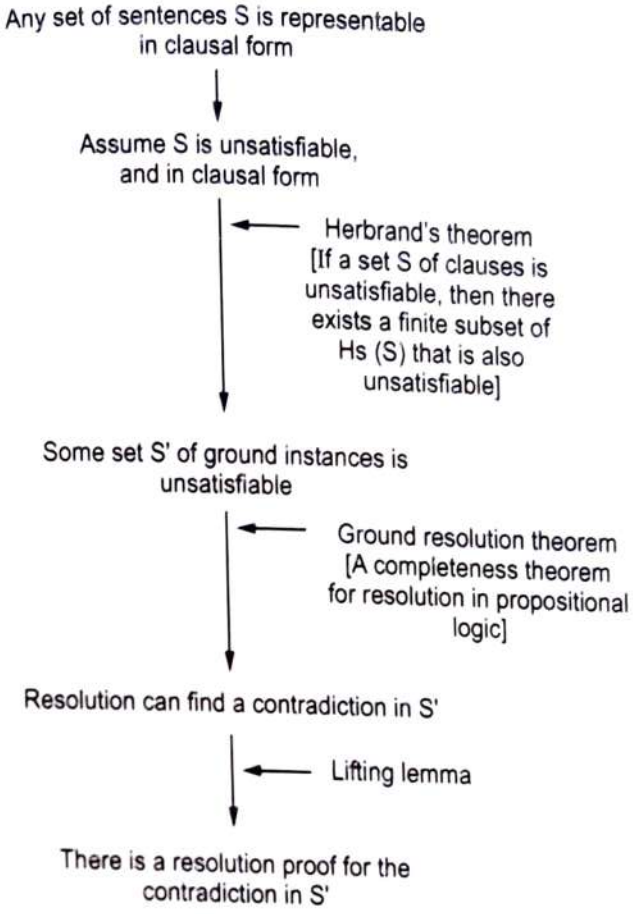


Fig. 6.4.15 Structure of a completeness proof for resolution

- **Resolution strategies :**

1) **Unit preference :**
   - Clauses with just one literal are prefered.
   - Unit resolution : Incomplete in general, complete for horn knowledgebase.

2) **Set of support :**
   - At least one of the clauses makes part from the set of support.
   - Complete if the remainder of the sentences are jointly satisfiable.
   - Using the negated query as set-of-support.

3) **Input resolution :**
   - At least one of the clauses makes part from the initial knowledgebase or the query.
   - Complete for horn, incomplete in the general case.

- Linear resolution : P and Q can be resolved if P is in the original knowledge base or P is an ancestor of Q in the proof tree; complete.

4) Subsumption :

- All sentences subsumed by others in the knowledgebase are eliminated.

Types of Resolution

There are various types of resolution are possible depending on the types and number of parent clauses as follows :

1) Binary resolution - Two clauses having complementary literals are combined as disjuncts to produce a single clause after deleting the complementary literals. For example

$$\neg P(x, a) \vee Q(x) \text{ and}$$

$$\neg Q(b) \vee R(x)$$

is just

$$\neg P(b, a) \vee R(b)$$

The substitution {b | x} was made in the two parent clauses to produce the complementary literals Q(b) and ¬ Q(b) which were then deleted from the disjunction of the two parent clauses.

2) Unit Resulting (UR) Resolution - A number of clauses are resolved simultaneously to produce a unit clause. All except one of the clauses are unit clauses, and that one clause has exactly one more literal than the total number of unit clauses. For example, resolving the set

$$\{ \neg \text{MARRIED}(x, y) \vee \neg \text{MOTHER}(x, z) \vee$$
$$\text{FATHER}(y, z), \text{MARRIED}(\text{sue}, \text{joe}),$$
$$\neg \text{FATHER}(\text{joe}, \text{bill}) \}$$

Where the substitution β = {sue | x, joe | y, bill | z} is used, results in the unit clause ¬ MOTHER (sue, bill).

3) Linear Resolution - When each resolved clause C_i is a parent to the clause C_{i+1} (i = 1, 2, ..., n – 1) the process is called linear resolution. For example, given a set S of clauses with $C_0 \subseteq S$, C_n is derived by a sequence of resolutions, C_0 with some clause B_0 to get C_1, then C_1 with some clause B_1 to get C_2 and so on until C_n has been derived.

4) Linear Input Resolution - If one of the parents in linear resolution is always from the original set of clauses (the Bi), then it is a linear input resolution. For example, given the set of clauses.

$S = \{ P \vee Q, \neg P \vee Q, P \vee \neg Q, \neg P \vee \neg Q \}$ let $C_0 = (P \vee Q)$. Choosing $B_0 = \neg P \vee Q$ from the set S and resolving this with C_0 we obtain the resolvent $Q = C_1$. B_1 must now be chosen from S and the resolvent of C_1 and B_1 becomes C_2 and so on.

5) Set-of-support resolution - Let S be an unsatisfiable set of clauses and T be a subset of S. Then T is a set-of-support for S if S-T is satisfiable. A set-of-support resolution is a resolution of two clauses not both from S – T.

This essentially means that given an unsatisfiable set $\{A_1, \ldots\ldots, A_K\}$, resolution should not be performed directly among the A_i.

Steps taken to speed up resolution

When the choice of the clauses to resolve together at each step is made in certain systematic ways, then the resolution procedure will find a contradiction if one exists. However, it may take a very long time. Strategies exist there so as to speed up the process considerably, as given below

1) Resolve only those pairs of clauses that contain complementary literals, since only such resolutions produce new clauses that are harder to satisfy than their parents.

2) Eliminate certain clauses as soon as they are generated so that they cannot participate in later resolutions, such as tautologies (which can never be unsatisfied) and that are subsumed by other clauses (i.e. they are easier to satisfy. For example, $P \vee Q$ is subsumed by P).

3) Whenever possible, resolve either with one of the clauses that is part of the statement which is one trying to refute or with a clause generated by a resolution with such a clause. This is called **set-of-support** strategy.

4) Whenever possible, resolve with clauses that have a single literal. Such resolutions generate new clauses with fewer literals than the larger of their parent clauses and thus are probably closer to the goal of a resolvent with zero terms. This method is called the unit-preference strategy.

For example consider following set of statements -

i) Marcus was a man.

ii) Marcus was a pompeian.

iii) All pompeians were Romans.

iv) Caesar was a Ruler.

v) All Romans were either loyal to Caesar or hated him.

vi) Everyone is loyal to someone.

vii) People only try to assassinate rulers they are not loyal to.

viii) Marcus tried to assassinate Caesar.

- Now, prove hate (Marcus, Caesar) i.e., Marcus hate Caesar.
- For this, first of all convert these facts into clause form as shown below

Axioms in Clause form

1. man (Marcus)
2. Pompeian (Marcus)
3. $\neg$ Pompeian $(x_1) \vee$ Roman (x_1)
4. ruler (Caesar)
5. $\neg$ Roman $(x_2) \vee$ loyalto $(x_2,$ Caesar$) \vee$ hate$(x_2,$ Caesar)
6. loyalto$(x_3,$ f1$(x_3))$
7. $\neg$ man$(x_4) \wedge \neg$ ruler $(y_1) \vee \neg$ tryassassinate $(x_4, y_1) \vee$ loyal to (x_4, y_1)
8. tryassassinate (Marcus, Caesar)

- For proving hate (Marcus, Caesar), negate the clause having

  $\neg$ hate (Marcus, Caesar) as the resulting clause.
- Now try to prove using steps given above. This procedure is shown in Fig. 6.4.16.

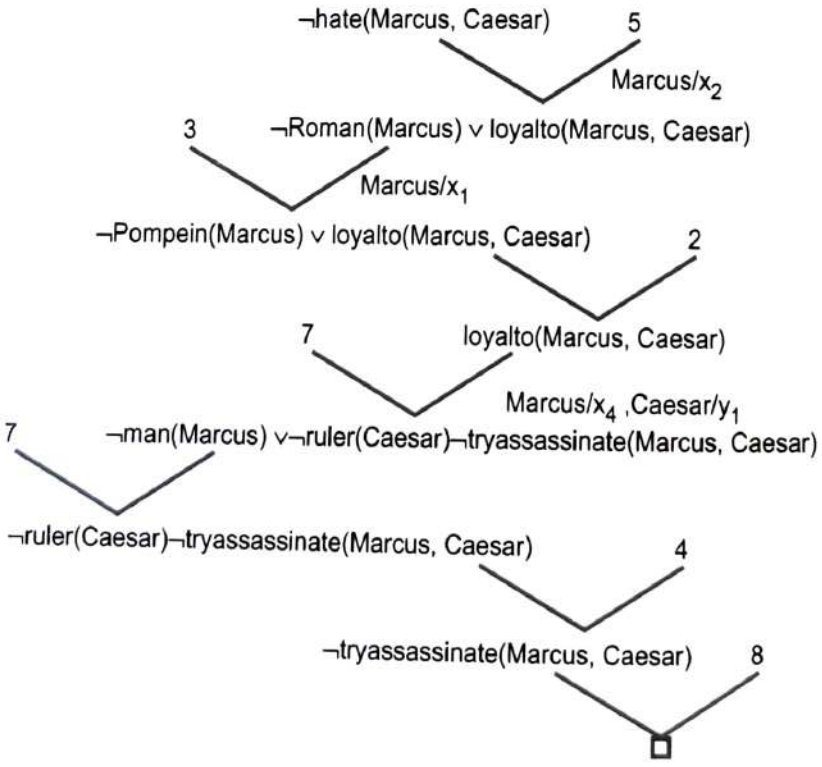


Fig. 6.4.16 A resolution proof hate (Marcus, Caesar)

- Start this proof with clause - hate (Marcus, Caesar) and consider, clause 5 as the second parent, telling x_2 = Marcus, the resolvent clause generated is

  ¬ Roman (Marcus) ∨ loyal to (Marcus, Caesar).

- Now, consider another contradictory clause i.e. clause 3 as parent with it and generate a resolvent clause.

  ¬ pompeian (Marcus) ∨ loyalto (Marcus, Caesar) telling Marcus = x_1.

- Further, clause 2 is considered and resulting clause is

  loyalto (Marcus, Caesar).

- Further, take 7^{th} clause as parent and resulting resolvent is

  ¬ Man (Marcus) ∨ ¬ Ruler (Caesar) ∨ ¬ tryassassinate (Marcus, Caesar)

  with putting X_4 = Marcus and y_1 = Caesar.

- Further, clause 1 is taken that generates resulting clause

  ¬ Ruler (Caesar) ∨ ¬ tryassassinate (Marcus, Caesar).

- Next, 4^{th} clause is taken that generates clause

  ¬ tryassassinate (Marcus, Caesar).

  Which with clause 8 result into an empty clause.

- Empty clause means that the clause which is initially taken totally contradicts and cannot be true any how. Which result that

  ¬ Hate (Marcus, Caesar) is false and its contradict

  i.e. Hate (Marcus, Caesar) is true.

6.4.11 Dealing with Equality

The unification algorithm described in the previous section will not unify pairs of sentences such as

i) isPriminister (M_Singh) and

ii) isPriminister (Manmohan_Singh), and rightly so, because syntactically, the constants M_Singh and Manmohan_Singh are quite different. We would not want our theorem proving agent to go around making assumptions about the equality of constants just based on their names, as this would make the system unsound. Surely we would want it to be able to unify the two isPriminister predicates, as this will allows it to do more resolution steps.

One way to get around the problem with equality is to add lots of extra clauses to our knowledge base, expressing every thing we wanted to say about equality. **These are known as equality axioms.** In particular, we would have to say that it is reflexive, symmetric and transitive :

$\forall x\ (x = x)$ [Reflexive]

$\forall x, y\ (x = y \rightarrow y = x)$ [Symmetric]

$\forall x, y, z\ (x = y \wedge y = z \rightarrow x = z)$ [Transitive]

This is not enough to enable the system to realize when it can unify two constants. We would have to put in additional statements about equality for each of the predicates in our knowledge base. So for example, if the predicate P of arity 1 was mentioned in the knowledge base, then we would have to add the following to the knowledge base :

$\forall x, y\ (x = y \rightarrow P(x) = P(y))$.

This kind of thing will have to be done for every predicate and function, if we are to be sure not to miss any opportunity for unification.

An alternative way to get a proving agent to use equality to its full is to employ **another inference rule called demodulation**. Like the resolution rule, this takes two sentences as input, but one of them must be an equality sentence relating two terms. Then, if, in the other sentence, there is a term T which can be unified with the left hand substituted for T. It's more obvious when written as following rule :

$$\frac{X = Y,\ A\ [T]}{SUBST\ (\theta, A\ [y])...)}\ Unify\ (x, s) = \theta$$

We can also allow demodulation to work the other way, i.e. replace y with x, but as with rewritting we need to take care not to get stuck in any loops. Our presidential example would be a trivial case for the demodulation rule : -

$$\frac{Manmohan\_Singh = M\_Singh,\quad isPriminister\ (Manmohan\_\ Singh)}{isPriminister\ (M\_\ Singh)}$$

In this case, the unifying substitution is just the trivial identify substitution.

Most state of the art resolution theorem provers have resolution at their heart, but use a number of other inference rules, including demodulation, and its big brother **paramodulation**, which deals with cases where we only know that

$x = y \vee P(x)$

- Induction is done over many different structures.
- It allows reasoning about recursion or iteration.
  [It is useful for hardware or software verification].

6.4.12 Theorem Provers (Automated Reasoners)

1) They accept full first order logic sentences.
2) In most theorem provers, the synthetic form chosen for the sentences does not affect the result.

3) Application areas : Verification of software and hardware

4) In mathematics theorem provers have a high standing. Now a days they have come up with novel mathematical results.

5) For example, in 1996 a version of well known OTTER was the first to prove (eight days of computation) that the axioms proposed by Herbert Robbins in 1933 really define Boolean algebra.

6) Following are various types of Theorem provers.

i. Inductive Theorem Proving :

- Deduction is done by mathematical inducation.

$$\frac{P(o)\ \forall x \cdot (P(x) \rightarrow P(x+1))}{\forall x \cdot P(x)}$$

ii. Interactive theorem proving :

- In this theorem proving it is necessary to interact with humans in order to prove theorems of any difficulty.
- These are like Mathematician's assistant. Let a theorem prover do simple tasks while Mathematician develop a theory (example : Buchberger's Theorem)
- It is a Guided theorem proving. Users follows and guides computer proof attempt.
- It needs visualization tools for proof trees.

iii. Higher order theorem proving

- It provides deduction in higher order logics.
- It allows more natural and succinct statements.
- Higher order theorem prover are used for verification tasks [Example - Verification of cryptographic protocols].
- It uses induction and interactive control.

6.5 Monotonic and Non-Monotonic Reasoning

GTU : Winter-18,19, Summer-18,19,20

There are various reasoning approaches used to solve problems posed by uncertain, fuzzy and often changing knowledge in the problem world.

6.5.1 Monotonic Reasoning

1) It is the reasoning in which the axioms and for the rules of interference are extended to make it possible to reason with consistent and complete information.
2) The only way it can change is that the new facts can be added as they become available.

3) If these new facts are consistent with all the other facts that have already been asserted, then nothing will ever be retracted from the set of facts that are known to be true. This property is called monotonicity, and a reasoning based on this property is known as Monotonic reasoning.

4) It works with information that is consistent and complete with respect to the domain of interest.

5) In other words, all the facts that are necessary to solve a problem are present in the system or can be derived from those that are by conventional rules of first-order logic.

6) It is a method of reasoning under consistent, complete, unchanging and certain facts. It was implicity assumed that a sufficient amount of reliable knowledge was available with which to deduce confident conclusions.

7) Thus a monotonic reasoning system cannot work effectively in real life environments because -

   i) Information available is always incomplete.

   ii) As process goes by, situations change and so are the solutions.

   iii) Default assumptions are made in order to reduce the search time and for quick arrival of solutions.

8) Logic based systems are monotonic in nature, i.e., if a proposition is made which is true, it remains true under all circumstances. But in real life, all statements made do not necessarily mean that they are correct under all circumstances.

9) Whenever one make a statement, one do not make it in a ad hoc fashion. The statement is made by manipulating a set of beliefs. Experts predict, diagnose and perform majority of their mental activity by relying on their beliefs. It is possible that during the course of action, events may take place which can either enhance the beliefs or reduce the dependency on the beliefs already existing.

10) Monotonic reasoning -

- Advantages -
  1) In monotonic reasoning new axioms are asserted, new wff's may become provable, but no old proofs ever become invalid.
  2) It gives valid deduction which remains so always.
- Drawbacks -
  1) It is not possible to describe many envisioned or real world concepts; that is, it is limited in expressive power.
  2) There is no way to express uncertain, imprecise, hypothetical or vague knowledge, only the truth or falsity of such statements.

3) Available inference methods are known to be inefficient.

4) There is no way to produce new knowledge about world. It is only possible to add what is derivable from the axioms and theorems in the knowledge base.

6.5.2 Nonmonotonic Reasoning

1) It is the reasoning in which the axioms and/or the rules of inference are extended to make it possible to reason with incomplete information.

2) These system preserve, however, the property that, at any given moment, a statement is either believed to be true, believed to be false, or not believed to be either.

3) AI systems provide solutions for those problems whose facts and rules of inference are explicitly stored in the database and knowledge base. But data and knowledge are incomplete in nature and generally default assumptions are made.

4) For example, if we say that "Somu is a bird" the conclusion the arrive (by default) is that Somu can fly. But it is not necessary that Somu can fly. May be it cannot fly because of many reasons such as -

   i) Somu could be an Ostrich.

   ii) Somu's wings are broken.

   iii) Somu is too weak to fly.

   iv) Somu could be caged.

   v) Somu could be dead bird etc.

5) So, if one says that "Somu is a bird" by default conclusion is that Somu can fly. But if one adds a statement that "Somu is a Ostrich" then this added statement retract the previous made assumption. This is the basic logic behind nonmonotonic reasoning.

6) It means that new facts become known which contradict and invalidate old knowledge. The old knowledge was retracted causing other dependent knowledge to become invalid, thereby requiring further retractions. The retractions led to a shrinkage or nonmonotonic growth in the knowledge at times.

7) Non-monotonic reasoning -

- Advantages -

   1) In this logic, axioms and/or rules of inference are extended to make it possible to reason with incomplete information.

2) It can express uncertain, imprecise, hypothetical or vague knowledge. It preserves, however the property that at any instant, a statement is either true or false or not believed to be either.

- Drawbacks -
  1) Adding a new assertion may invalidate the old inference that depended on the absence of that assertion.
  2) The old knowledge is retracted causing other dependent knowledge to become invalid.
- Comparison (Monotonic and Non-Monotonic Reasoning) -

The conclusions derived using monotonic logics are valid deductions and they remain so. Adding new axioms increases the amount of knowledge contained in the knowledge base. Therefore, the set of facts and inferences in such systems can only grow larger, they cannot be reduced, that is they increase monotonically.

Whereas, in non-monotonic logic, new facts became known which contradicted and invalidated old knowledge. The old knowledge was retracted causing other dependent knowledge to become invalid, thereby requiring further retractions. The retractions led to a shrinkage or non-monotonic growth in the knowledge.

Solved Examples

Example 6.1 *Convert the following sentences to predicate logic -*

i) Marcus was a man ii) Marcus was a pompeian iii) All pompeians were Romans iv) Caesar was a ruler v) All romans were either loyal to caesar or hated him vi) Everyone is loyal to someone vii) People only try to assassinate rulers they are not loyal to viii) Marcus tried to assassinate caesar.

Solution : Set of wff's in predicate logic are as follows -

i) Marcus was a Man - Man (Marcus)

The representation captures the critical fact of Marcus being a man, but if fails to capture some of the information in the english sentence, namely the notion of past tense, i.e., one cannot conclude from above representation that a Marcus was a man, or Marcus is a man, or Marcus will be a man etc. Whether this omission is acceptable or not depends on the use to which we intend to put the knowledge. For this example, it will be all right.

ii) Marcus was a Pompeian

Pompeian (Marcus) - Its explanation is same as of first one.

iii) All pompeians were Romans

$\forall x$: Pompeian (x) $\rightarrow$ Roman (x)

In this presentation universal quantifier $\forall$ is used to present universal value of x. i.e., for all x in universe.

Pompeian (x) indicates that x is a pompeian. Next symbol '$\rightarrow$' is implies i.e., means and roman (x) indicates that x is Roman.

In other words we say that all x i.e., all pompeians were Roman.

iv) Caesar was a ruler

ruler (Caesar)

Here, one can ignore the fact that proper names are often not references to unique individuals, since many people share the same name. Sometimes deciding which of several people of the same name is being referred to in a particular statement may require a fair amount of knowledge and reasoning.

v) "All romans were either loyal to Caesar or hated him"

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

In English, the word "OR" sometimes means the logical inclusive-OR and sometimes means the logical exclusive-OR (XOR).

Here, inclusive interpretation is used.

But, if one considers it as 'exclusive-OR' then presentation would be some what different as follows -

$\forall x$: Roman (x) $\rightarrow$ [loyalto (x, Caesar) $\vee$

hate (x, Caesar)) $\wedge$

$\neg$ (loyalto (x, Caesar) $\wedge$ hate (x, Caesar))]

vi) "Everyone is loyal to someone"

$\forall x : \exists y$: loyalto (x, y).

Here, one can use both the quantifiers i.e. $\forall \rightarrow$ universal quantifier and $\exists \rightarrow$ existential quantifier.

So, one can read above presentation as for all x, there exists a y such that x is loyal to y or indirectly we say

"Everyone is loyal to someone".

But here one can understand a different meaning i.e. there is someone to whom everyone is loyal, which would be written as

$\exists y : \forall x$: loyalto (x, y)

Now read it as, there exists a y, such that for all x, x is loyal to y.

Or

"everyone is loyal to y".

vii) "People only try to assassinate rulers they are not loyal to"

$\forall x : \forall y$: person (x) $\wedge$ ruler (y) $\wedge$

tryassassinate (x, y) $\rightarrow \neg$ loyal to (x, y)

This sentence, too, is ambiguous. Does it mean that the only rulers that people try to assassinate are those to whom they are not loyal, one above presented or does it mean that the only thing people try to do is to assassinate rulers to whom they are not loyal ?

viii) "Marcus tried to assassinate Caesar"

tryassassinate (Marcus, Caesar)

Example 6.2 *Convert the following sentences to predicate logic -*

i) Marcus was a man ii) Alive means not dead iii) If x divides y and y divides z then x divides z.

Solution : i) Refer to example 6.1.

ii) Alive means not dead

$\forall x : \forall t$: [alive (x, t) $\rightarrow \neg$ dead (x, t)]

$\wedge$ [$\neg$ dead (x, t) $\rightarrow$ alive (x, t)]

This is not strictly correct, since $\neg$ dead implies alive only for animate objects.

iii) If x divides y and y divides z then x divides z.

$\exists x : \exists y : \exists z$: [divides (x, y) $\wedge$ divides (y, z)] $\rightarrow$ divides (x, z)

Example 6.3 *Suppose you are given some facts about Marcus as follows -*

i) Marcus was a man. ii) Marcus was pompeian. iii) Marcus was born in 40 A.D. iv) All men are mortal. v) All pompeians died when the volcano erupted in 79 A.D. vi) No mortal lives longer than 150 years. vii) It is now 1991.

Now, answer the question "Is Marcus alive"? By proving.

Solution : Representing facts in predicate logic :

i) Marcus was a man.

man (Marcus)

Again we ignore the issue of tense.

ii) Marcus was a pompeian.

Pompeian (Marcus).

iii) Marcus was born in 40 A.D.

born (Marcus, 40).

iv) All men are mortal.

$\forall x$: man (x) $\rightarrow$ mortal (x)

v) All pompeians died when the volcano erupted in 79 A.D.

erupted (volcano, 79) $\wedge \forall x$: [Pompeian (x) $\rightarrow$ died (x, 79)]

vi) No mortal lives longer than 150 years.

$\forall x : \forall t_1 : \forall t_2$: mortal (x) $\wedge$ born (x, t_1) $\wedge$ gt(lt_2–t_1, 150) $\rightarrow$ dead (x, t_2)

There are various ways that the content of this sentence could be expressed. For example, one could introduce a function age and assert that its value is never greater than 150.

vii) It is now 1991.

now = 1991.

Here, exploit the idea of equal quantities that can be substituted for each other.

By using above facts we try to answer that whether "Marcus is alive or dead".

This can be done in two ways.

a) Prove that Marcus is dead because he was killed by the volcano.

b) Show that if Marcus is alive today, then it must of age more than 150 years, which is not possible.

To prove this additional knowledge is required.

For example.

viii) Alive means not dead.

$\forall x : \forall t$: [alive (x, t) $\rightarrow \neg$ dead (x, t)]

$\wedge$ [$\neg$ dead (x, t) $\rightarrow$ alive (x, t)]

ix) If someone dies, then he is dead at all later times.

$\forall x : \forall t_1 : \forall t_2$: died (x, t_1) $\wedge$ gt(t_2, t_1) $\rightarrow$ dead (x, t_2).

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.

To answer that requires breaking time up into smaller units than years.

If this is done then one can add rules that say such things as "one is dead at the (year1, month1) if one died during (year1, month2) and month 2 precedes month 1". One can extend this to days and hours etc.

Fig. 6.1 and 6.2 shows these proofs by different ways.

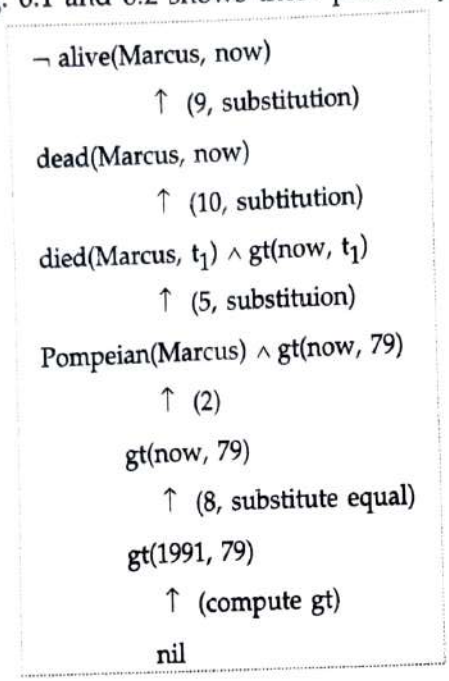


Fig. 6.1 One way of proving that Marcus is dead

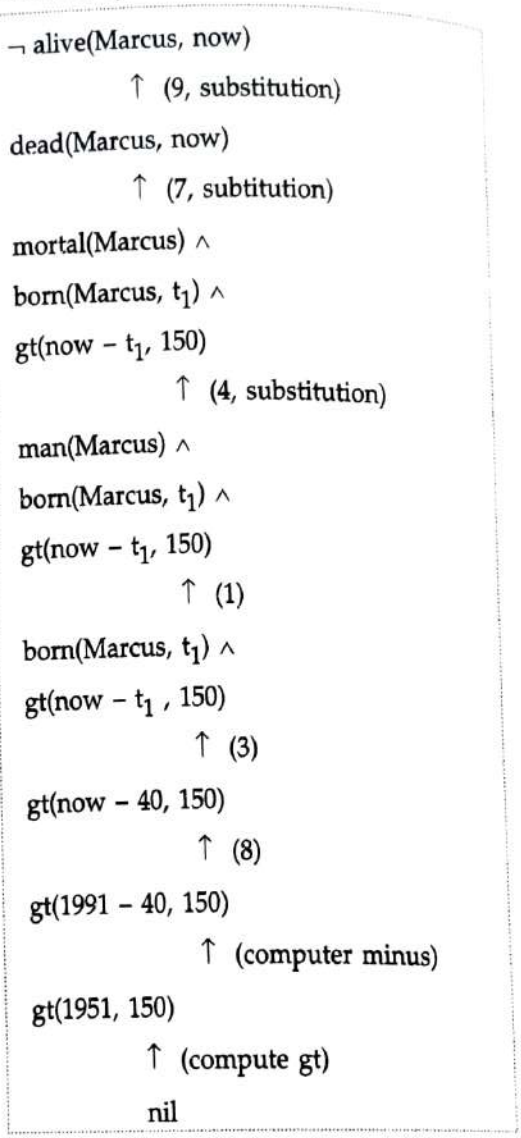


Fig. 6.2 Another way of proving that Marcus is dead

- From looking at the proofs just shown, two things should be clear -
  a) Every very simple conclusions can require many steps to prove.
  b) A variety of processes, such as matching, substitution and application of modus ponens are involved in the production of a proof. This is true even for the simple statements we are using.

Example 6.4 *Consider the following set of sentences -*

i) Marcus was a man ii) Marcus was a pompeian iii) All pompeians were Romans iv) Caesar was a ruler v) All romans were either loyalto caesar or hated him vi) Everyone is loyal to someone vii) People only try to assassinate rulers they are not loyal to viii) Marcus tried to assassinate caesar.

Prove that was Marcus loyal to caesar.

Solution : All the given sentences in predicate logic are given in example 6.1.

Now, by using the above facts one need to answer that, was marcus loyal to caesar or not.

Let's try to produce a formal proof, reasoning backward from the desired goal -

¬ loyal to (Marcus, Caesar)

In order to prove the goal, one need to use the rules of inference to transform it into another goal that can in turn be transformed and so on, until there are no satisfied goals remaining.

But there is a problem, that is, although we know that Marcus was a man, one do not have any way to conclude that Marcus was a person. So one add this fact.

ix) Marcus was a person

$\forall x$: man (x) $\rightarrow$ person (x)

Now one can satisfy the goal and produce proof that Marcus was not loyal to Caesar (as seen from rule (vii) and (viii))

Proof -

¬ loyalto (Marcus, Caesar)

↑ (7, substitution)

person (Marcus) ∧

ruler (Caesar) ∧

tryassassinate (Marcus, Caesar)

↑ (4)

person (Marcus)

tryassassinate (Marcus, Caesar)

↑ (8)

person (Marcus)

Example 6.5 *Consider the following sentences :*

i) If X is on top of y then y supports X

ii) If X is above y and they are touching each other then x is on top of y.

iii) A cup is above a book.

iv) A cup is touching a book

Transtate these statements in predicate logic. Prove by backward reasoning "Book Support Cup".

Solution : Conversion of above statements in predicate logic as follows -

i) $\exists x : \exists y$: on-top-of (x, y) $\rightarrow$ support (y, x) ... (i)

ii) $\exists x : \exists y$: [above (x, y) $\wedge$ touch (x, y) $\wedge$ touch (y, x)] $\rightarrow$ on-top-of (x, y) ... (ii)

iii) $\exists x : \exists y$: above (cup, book) ... (iii)

iv) $\exists x : \exists y$: touch (cup, book) ... (iv)

Proof by backward reasoning is as follows -

It predicate logic conversion will be -

Support (Book, cup)

$\uparrow$ (1, substitution)

Ontopof (Cup, Book)

$\uparrow$ (2, substitution)

above (Cup, Book) $\wedge$ touch (Cup, Book) $\wedge$ touch (Book, Cup)

(3, substitution)

touch (Cup, Book) $\wedge$ touch (Book, Cup)

$\uparrow$ (4 substitution)

Nill

The proof of given statement is as above. The term Nil at the end of proof indicate that the list of conditions remaining to be proved is empty and so the proof has succeeded.

Example 6.6 *Convert the following statement into clause form*

i) $(\forall x)\ (\exists y)\ (\forall z)\ [p(x, y) \wedge q(x, z) \rightarrow r(z, x)]$

ii) $(\forall x)\ (\forall y)\ [(\exists z)\ p(x, z) \wedge p(y, z)) \rightarrow (\exists u)\ Q(x, y, u)]$

Solution : i) Given statement is

$$(\forall x)\ (\exists y)\ (\forall z)\ [p(x, y) \wedge q(x, z) \rightarrow r(z, x)]$$

By applying rule 1,

$$(\forall x(\ (\exists y)\ (\vee z)\ [\sim(p(x, y) \wedge q(x, y)) \vee r(z, x)]$$

By applying rule 2,

$$(\forall x)\ (\exists y)\ (\forall z)\ [\sim p(x, y) \vee \sim q(x, z) \vee r(z, x)]$$

Rule 3 and 4 is not required here, so by applying rule 5,

$$(\forall x)\ (\forall z)\ [\sim p\ (x, A(x)) \vee \sim q(x, z) \vee r(z, x)]$$

By applying rule 6,

$$[\sim p(x, A(x)) \vee \sim q(x, z) \vee r(z, x)]$$

By applying rule 7,

$$\sim p(x, A(x)) \vee \sim q(x, z) \vee r(z, x)$$

The above expression is a clausal form.

ii) Given statement is

$$(\forall x)\ (\forall y)\ [((\exists z)\ p(x, z) \wedge p(y, z)) \rightarrow (\exists u)\ Q(x, y, u)]$$

By applying rule 1,

$$(\forall x)\ (\forall y)\ [\sim((\exists z)\ p(x, z) \wedge p(y, z)) \vee (\exists u)\ Q(x, y, u)]$$

By applying rule 2,

$$(\forall x)\ (\forall y)\ [((\forall z)\ \sim p(x, z) \vee \sim p(y, z)) \vee (\exists u)\ Q(x, y, u)]$$

By applying rule 4,

$$(\forall x)\ (\forall y)\ (\forall z)\ (\exists u)\ [((\sim p(x, z) \vee \sim p(y, z)) \vee (\exists u)\ Q(x, y, u)]$$

By applying rule 5,

$$(\forall x)\ (\forall y)\ [((\sim p(x, A(x, y)) \vee \sim p(y, A(x, y))) \vee Q(x, A(x, y), B(x, y))]$$

By applying rule 6,

$$[((\sim p(x, A(x, y)) \vee \sim p(y, A(x, y))) \vee Q(x, A(x, y), B(x, y))]$$

By applying rule 7,

$$\sim p(x, A(x, y)) \vee \sim p(y, A(x, y) \vee Q(x, A(x, y), B(x, y))$$

The above expression is a clausal form.

Example 6.7 *Explain the following terms in common sense -*

i) Ribbon (Object, Side$_1$, Side$_2$) ii) Along (x, y) iii) Across (x, y).

Solution : i) Ribbon (Object, Side$_1$, Side$_2$) -

A ribbon is essentially a curve viewed at a coarser level of granularity, resulting in a two-dimensional ribbon like shape.

The problem world contains many objects that are usefully viewed as ribbons, e.g., rivers and bridges (Fig. 6.3).

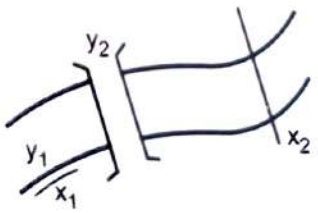


Fig. 6.3 Two ribbons (y_1 and y_2) and Two curves (x_1 and x_2)

$$\text{TERMINAL}(P, c) \equiv$$
$$\text{INSIDE}(P, c) \wedge$$
$$\forall c_1, c_2 : \text{INSIDE}(c_1, c) \wedge \text{INSIDE}(c_2, c)$$
$$\wedge \text{INSIDE}(P, c_1) \wedge \text{INSIDE}(p, c_2)$$
$$\rightarrow \text{INSIDE}(c_1, c_2) \vee \text{INSIDE}(c_2, c_1)$$

It is the definition of a terminal point P of a curve c.

ii) Along (x, y) -

$$\text{ALONG}(x, y) \equiv \text{CURVE}(x) \wedge \text{RIBBON}(y, s_1, s_2) \wedge$$
$$\forall_z : \text{INSIDE}(z, x) \rightarrow \text{ADJACENT}(z, y)$$

iii) Across (x, y) -

$$\text{ACROSS}(x, y) \equiv \text{CURVE}(x) \wedge \text{RIBBON}(y, s_1, s_2) \wedge$$
$$\text{PERPENDICULAR}(x, \text{AXIS}(y, s_1, s_2)) \wedge$$
$$\text{ADJACENT}(x, y) \wedge \text{ADJACENT}(x, s_1) \wedge$$
$$\text{ADJACENT}(x, s_2)$$

These definitions assume that the terms PERPENDICULAR, AXIS and ADJACENT have been defined and that the commonsense axiom that an object x is ADJACENT to an object y if any part of x is ADJACENT to y, are supplied.

A robot could use the ALONG relation to plot a course down the river's edge. It could similarly use the ACROSS relation to navigate to the other side of the river. Unfortunately, the ACROSS relation is not enough, as the robot might try to cross the river without using the bridge.

Example 6.8 *Consider the following sentences -*

a) Krishna like all kind of food. b) Apples are food. c) Bread is food. d) Anything anyone eat and is not killed by is food. e) Ravi eats peanuts and is still alive. f) Chetan eat everything Ravi eats. g) Translate these sentences into formula in predicate logic. h) Prove that Krishna likes peanuts using resolution.

Solution : i) Translation in Predicate logic -

a) Krishna like all kind of food.

$\forall x$: food (x) $\rightarrow$ like (Krishna, x)

b) Apples are food.

food (Apple)

c) Bread is food.

food (Bread)

d) Anything everyone eat and is not killed by is food

$\forall x : \forall y$: eat (x, y) $\wedge \sim$ kill (x, y) $\rightarrow$ food (y)

e) Ravi eats peanuts and is still alive

eat (Ravi, peanut) $\wedge$ alive (Ravi)

f) Chetan eat everything Ravi eats

$\forall x$: eat (Chetan, x) $\rightarrow$ eat (Ravi, x)

ii) Proof by resolution - Need to prove that like (Krishna, peanut)

- Assume the negation of the result

  $\sim$ like (Krishna, peanut) ... (i)

- The given axioms are

  $\forall x$: food(x) $\rightarrow$ like(Krishna, x) ... (ii)

  food(Apple) ... (iii)

  food(Bread) ... (iv)

  $\forall x : \forall y$: eat(x, y) $\wedge \sim$ kill(x, y) $\rightarrow$ food(y) ... (v)

  eat(Ravi, peanut) $\wedge$ alive (Ravi) ... (vi)

  $\forall x$: eat(Chetan, x) $\rightarrow$ eat(Ravi, x) ... (vii)

- Equation (ii) can be written as

  $\sim$ food(x) $\vee$ like(Krishna, x) ... (viii)

- In equation (viii), substitute x = peanut

  $\sim$ food(peanut) $\vee$ like(Krishna, peanut) ... (ix)

- Resolving equations (i) and (ix), we have

  $\sim$ food(peanut) ... (x)

- Resolving equations (iii) and (x) contradiction arrives. Hence, the negation of the result is false or the result is true.

Example 6.9 *Convert the following sentences to predicate logic, clause form and thus prove by resolution that - "Marcus was a Roman"*

i) Marcus was a Pompeian. ii) All Pompeian were Romans.

Solution : In Predicate logic -

i) Marcus was a Pompeian.

Pompeian (Marcus)

ii) All pompeians were Romans.

$\forall x$: Pompeian (x) $\rightarrow$ Roman (x)

In clause form -

i) Pompeian (Marcus).

ii) $\neg$ Pompeian (x_1) $\vee$ Roman (x_1)

To prove -

Roman (marcus)

Proof -

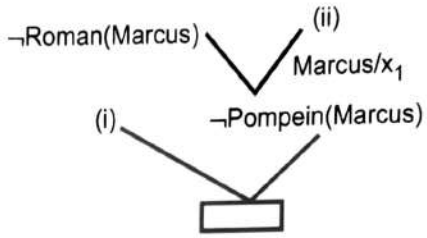


Fig. 6.4

Empty clause means that the clause which initially taken is totally contradict and cannot be true, which results that

$\neg$ Roman (marcus) is FALSE or

Roman (marcus) is TRUE. **Hence proved.**

Example 6.10 *Convert the following sentences into propositional form :*

Given :

Sentence	*Symbol*
If rains	*r*
Picnic is cancelled	*pc*
be wet	*wet*
stay at home	*s*

i) I will be wet if it rains

ii) If it rains but I stay at home I won't be wet

iii) If it rains and the picnic is not cancelled or I don't stay at home, I will be wet.

iv) Either it does not rain or I am staying at home

v) Whether or not the picnic is cancelled, I am staying at home, if it rains

Solution : 1. Propositional form :

i) $r \rightarrow wet$

ii) $(r \vee s) \rightarrow wet$

iii) $((r \wedge \neg pc) \vee \neg s) \rightarrow wet$

iv) $\neg r \vee s$

v) $(pc \vee \neg pc) \wedge r \rightarrow s$

Example 6.11 *Convert the following sentences into FOL representation*

a) Some students took English in spring 2004

b) Every student who takes English passes it

c) John is a man

d) Father of Bill is a teacher

e) John likes books and music

f) John lives in a yellow house

g) If the car belongs to John then it is green

h) All elephants are gray in color

i) John likes all kind of foods

j) Anything any one eats and isn't killed by is food.

k) Sue eats everything Bill eats

l) Not all student take both History and Biology

m) Only one student failed in History

Solution :

a) $\exists x$ Student (x) $\wedge$ Take (x, English, Spring2004)

b) $\forall x$ Student (x) $\wedge$ Take (x, English)=>Passes (x, English)

c) Man (John)

d) Teacher (Father of (Bill))

e) Likes (John, Books) $\wedge$ Likes (John, Music)

f) Lives (John, House) $\wedge$ Color (House, Yellow)

(or)

Lives ((John, Color(House, Yellow))

g) Belongs (Car, John)=> Color(Car, Green)

h) $\forall x$ Elephant (x)=> Color (x, Gray)

i) $\forall x$ Likes (John, x) $\wedge$ Food(x)

j) $\forall x \forall y$ Person(x) $\wedge$ Food(y) $\wedge$ Eats(x, y) $\wedge \neg$ Killed(x)=>Isfood(x)

k) $\forall x$ Eats (Sue, x) = Eats (Bill, x)

l) $\exists x$ student (x) $\wedge$ Take(x, History) $\wedge$ Take(x, Biology)

m) $\exists x$ student (x) $\wedge$ Failed (x, History) $\wedge \forall y$ y $\neq$ x = $\neg$ Failed (y, History)

Example 6.12 *I. From the following KB, solve the given task using any one of the resolution techniques.*

1. Mani likes easy games 2. Boxing is hard 3. All the indoor games are easy

4. Table Tennis is an indoor game.

Task : Find the name of the game. which is liked by Mani.

II. From the following KB, solve the given task using any one of the resolution techniques for INF and CNF representation.

1. Jack owns a dog 2. Every dog owner is an animal lover 3. No animal lover kills an animal 4. Either Jack or Curiosity killed the cat, who is named Tuna 5. Tuna is a cat 6. All cats are animals

Task : Show that Kills (Curiosity, Tuna) is true.

III. For each pair of atomic sentences, give the most general unifiers if it exists.

a) P(A, B, B), P(x, y, z) b) Q(y, G(A, B)), Q(G(x, y), y)

Solution : I. FOL representation :

1. $\forall x$ Easygame (x) $\rightarrow$ Likes (Mani, Game (x))
2. Hard (Boxing)
3. $\forall x$ Indoorgame (x) $\rightarrow$ Easygame (x)
4. Indoorgame (TableTennis)

CNF representation :

1. ~Easygame (x) $\vee$ Likes (Mani, Game (x))
2. Hard (Boxing)
3. ~Indoorgame (x) $\vee$ Easygame (x)
4. Indoorgame (TableTennis)

Proof :

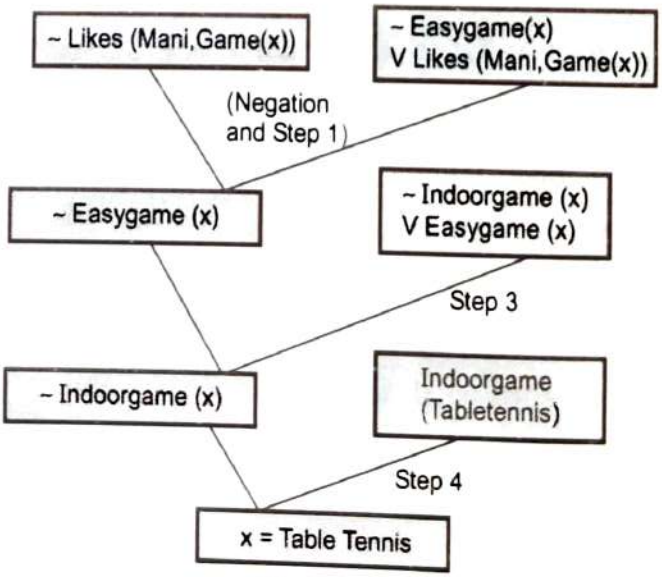


Fig. 6.5 (a)

$\therefore$ Mani like TableTennis

II. FOL representation :

1. $\exists x$ Dog (x) $\wedge$ Owns (Jack, x)
2. $\forall x$ ($\exists y$ Dog (y) $\wedge$ Owns (x, y)) $\Rightarrow$ Animallover (x)
3. $\forall x$ Animallover (x) $\Rightarrow$ $\forall y$ Animal (y) $\Rightarrow \neg$ Kills (x, y)
4. Kills (Jack, Tuna) $\vee$ Kills (Curiosity, Tuna)
5. Cat (Tuna)
6. $\forall x$ Cat (x) $\Rightarrow$ Animal (x)

INF representation :

1. a. Dog (D)

   b. Owns (Jack, D)
2. Dog (y) $\wedge$ ~Owns (x, y) $\Rightarrow$ Animallover (x)
3. Animallover (x) $\wedge$ Animal (y) $\wedge$ Kills (x, y) $\Rightarrow$ False
4. Kills (Jack, Tuna) $\vee$ Kills (Curiosity, Tuna)
5. Cat (Tuna)
6. Cat (x) $\Rightarrow$ Animal (x)

CNF representation :

1. a. Dog (D)

   b. Owns (Jack, D)
2. ~ Dog (y) $\vee$ ~ Owns (x, y) $\vee$ Animallover (x)
3. ~ Animallover (x) $\vee$ ~ Animal (y) $\vee$ ~ Kills (x, y)
4. Kills (Jack, Tuna) $\vee$ Kills (Curiosity, Tuna)
5. Cat (Tuna)
6. ~ Cat (x) $\vee$ Animal (x)

Resolution with refutation using INF representation :

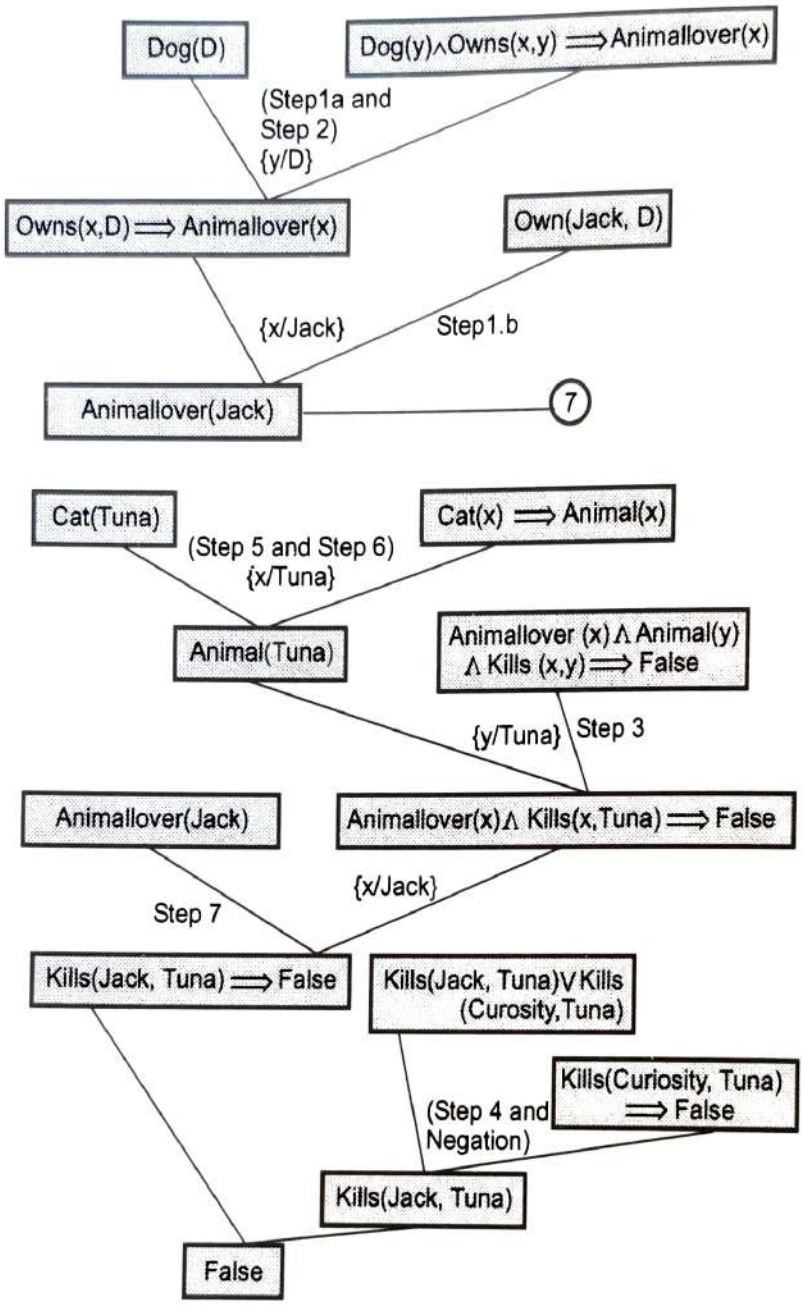


Fig. 6.5 (b)

To solve the given task we assumed the negation (Kills (Curiosity, Tuna) ⇒ False) and using inference rules we derive a contradiction, False, which means that the assumption must be false, and Kills (Curiosity, Tuna) is true.

Resolution with refutation using CNF representation :

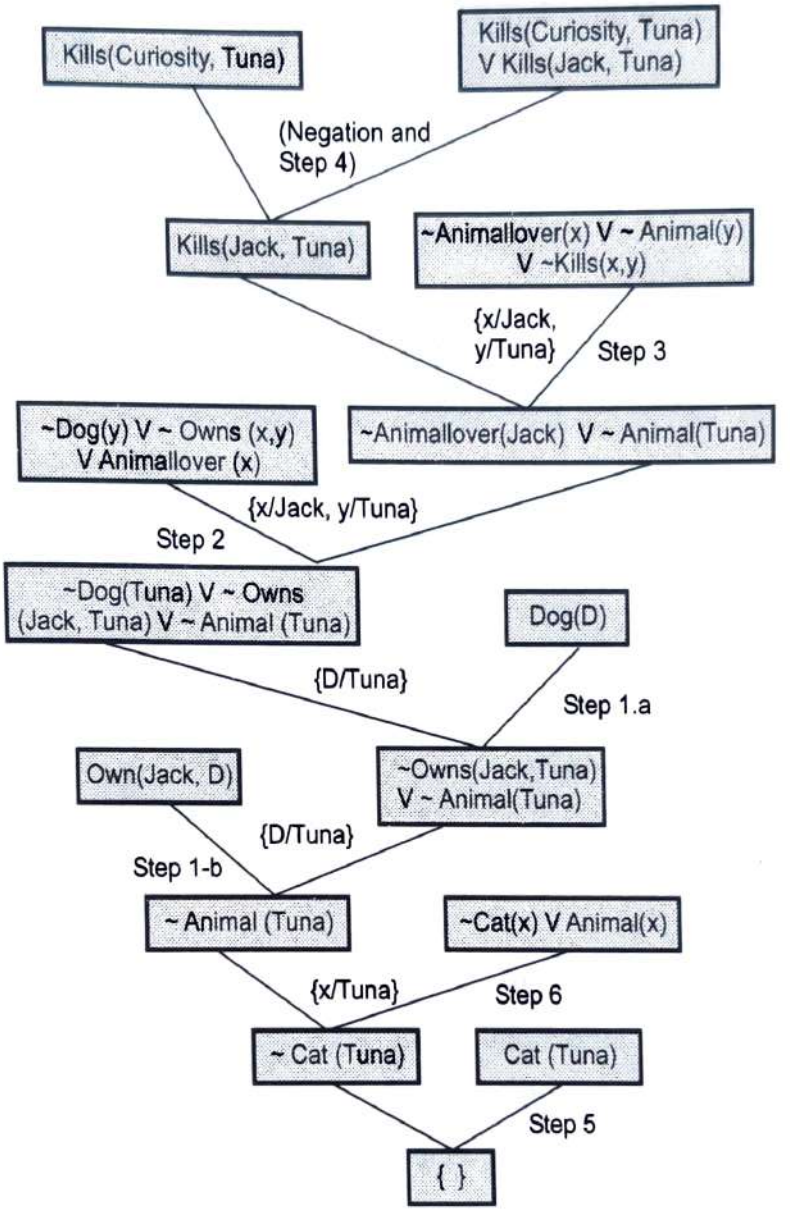


Fig. 6.5 (c)

To prove the given task, we assumed the negation of it (i.e.) ~ Kills (Curiosity, Tuna) and using inference rule we derive a contradiction as empty set, which means that the assumption must be false, and Kills (Curiosity, Tuna) is true.

III. a. {x | A, y | B, z | B}

b. No unifer (x cannot bind to both A and B)

Example 6.13 *Represent the following sentences in FOL :*

a) Perrier is a kind of water

b) John has perrier in his water bottle

c) Water is a liquid between 0 and 100 degrees

d) The water in John's water bottle is frozen.

Solution : a) Perrier $\subset$ Water

b) $\exists b \; \forall \; w \; w \in$ Water $\wedge \; b \in$ WaterBottles $\wedge$ Has (John, b, Now) $\wedge$ Inside (w, b, Now) $\Rightarrow w \in$ Perrier.

c) $\forall \; w \; w \in$ Water $\Rightarrow$ (Degree (0) < Temperature (w) < Degree (100)) $\Leftrightarrow w \in$ Liquid.

d) $\exists b \; \forall \; w \; w \in$ Water $\wedge \; b \in$ WaterBottles $\wedge$ Has (John, b, Now) $\wedge$ Inside (w, b, Now) $\Rightarrow$ (w $\in$ Solid, Now)

Example 6.14 *Consider the following facts :*

- *Steve only likes easy courses.*
- *Science courses are hard.*
- *All the courses in the HaveFun department are easy.*
- *BK301 is a HaveFun department course.*

Use resolution to answer the question "What course would Steve like"?

Solution : The following statements are assumed to be true :

1. Steve only likes easy courses.
2. Science courses are hard.
3. All the courses in the HaveFun department are easy.
4. BK301 is a HaveFun deparment course.

To answer : What course would Steve like ?

The predicate logic encoding of the premises of the previous problem is as follows :

1. forall x easy(x) -> likes(steve,x)
2. forall x science(x) -> ~easy(x)
3. forall x HaveFun(x) -> easy(x)
4. HaveFun(BK301)

The conclusion is encoded as

likes(steve,x).

First put premises in the clause form and the negation of conclusion to the set of clauses.

- (1) ~easy(x) or likes(steve,x)
- (2) ~science(x) or ~easy(x)

- (3) ~science(x) or ~easy(x)
- (4) ~HaveFun(x) or easy(x)
- (5) HaveFun(BK301)
- (6) ~likes(steve,x)

A resolution proof may be obtained by the following sequence of resolutions (each step includes a parenthesized number of the resolvent generated in the current step; 1 and 5 means that we resolve clauses (1) and (5)):

- (7) 1 and 6 yields resolvent ~easy(x).
- (8) 4 and 7 yields resolvent ~basketweaving(x).
- (9) 5 and 8 yields empty clause; the substitution x/BK301 is produced by the unification algorithm which says that the only wff of the form likes(steve,x) which follows from the premises is

  likes(steve,BK301).

Thus, resolution gives us a way to find additional assumptions (in this case x=BK301) which make theorem true.

Answer in Brief

1. *Convert the following sentences to predicate logic : i) Marcus was a man. ii) All men are mortal. iii) No mortal lives longer than 150 years.* **(Refer section 6.1.8)**
2. *Convert the following sentences to predicate logic : i) All Romans were either loyal to Caesar or hated him. ii) Everyone is loyal to someone. iii) Marcus tried to assassinate Caesar. iv) Caesar was ruler.* **(Refer section 6.18)**
3. *Give resolution in propositional logic.* **(Refer section 6.4)**
4. *Write down and explain the unification algorithm in predicate logic.*
   (Refer section 6.4)

   OR

   Explain unification algorithm.

   OR

   Write short note on unification algorithm.
5. *Differentiate the monotonic and non-monotonic reasoning.* **(Refer section 6.5)**
6. *Name two standard quantifier.*

Ans. : The two standard quantifier are universal quantifiers (represented as ∀, which means "For all") and existential quantifier (represented as ∃, which means "there exists some object").

They are used for expressing properties of entire collection of objects rather than just a single object.

For example :

i) $\forall$ x Happy (x) means that "if the universe of discourse is people, then everyone is happy".

ii) $\exists$ x Happy (x) means that "if the universe of discourse is people, then this means that there is at least one happy person".

7. *What is a purpose of unification ?*

Ans. : It is used for finding substitutions for inference rules, which can make different logical expression to look identical. It helps to match to logical expressions. Therefore it is used in many algorithm in first order logic.

8. *What is ontological commitment (what exists in the world) of first order logic ? Represent the sentence "Brothers are siblings" in first order logic.*

Ans. : Ontological commitment means what assumptions language makes about the nature if reality.

Representation of "Brothers are siblings" in first order logic is

$\forall$ x, y [Brother (x, y) $\Rightarrow$ Siblings (x, y)].

9. *Diffentiate between propositional versus first order predicate logic.*

OR Distinguish between predicate logic and propositional logic.

OR Diffentiate propositional and first order logic.

Ans. : Following are the comparative differences between propositional logic and first order logic

1) Propositional logic is less expressive and do not reflect individual object's properties explicitely.

   First order logic is more expressive and can represent individual object along with all its properties.

2) Propositional logic can not represent relationship among objects whereas first order logic can represent relationship.

3) Propositional logic do not consider generalization of objects where as first order logic handles generalization.

4) Propositional logic includes sentence letters (A, B, C) and logical connectives, but not quantifier.

   First order logic has the same connectives as propositional logic, but it also has variables for individual objects, quantifier, symbols for functions, and symbols for relations.

10. *What factors justify whether the reasoning is to be done in forward or backward reasoning ?*

Ans. : Following factors justify whether the reasoning is to be done in forward or backward reasoning -

1) Which state is more possible to begin with, the start state or goal state ?

2) Is there a need to justify the reasoning ?

3) What kind of events trigger the problem - solving ?

4) In which direction is the branching factor greatest ? One should go in the direction with lower branching factor.

11. Define diagnostic rules with example.

Ans. : Diagnostic rules are used in first order logic for inferencing. The diagnostic rules generate hidden causes from observed effect. They help to deduce hidden facts in the world. For example consider the wumpus world.

The diagnostic rule finding 'pit' is,

"If square is breezy some adjacent square must contain pit", which is written as,

$\forall s$ Breezy(s) $\Rightarrow \exists_r$ Adjacent (r,s) $\wedge$ pit(r)

12. Represent the following sentence in predicate form "All the children likes sweets".

Ans. : $\forall x$ child(x) $\wedge$ sweet(y) $\wedge$ likes (x,y).

13. What is skolemization ?

Ans. : Skolemization is the process of removing existential quantifier by elimination. It converts a sentence with existential quantifier into a sentence without existential quantifier such that the first sentence is satisfiable if and only if the second is.

For eliminating an existential quantifier, each occurance of its variable is replaced by a skolem function whose argument are the variables of universal quantifier whose scope includes the scope of the existential quantifier.

14. Define the first order definite clause.

Ans. : 1) They are disjunctions of literals of which exactly one is positive.

2) A definite clause is either atomic sentence or is an implication whose antecedants (left hand side clause) is a conjunction of positive literals and consequent (Right hand side clause) is a single positive literal.

For example :

Princess (x) $\wedge$ Beautiful (x) $\Rightarrow$ GoodHearted (x)

Princess (x)

Beautiful (x)

15. Write the generalized modus ponens rule.

OR State generalized modus ponens.

Ans. : 1) Modus ponens :

If the sentence **P** and **P -> Q** are known to be true, then modus ponens lets us infer **Q**.

For example : If we have statment, " If it is raining then the ground will be wet" and "It is raining". If P denotes " It is raining" and Q is "The ground is wet" then the first expression becomes $P \rightarrow Q$. Because it is indeed now raining (P is true), our set of axioms becomes,

$\{ P \rightarrow Q$

$P \}$

Through an application of modus ponens, the fact that "The ground is wet" (Q) may be added to the set of true expressions.

- **The generalized modus ponens :**

  For atomic sentences P_i, P_i', and q, where there is a substitution Q such that SUBST $(\theta, P_i') =$ SUBST (θ, P_i'),

  For all i,

$$\frac{P_1', P_2', \ldots P_n', (P_1 \wedge P_2 \wedge \ldots \wedge P_n \Rightarrow q)}{\text{SUBST}(\theta, q)}$$

  There are n+1 premises to this rule : - The 'n' atomic sentences p_i' and the one implication. The conclusion is the result applying the substitution θ to the consequent q.

Q.16 Define atomic sentence and complex sentence.

Ans. :

i. An atomic sentence is formed from a predicate symbol followed by a parenthesized list of terms.

   For example : Stepsister (Cindrella, Drizella)

ii. Atomic sentences can have complex terms as the arguments.

   For example : Married (Father (Cindrella), Mother(Drizella))

iii. Atomic sentences are also called atomic expressions, atoms or propositions.
   For example : Equal (plus (two, three), five) is an atomic sentence.

Complex sentences

i. Atomic sentences can be connected to each other to form complex sentence.

   Logical connectives, $\wedge$, $\vee$, $\neg$, $\rightarrow$ can be used to connect atomic sentences.

   For example :

   $\neg$ Princess (Drizella) $\rightarrow$ Princess (Cindrella)

ii. (foo (two, two, plus (two, three))) $\rightarrow$ (equal (plus (three, two), five) $\equiv$ true) is a sentence because all its components are sentences, appropriatly connected by logical operators.

- **Various sentences in first order logic formed using connectives :**

1) If S is a sentence, then so is its negation, $\neg S$.
2) If S_1, and S_2 are sentences, then so is their conjunction, $S_1 \wedge S_2$.
3) If S_1 and S_2 are sentences, then so is their disjunction, $S_1 \vee S_2$.
4) If S_1 and S_2 are sentences, then so is their implication, $S_1 \rightarrow S_2$.
5) If S_1 and S_2 are sentences, then so is their equivalence, $S_1 \equiv S_2$.

17. What is unification ?

Ans. : 1) It is the process of finding substitutions for lifted inference rules, which can make different logical expression to look similar (identical).

2) Unification is an procedure for determining substitutions needed to make two first order logic expressions match.

3) Unification is important component of all first order logic inference algorithms.

4) The unification algorithm takes two sentences and returns a unifier for them, if one exists.

18. Differentiate forward chaining and backward chaining.

Ans. : 1) Forward chaining is data driven.

- It is automatic unconscious processing.
- Example - Object recognition, routine decisions.
- It may do lots of work that is irrelevant to the goal.

2) Backward chaining is goal driven.

- It is appropriate for problem solving.
- Example : Where are my keys ?,

  How do I get into a PhD programme ?
- Complexity of backward chaining can be much less than linear in size of knowledgebase.

19. Define meta rules.

Ans. : The rules that determine the conflict resolution strategy are called meta rules. Meta rules define knowledge about how the system will work. For example, meta rules may define that knowledge from Expert 1 is to be trusted more than knowledge from Expert 2. Meta rules are treated by the system like normal rules, but are they are given higher priority.

20 Convert the following into Horn clauses.

$\forall x : \forall y : cat(x) \vee fish(y) \rightarrow likes\_to\_eat(x,y)$

Ans. : Horn clauses are as follows,

$$\neg \text{cat}(x) \vee \neg \textit{fish}(y) \vee \textit{likes\_to\_eat}\ (x,y)$$

21. *Discuss forward and backward chaining ?* **(Refer section 6.4)**
22. *Explain resolution procedure ?* **(Refer section 6.4)**
23. *Discuss the syntax and semantics of first order logic.* **(Refer section 6.1)**
24. *Describe the general process of knowledge engineering ?* **(Refer section 6.3)**
25. *Discuss forward and backward chaining with example ?* **(Refer section 6.4)**
26. *Describe forward chaining and backward chaining algorithm ?* **(Refer section 6.4)**
27. *Apply both algorithm to prove that "West is criminal"* **(Refer section 6.4)**
28. *What is conjunctive normal form of a rule ? Define Skolemization.* **(Refer section 6.4)**
29. *Describe in detail the steps involved in the knowledge engineering process ?* **(Refer section 6.3)**
30. *Explain Unification algorithm used for reasoning under predicate logic (first order logic) with an example.* **(Refer section 6.4)**
31. *Explain the forward chaining process in detail with example. What is the need of incremental chaining ?* **(Refer section 6.4)**
32. *Explain the inferencing process in first order logic, using suitable example.* **(Refer section 6.4)**
33. *Explain the steps involved in knowledge engineering process with example.* **(Refer section 6.3)**
34. *Discuss backward chaining algorithm.* **(Refer section 6.4)**
35. *Explain the algorithm for computing most general unifiers.* **(Refer section 6.4)**
36. *Explain with an example the use of unification algorithm to prove the concept of resolution.* **(Refer section 6.4)**
37. *Explain the forward chaining process and efficient forward chaining with example. State its usage.* **(Refer section 6.4.9)**
38. *State and explain the various steps in knowledge engineering process.* **(Refer section 6.3)**
39. *What are the steps to convert first order logic sentence to normal form ? Explain each step.* **(Refer section 6.4.11)**
40. *Represent the following sentences in predicate logic and convert the following sentences to CNF form.*
    (1) All women who like ice-creams like chocolates.
    (2) No man is happy with a spendthrift wife.
    (3) The best movie in Hollywood is always better than the best movie in Bollywood.
    (4) Some people like eating outside all the time and some people like eating at home all the time.

(5) *It might be argued that one aspect of intelligent behavior is the ability to infer new facts about the world by combining existing ones. Has the theory of logic given us a tool to allow computers to display this sort of intelligence ? Can humans make other leaps of inference that are impossible with logic alone ?* **(Refer section 6.4.11)**

41. *Differentiate propositional logic with FOL. List the inference rules along with suitable examples for first order logic.* **(Refer section 6.1)**

42. *Consider the following sentences :*
    (1) *John likes all kinds of food.*
    (2) *Apples are food.*
    (3) *Chicken is food.*
    (4) *Anything anyone eats and isn't killed alive.*
    (5) *Sue eats everything bill eats.*
    (A) *Translate these sentences into formulas in predicate logic.*
    (B) *Convert the formulas of a part into clause form.*
    (C) *Prove that "John likes peanuts" using forward chaining.*
    (D) *Prove that "John likes peanuts" using backward chaining.* **(Refer section 6.4.11)**

43. *Explain standard quantifiers of first order logic with example.* **(Refer section 6.4.3)**

44. *Explain the forward chaining algorithm with the help of the pseudo-code.* **(Refer section 6.4.9)**

45. *Give the completeness proof of resolution.* **(Refer section 6.4.11)**

46. *Explain forward chaining and backward chaining algorithm with an example.* **(Refer section 6.4.9)**

47. *Illustrate the use of first order logic to represent knowledge.* **(Refer section 6.1)**

48. *Write short note on unification.* **(Refer section 6.4.6)**

49. *Explain forward chaining and backward chaining algorithm with an example.* **(Refer section 6.4.9)**

50. *Illustrate the use of first order logic to represent knowledge.* **(Refer section 6.1)**

51. *Write short note on unification.* **(Refer section 6.4.6)**

52. *Explain in detail about forward and backward chaining with an example.* **(Refer section 6.4.9)**

53. *Consider the following sentences :*
    John likes all kinds of food
    Apples are food
    Chicken is food
    Anything anyone eats and isn't killed by is food
    Bill eats peanuts and is still alive
    Sue eats everything Bill eats.
    i) Translate these sentences into formulas in predicate logic.
    ii) Convert the formulas of part a into clause form. **(Refer section 6.4)**

54. *Trace the operation of the unifacination algorithm on each of the following pairs of literals :*
    i) f(Marcus) and f(Caesar)
    ii) f(x) and f(g(y))
    iii) f(Marcus, g(x,y)) and f(x, g(Caesar, Marcus) **(Refer section 6.4)**

55. *Consider the following facts :*
    - *Steve only likes easy courses.*
    - *Science courses are hard.*
    - *All the courses in the HaveFun department are easy.*
    - *BK301 is a HaveFun department course.*

    Use resolution to answer the question "What course would Steve like"?
    (Refer example 6.14)

56. *Relate first order logic with proposition logic and discuss in detail about the same.*
    (Refer section 6.1)

57. *Describe a procedure for converting a sentence to CNF with an example.*
    (Refer section 6.4)

58. *Explain forward chaining and backward chaining for proportional definite clauses.*
    (Refer section 6.4)

59. *Consider the following facts*
    1) All students in 4^{th} year are intelligent.
    2) Raja is a 4^{th} year student.
    3) Ragu is a 3^{rd} year student.
    4) 3^{rd} year studensts are not intelligent.
    5) 4^{th} year students have no friends in 3^{rd} year.
    Represent the facts in predicate convert to clause form and prove by resolution, "Raja is not friend of Ragu". **(Refer section 6.4)**

60. *Explain the unification algorithm with an example.* **(Refer section 6.4)**

61. *Consider the following facts*
    1) There are 5000 employees in XYZ company.
    2) Employees earning more than ₹ 25000/- annum pay tax.
    3) John is a manager in XYZ company.
    4) Manager earns ₹ 50,000/-.
    Represent the facts in predicate convert to clause form and prove by resolution, "John pays tax".
    (Refer section 6.4)

62. *Explain dempster shafer theory with an example.* **(Refer section 6.5)**
63. *What are fuzzy membership functions ? Explain them with examples.*
    (Refer section 6.5)

6.6 University Questions and Answers

Summer - 18

Q.1 *Discuss non-monotonic reasoning.* **(Refer section 6.5)** [3]

Q.2 *Explain following terms in reference to predicate logic Resolution.*
a. Unsuccessful attempt at resolution b. Equality c. Reduce
d. Trying several substitute **(Refer section 6.4)** [7]

Winter - 18

Q.3 *Explain the procedure to convert well formed formula to clause form with the help of example.* **(Refer section 6.1)** [7]

Q.4 *Differentiate monotonic and non-monotonic reasoning.* **(Refer section 6.5)** [4]

Q.5 *Explain resolution in predicate logic.* **(Refer section 6.4)** [7]

Summer - 19

Q.6 *Discuss non-monotonic reasoning.* **(Refer section 6.5)** [3]

Winter - 19

Q.7 *Write a note on non-monotonic reasoning.* **(Refer section 6.5)** [4]

Summer - 20

Q.8 *Define the following words in the context of AI : Logical reasoning.* **(Refer section 6.1)** [1]

Q.9 *Define predicate logic.* **(Refer section 6.1)** [2]

Q.10 *List out the property of monotonic and non monotonic reasoning.* **(Refer section 6.5)** [4]

❑❑❑

7 Uncertainty

Syllabus

Uncertainty - *Acting under Uncertainty, Basic Probability Notation, The Axioms of Probability, Inference Using Full Joint Distributions.*

Contents

7.1 Acting Under Uncertainty

Introduction

A agent working in real world environment almost never has access to whole truth about its environment. Therefore, agent needs to work under uncertainity.

Earlier agents we have seen make the epistemological commitment that either the facts (expressed as propositions) 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 not conclude that any perticular 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.

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 :

1) Laziness :

It is too much work to list the complete set of antecedents or consequents needed to ensure an exceptionless rule and too hard to use such rules.

2) Theoretical ignorance :

A perticular problem may not have complete theory for the domain.

3) Practical ignorance :

Even if all the rules are known, perticular aspects of problem are not checked yet or some details are not considered at all (missing out the details).

The agent's knowledge can provide it with a **degree of belief** with relevent 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 uncertainity 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 percepts of agent at perticular time. These percepts 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 percepts, 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 percepts 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 perticular 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 perticular 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.

7.2 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**".

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.

7.3 The Basic Probability Notation

GTU : Winter-18

The probability theory uses propositional logic language with additional expressivness.

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.

7.3.1 Propositions

1) The propositions (assertions) are attached with the degree of belief.
2) Complex proposition can be formed using standard logical connectives.

   For example : [(Cavity = True) $\wedge$ (Toothache = False)] and [(Cavity $\wedge \neg$ Toothache)] both are same assertions.

- **The random variable :**

1) The basic element of language is random variable.
2) It reffers 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.
3) Random variables are like symbols in propositional logic.
4) 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)
5) 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 < true, false >
6) A random variable's proposition will assert that what value is drawn for the random variable from its domain.

   For example : Cavity = True is proposition.

   Saying that "there is cavity in my lower left wisdom tooth".
7) Random variables are divided into three kinds, depending on their domain. The types are as follows.
   i) **Boolean random variables** : These are random variables that can take up only boolean values.

      For example : Cavity, it takes value either true or false.

ii) **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 < sunny, rainy, cloudy, cold >

iii) **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 $X \leq 4.14$.

7.3.2 Atomic Events

1) An atomic event is a complete specification of the state of the world about which agent is uncertain.
2) 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) Cavity = False $\wedge$ Toothache = True
b) Cavity = False $\wedge$ Toothache = False
c) Cavity = True $\wedge$ Toothache = False
d) Cavity = True $\wedge$ Toothache = True
e) properties of atomic events
   i) They are mutually exclusive - That is at most one can actually be the case.

      For example : (Cavity $\wedge$ Toothache) and (Cavity $\wedge \neg$ Toothache) can not both be the case.

   ii) 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.
   iii) Any particular atomic event entails the truth or falsehood of every proposition, whether simple or complex.

      For example -

      The atomic event
      (Cavity $\wedge \neg$ Toothache) entails the truth of cavity and the Falsehood of (cavity $\rightarrow$ toothache).
   iv) 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).

7.3.3 Prior Probability (Unconditional Probability)

1) The prior (unconditional) probability is associated with a proposition 'a'.
2) It is the degree of belief accorded to a proposition in the **absence of any other information.**
3) It is written as P(a).

   For example : The probability that, Ram has cavity = 0.1, then prior probability is written as, P(Cavity = true) = 0.1 or P(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 **P**(a). This represents values for the probabilities of each individual state of the 'a'.

   For example : **P**(Weather) = < 0.7, 0.2, 0.08, 0.02 > is representing four equations

   P (Weather = Sunny) = 0.7

   P (Weather = Rain) = 0.2

   P (Weather = Cloudy) = 0.08

   P (Weather = Cold) = 0.02
6) The expression P(a) is said to be defining prior probability distribution for the random variable 'a'.
7) 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.
8) A simple example of joint probability distribution is,

   → **P**<Weather, Cavity> can be represented as, 4×2 table of probabilities.

   (Weather's probability) (Cavity probability)
9) A joint probability distribution that covers the complete set of random variables is called as **full joint probability distribution.**
10) A simple example of full joint probability distribution is,

    If problem world consists of 3 random variables, wheather, cavity, toothache then full joint probability distribution would be,

    P<Weather, Cavity, Toothache>

It will be represented as, $4 \times 2 \times 2$, table of probabilities.

11) **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.

7.3.4 Conditional Probability

1) When agent obtains evidence concerning previously unknown random variables in the domain, then prior probability are not used. Based on new information conditional or posterior probabilities are calculated.

2) The notation is $P(a \mid b)$ where a and b are any proposition.

The 'P' is read as "the probability of a given that all we know is b". That is when b is known it indicates probability of a.

For example : $P(\text{Cavity} \mid \text{Toothache}) = 0.8$

it means that, if patient has toothache (and no other information is known) then the chances of having cavity are = 0.8

3) Prior probability are infact special case of conditional probability. It can be represented as **P**(a) which means that probability 'a' is conditioned on no evidence.

4) Conditional probability can be defined interms of unconditional probabilities. The equation would look like,

$$P(a/b) = \frac{P(a \wedge b)}{P(b)}, \text{ it holds whenever } P(b) > 0 \quad \ldots (7.3.1)$$

The above equation can also be written as,

$P(a \wedge b) = P(a \mid b) P(b)$

This is called as **product rule**. In other words it says, for 'a' and 'b' to be true we need 'b' to be true and we need a to be true given b. It can be also written as,

$P(a \wedge b) = P(b \mid a) P(a)$.

5) Conditional probability are used for probabilistic inferencing.

6) **P** notation can be used for conditional distribution. $P(x|y)$ gives the values of $P(X = x_i | Y = y_j)$ for each possible i, j.

following are the individual equations,

$$P(X = x_1 \wedge Y = y_1) = P(X = x_1 | Y = y_j) P(Y = y_1)$$

$$P(X = x_1 \wedge Y = y_2) = P(X = x_1 | Y = y_2) P(Y = y_2)$$

$$\vdots$$

These can be combined into a single equation as,

$$\mathbf{P}(X, Y) = \mathbf{P}(X|Y)\ \mathbf{P}(Y)$$

7) Conditional probabilities **should not be** treated as logical implications. That is, "When 'b' holds then probability of 'a' is something", is a conditional probability and not to be mistake as logical implication. It is wrong on two points, one is, P(a) always denotes prior probability. For this it do not require any evidence. Secondly $P(a|b) = 0.7$, is immediately relevant when b is available evidence. This will keep on altering. When information is updated logical Implications do not change over time.

7.3.5 The Probability Axioms

Axioms gives semantic of probability statements. The basic axioms (Kolmogorov's axioms) serve to define the probability scale and its end points.

1) All probabilities are between 0 and 1. For any proposition a, $0 \leq P(a) \leq 1$.

2) Necessarily true (i.e, valid) propositions have probability 1, and necessarily false (i.e., unsatisfiable) propositions have probability 0.

$$P(\text{true}) = 1 \qquad P(\text{false}) = 0$$

3) The probability of a disjunction is given by

$$P(a \vee b) = P(a) + P(b) - P(a \wedge b)$$

This axiom connects the probabilities of logically related propositions. This rule states that, the cases where 'a' holds, together with the cases where 'b' holds, certainly cover all the causes where 'a ∨ b' holds; but summing the two sets of cases counts their intersection twice, so we need to subtract $P(a \wedge b)$.

Note : The axioms deal only with prior probabilities rather than conditional probabilities; this is because prior probability can be defined in terms of conditional probability.

Using the axioms of probability :

From basic probability axioms following facts can be deduced.

$P(a \vee \neg a) = P(a) + P(\neg a) - P(a \wedge \neg a)$ (by axiom 3 with $b = \neg a$)

$P(true) = P(a) + P(\neg a) - P(false)$ (by logical equivalence)

$1 = P(a) + P(\neg a)$ (by axiom 2)

$P(\neg a) = 1 - P(a)$ (by algebra).

- Let the discrete variable D have the domain $<d_1, ..., d_n>$.

  Then, $\sum_{i=1}^{n} P(D = d_i) = 1$.

  That is, any probability distribution on a single variable must sum to 1.

- It is also true that any joint probability distribution on any set of variables must sum to 1. This can be seen simply by creating a single megavariable whose domain is the cross product of the domains of the original variables.
- Atomic events are mutually exclusive, so the probability of any conjunction of atomic events is zero, by axiom 2.
- From axiom 3, we can derive the following simple relationship : The probability of a proposition is equal to the sum of the probabilities of atomic events in which it holds; that is $P(a) = \sum_{e_i \in e(a)} P(e_i)$ ... (7.3.2)

7.3.6 Inference using Full Joint Distribution

Probabilistic inference means, computation from observed evidence of posterior probabilities, for query propositions. The knowledge base used for answering the query is represented as full joint distribution. Consider simple example, consists of three boolean variables, toothache, cavity, catch. The full joint distribution is $2 \times 2 \times 2$, as shown below.

	Toothache		Toothache	
	Catch	¬ Catch	Catch	¬ Catch
Cavity	0.108	0.012	0.072	0.008
¬ Cavity	0.016	0.064	0.144	0.576

Note that the probability in the joint distribution sum to 9.

- One perticular common task in inferencing is to extract the distribution over some subset of variables or a single variable. This distribution over some variables or single variables is called as **marginal probability**.

  For example : P(Cavity) = 0.108 + 0.012 + 0.072 + 0.008 = 0.2

This process is called as **marginalization** or **summing**. Because the variables other than 'cavity' (that is whose probability is being counted) are summed out.

- The general marginalization rule is as follows,

  For any sets of variables Y and Z,

$$\mathbf{P}(Y) = \sum_{z} P(Y, z). \qquad \ldots (7.3.3)$$

  It indicates that distribution Y can be obtained by summing out all the other variables from any joint distribution X containing Y.

- Variant of above example of general marginalization rule involved the conditional probabilities using product rule.

$$\mathbf{P}(Y) = \sum_{z} P(Y \mid z)\, P(z) \qquad \ldots (7.3.4)$$

  This rule is conditioning rule.

  For example : Computing probability of a cavity, given evidence of a toothache is as follows,

$$P(\text{Cavity} \mid \text{Toothache}) = \frac{P(\text{Cavity} \wedge \text{Toothache})}{P(\text{Toothache})} = \frac{0.108 + 0.012}{0.108 + 0.012 + 0.016 + 0.064} = 0.6$$

- **Normalization constant :** It is variable that remains constant for the distribution, which ensures that it adds in to 1. α is used to denote such constant.

For example : We can compute the probability of a cavity, given evidence of a toothache, as follows :

$$P(\text{Cavity} \mid \text{Toothache}) = \frac{P(\text{Cavity} \wedge \text{Toothache})}{P(\text{Toothache})}$$

$$= \frac{0.108 + 0.012}{0.108 + 0.012 + 0.016 + 0.064} = 0.6$$

Just to check we can also compute the probability that there is no cavity given a toothache :

$$P\ (\text{Cavity/Tootache}) = \frac{P\ (\neg\ \text{Cavity} \wedge \text{Toothache})}{P\ (\text{Toothache})}$$

$$= \frac{0.016 + 0.064}{0.108 + 0.012 + 0.016 + 0.064} = 0.4$$

Notice that in these two calculations the term 1/P (toothache) remains constant, no matter which value of cavity we calculate. With this notation we can write above two equations in one.

$$\mathbf{P}(\text{Cavity} \mid \text{Toothache}) = \propto \mathbf{P}(\text{Cavity, Toothache})$$

$$= \alpha\ [\mathbf{P}(\text{Cavity, Toothache, Catch}) + \mathbf{P}(\text{Cavity, Toothache, } \neg\ \text{Catch})]$$

$$= \alpha\ [<0.108, 0.016> + <0.012, 0.064>]$$

$= \alpha \langle 0.12, 0.08 \rangle = \langle 0.6, 0.4 \rangle$

From above one can extract a **general inference procedure**.

Consider the case in which query involves a single variable. The notation used is let X be the query variable (cavity in the example), let E be the set of evidence variables (just toothache in the example) let e be the observed values for them, and let Y be the remaining unobserved variable (just catch in the example). The query is **P**(X/e) and can be evaluated as

$$\mathbf{P}(X \mid e) = \alpha\ \mathbf{P}(X, e) = \alpha \sum_{y} \mathbf{P}(X, e, y) \quad \ldots (7.3.5)$$

where the summation is over all possible ys (i.e. all possible combinations of values of the unobserved variables 'y'). Notice that together the variables, X, E and Y constitute the complete set of variables for the domain, so **P** (X, e, y) is simply a subset of probabilities from the full joint distribution.

7.3.7 Independance

It is a relationship between two different sets of full joint distributions. It is also called as **marginal** or **absolute independance** of the variables. Independence indicates that whether the two full joint distributions affects probability of each other.

The independance between variables X and Y can be written as follows,

$P(X \mid Y) = P(X)$ or $P(Y \mid X) = P(Y)$ or $P(X, Y) = P(X)\ P(Y)$

- For example : The weather is independant of once dental problem. Which can be shown as below equation.

P(Toothache, Catch, Cavity, Weather) = **P**(Toothache, Catch, Cavity) **P**(Weather).

- Following diagram shows factoring a large joint distributing into smaller distributions, using absolute independence. Weather and dental problems are independent.

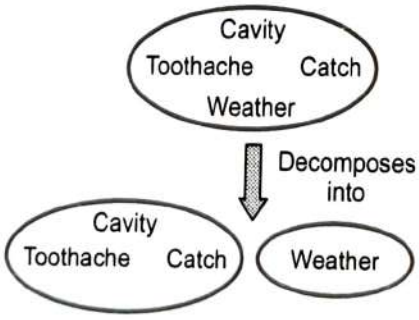


Fig. 7.3.1 Factoring a large joint distributing into smaller distribution

7.3.8 Bayes' Rule

Bayes' rule is derived from the product rule.

The product rule can be written as,

$$P(a \wedge b) = P(a|b)\, P(b) \quad \ldots (7.3.6)$$

$$P(a \wedge b) = P(b|a)\, P(a) \quad \ldots (7.3.7)$$

[because conjunction is commutative]

Equating right sides of equation (7.3.6) and equation (7.3.7) and dividing by P(a),

$$\mathbf{P}(b|a) = \frac{P(a|b)P(b)}{P(a)}$$

This equation is called as Bayes' rule or Bayes' theorem or Bayes' law. This rule is very useful in probabilistic inferences.

Generalized Bayes' rule is,

$$\mathbf{P}(Y|X) = \frac{\mathbf{P}(X|Y)\ \mathbf{P}(Y)}{\mathbf{P}(X)}$$

(where **P** has same meanings)

We can have more general version, conditionalized on some background evidence e.

$$\mathbf{P}(Y|X, e) = \frac{\mathbf{P}(X|Y,e)\ \mathbf{P}(Y|e)}{\mathbf{P}(X|e)}$$

General form of Bays' rule with normalization's

$\mathbf{P}(y|x) = \alpha\ \mathbf{P}(x|y)\ \mathbf{P}(y)$.

Applying Bays' Rule :

1) It requires total three terms (1 conditional probability and 2 unconditional Probabilities). For computing one conditional probability.

   For example : Probability of patient having low sugar has high blood pressure is 50 %.

   Let, M be proposition, 'patient has low sugar'.

   S be a proposition, 'patient has high blood pressure'.

   Suppose we assume that, doctor knows following unconditional fact,

   i) Prior probabilition of (m) = 1/50,000.

   ii) Prior probability of (s) = 1/20.

   Then we have,

$$P(s|m) = 0.5$$

$$P(m) = 1|50000$$

$$P(s) = 1|20$$

$$P(m|s) = \frac{P(s|m)P(m)}{P(s)}$$

$$= \frac{0.5 \times 1|50000}{1|20}$$

$$= 0.0002$$

That is, we can expect that 1 in 5000 with high B.P. will has low sugar.

2) Combining evidence in Bayes' rule.

Bayes rule is helpful for answering queries conditioned on evidences.

For example : Toothache and catch both evidences are available then cavity is sure to exist. Which can be represented as,

$\mathbf{P}(\text{Cavity} | \text{Toothache} \wedge \text{Catch}) = \alpha \langle 0.108, 0.016 \rangle \approx \langle 0.871, 0.129 \rangle$

By using Bayes' rule to reformulate the problem :

$\mathbf{P}(\text{Cavity} | \text{Toothache} \wedge \text{Catch}) = \alpha\ \mathbf{P}(\text{Toothache} \wedge \text{Catch} | \text{Cavity})\ \mathbf{P}(\text{Cavity})$

... (7.3.8)

For this reformulation to work, we need to know the conditional probabilities of the conjunction Toothache $\wedge$ Catch for each value of Cavity. That might be feasible for just two evidence variables, but again it will not scale up.

If there are n possible evidence variable (X-rays, diet, oral hygiene, etc.), then there are 2^n possible combinations of observed values for which we would need to know conditional probabilities.

The notion of independence can be used here. These variables are independent, however, given the presence or the absence of a cavity. Each is directly caused by the cavity, but neither has a direct effect on the other. Toothache depends on the state of the nerves in the tooth, where as the probe's accuracy depends on the dentist's skill, to which the toothache is irrelevant.

Mathematically, this property is written as,

$\mathbf{P}(\text{Toothache} \wedge \text{Catch} | \text{Cavity}) = \mathbf{P}(\text{Toothache} | \text{Cavity})\ \mathbf{P}(\text{Catch} | \text{Cavity})$ ... (7.3.9)

This equation expresses the conditional independence of toothache and catch, given cavity.

Substitute equation (7.3.3) into (7.3.4) to obtain the probability of a cavity :

$\mathbf{P}(\text{Cavity} | \text{Toothache} \wedge \text{Catch}) = \alpha\ \mathbf{P}(\text{Toothache} | \text{Cavity})\ \mathbf{P}(\text{Catch} | \text{Cavity})\ \mathbf{P}(\text{Cavity})$

Now, the information requirement are the same as for inference using each piece of evidence separately the prior probability $\mathbf{P}(\text{Cavity})$ for the query variable and the conditional probability of each effect, given its cause.

Conditional independence assertions can allow probabilistic systems to scale up; more over, they are much more commonly available than absolute independence assertions. When their are 'n' variables given that they are all conditionally independent, the size of the representation grows as O(n) instead of $O(2^n)$.

For example -

Consider dentistry example, in which a single cause, directly influences a number of effects, all of which are conditionally independent, given the cause.

The full joint distribution can be written as,

$\mathbf{P}(\text{Cause}, \text{Effect}_1, \ldots, \text{Effect}_n) = \mathbf{P}(\text{Cause}) \prod \mathrm{P}(\text{Effect}_i \mid \text{Cause})$.

Such a probability distribution is called as **naive Bayes' model** - "naive" because it is often used (as a simplifying assumption) in cases where the "effect"variables are not conditionally independent given the cause variable. The naive Bayes model is sometimes called as **Bayesian classifier**.

Answer in Brief

1. *Explain the process of inference using full joint distribution with example.*
   (Refer section 7.3.6)
2. *Define Dempster-Shafer theory.*

Ans. : The Dempster-Shafer theory is designed to deal with the distinction between uncertainty and ignorance.

Rather than computing the probability of a proposition, it computes the probability the evidence that supports the proposition.

3. *Define : Baye's theorem.*

Ans. : In probability theory and applications, Baye's theorem (alternatively called as Baye's law or Bayes rule) links a conditional probability to its inverse.

$$\mathrm{P}(b \mid a) = \frac{\mathrm{P}(a \mid b)\,\mathrm{P}(b)}{\mathrm{P}(a)}$$

This equation is called as Baye's Rule or Baye's Theorem.

4. *What is reasoning by default ?*

Ans. : We can do qualitative reasoning using technique like default reasoning.

Default reasoning treats conclusions not as "believed to a certain degree", but as "believed until a better reason is found to believe something else".

5. *What are the logics used in reasoning with uncertain information ?*

Ans. : There are two approaches that can be taken for reasoning with uncertain information in which logic is used.

Non-monotonic logic is used in default reasoning process. Default reasoning also uses other type or logic called as **default logic**.

The second approach towards reasoning is vagueness which uses fuzzy logic. Fuzzy logic is a method for reasoning with logical expressions describing membership in fuzzy sets.

6. Define prior probability.

Ans. : The prior (unconditional) probability is associated with a proposition 'a'.

The prior probability 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 the prior priobility is written as,

P(Cavity = true) = 1 or P(cavity) = 0.1

7. State the types of approximation methods.

Ans. : For approximate inferencing randomize sampling algorithm (Monte Carlo Algorithm) is used. There are two approximation methods that are used in randomize sampling algorithm which are 1) Direct sampling algorithm and 2) Markov chain sampling algorithm.

In direct sampling algorithm samples are generated from known probability distribution. In Markov chain sampling each event is generated by making a random change to the preceding event.

8. What do you mean by hybrid Bayesian network ?

Ans. : A network with both discrete and continuous variables is called as hybrid Bayesian network. In hybrid Bayesian network, for representing the continuous variable its discretization is done in terms of intervals because it can have infinite values.

For specifying the hybrid network two kinds of distribution are specified. The conditional distribution for a continuous variable given discrete or continuous parents and the conditional distribution for a discrete variable given continuous parent.

9. Define computational learning theory.

Ans. : The computational learning theory is a mathematical field related to the analysis of machine learning algorithms.

The computational learning theory is used in the evaluation of sample complexity and computational complexity. Sample complexity targets the issue that, how many training examples are needed to learn a successful hypothesis ? The computational complexity evaluates that how much computational effort is needed to learn a successful hypothesis ?

In addition to performance bounds, computational learning theory also deals with the time complexity and feasibility of learning.

10. Give the full specification of Bayesian network.

Ans. : 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 :-

1) A set of random variables makes up the nodes of the network. Variables may be discrete or continuous.
2) 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.
3) 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.
4) 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.**

7.4 University Question with Answer

Winter - 18

Q.1 *Explain Bay's theorem.* **(Refer section 7.3)** **[4]**

❑❑❑

8 Probabilistic Reasoning

Syllabus

Representing Knowledge in an Uncertain Domain, The Semantics of Bayesian Networks, Efficient Representation of Conditional Distribution, Exact Inference in Bayesian Networks, Approximate Inference in Bayesian Networks.

Contents

8.1 Probabilistic Reasoning

GTU : Summer-18,19, Winter-19

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.

8.1.1 Representing Knowledge in an Uncertain Domain and Bayesian Network

8.1.1.1 Bayesian Networks 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 :-

1) A set of random variables makes up the nodes of the network. Variables may be discrete or continuous.
2) 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.
3) Each node X_i, has a conditional probability distribution $P(X_i \mid Parents(X_i))$ that quantifies the effect of the parents on the node.
4) 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 intutive 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 :

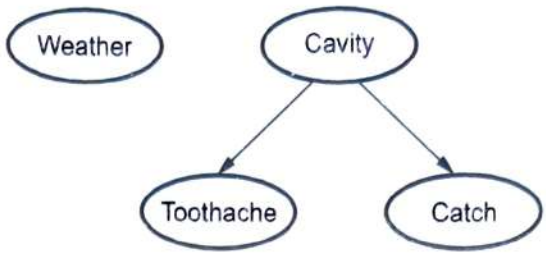


Fig. 8.1.1 A simple Bayesian network in which Weather is independent of other three variables and Toothache and Catch are conditionally independent, given the cavity

Note

1) The conditional independence of toothache and catch given the cavity is indicated by the absence of a link between toothache and catch.
2) The network represents the fact that cavity is a direct cause of toothache and catch.
3) 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. M 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 :

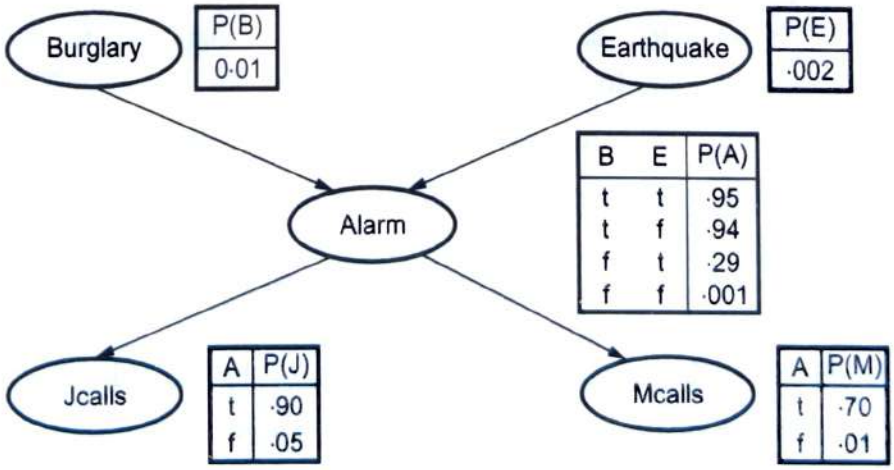


Fig. 8.1.2 A Typical Bayesian network, showing both the topology and the conditional probability tables (CPTs).

Note *In the figure, in the CPTs, the letters B, E, A, J and M stand for Burglary, Earthquake, Alarm, Johncalls, Marycalls respectively.*

1) 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.

2) For boolean variable if it is true and its probability is know to be 'P' then when it is false its probability is 1–P and it is ommited from the table (because it can be deduced from 'P').

3) 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.

8.1.1.2 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 constructs 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 1

- Representing the full joint distribution :
  1) Every entry in the full joint probability distribution (hereafter abbreviated as "joint") can be calculated from the information in the network.
  2) A generic entry in the joint distribution is the probability of a conjunction of particular assignments to each variable, such as $P(X_1 = x_1 \wedge ..... \wedge X_n = x_n)$
  3) The notation $P(x_1, ..., x_n)$ is used as an abbrevation for this.
  4) The value of this entry is given by the formula.

$$P(x_1, ..., x_n) = \prod_{i=1}^{n} P(x_i \mid \text{parents } (X_i)) \quad \ldots (8.1.1)$$

  Where parents (x_i) denotes the specific values of the variables in parents (x_i).

  5) 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 :

$$P(j \wedge m \wedge a \wedge \neg b \wedge \neg e) = P(j|a)\, P(m|a)\, P(a|\neg b \wedge \neg e)\, P(\neg b)\, P(\neg e)$$

$$= 0.90 \times 0.70 \times 0.001 \times 0.999 \times 0.998 = 0.00062$$

6) 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 :

1) We rewrite the joint distribution in terms of a conditional probability, using the product rule.

$$P(x_1, \ldots, x_n) = P(x_n | x_{n-1}, \ldots, x_1)\ P(x_{n-1}, \ldots, x_1).$$

2) Then the process is represented, reducing each conjunctive probability to a conditional probability and a smaller conjunction. We end up with one big product.

$$P(x_1, \ldots, x_n) = P(x_n | x_{n-1}, \ldots, x_1) P(x_{n-1} | x_{n-2}, \ldots, x_1) \ldots P(x_2 | x_1)\, P(x_1)$$

$$= \prod_{i=1}^{n} P(x_i | x_{i-1}, \ldots x_1) \qquad \ldots (8.1.2)$$

3) Above identity holds true for any set of random variables and is called the **Chain rule.** Comparing it with equation (8.1.1). We see that the specification of the joint distribution is equivalent to the general assertion that, for every variable X_i in the network,

$$P(X_i | X_{i-1}, \ldots X_1) = P(X_i | \text{Parents}(X_i)),$$

provided that parents $(X_i) \subseteq \{X_{i-1}, \ldots X_1\}$. 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 $X_i, \ldots X_{i-1}$ that directly influence X_i.

For example : Suppose we have completed the network in Fig. 8.1.2 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. Intitutively, 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 :

P(M Calls | J Calls, Alarm, Earthquake, Burglary) = P(M calls | 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 :

1) 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.
2) If wrong order is choosen we get more complicated network.

For example : Consider network shown in following diagram.

The network construction process goes as follows :

i) Adding M calls : No parent.

ii) Adding J calls : If M calls, that probably means the alarm has gone off, which ofcourse would make it more likely that J calls. Therefore J calls needs M calls as a parent.

iii) 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.

iv) 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(Burglary | Alarm, J calls, M calls) = **P**(Burglary | Alarm).

Hence we need just alarm as parent.

v) 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 diagram 8.1.4 :

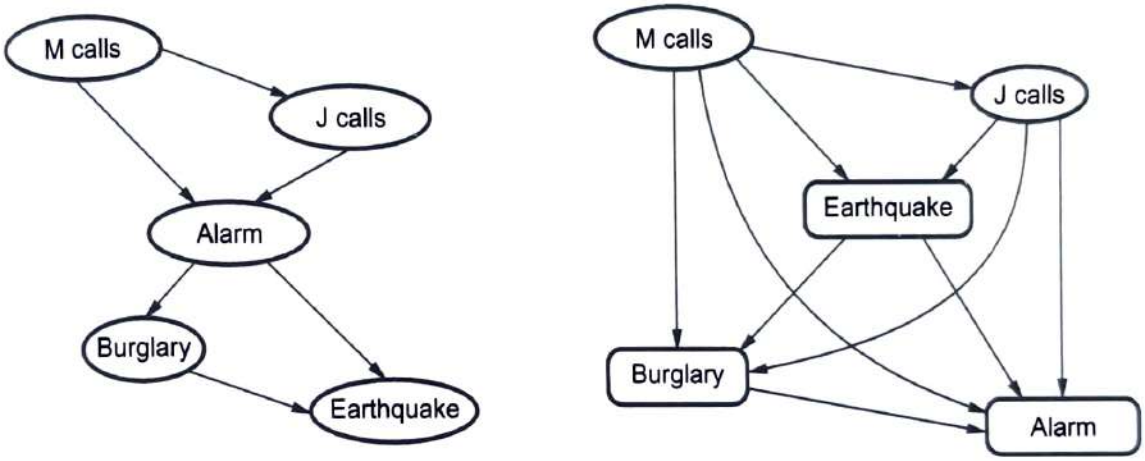


Fig. 8.1.3 and 8.1.4 Network structure depends on order of introduction. In each network nodes are added from top to bottom

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.

8.1.1.3 Efficient Representation of Conditional Distribution

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. 8.1.2 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.

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).

A node X is conditionally independent of all other nodes in the network given its Markove blanket (the gray area).

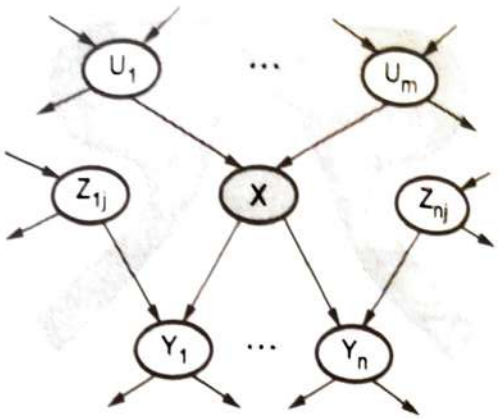


Fig. 8.1.5 (a) A node x is conditionally independent of its non-descendants. (e.g., the Z_{ij}s) given its parents (the U_i s shown in the gray area).

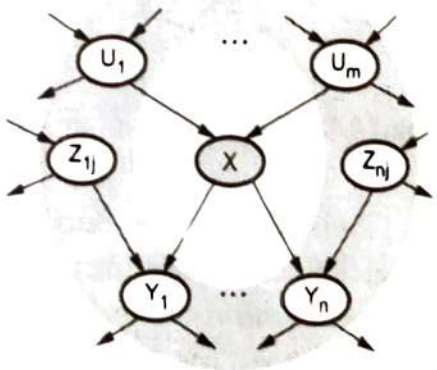


Fig. 8.1.5 (b) A node x is conditionally independent of all other nodes in the network given its markov blanket (the gray area)

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 canconical 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 uncertainity.

The uncertain relationships are often represented by noisy-OR relationships. [Noisy-OR is generalization of logical OR]. The noisy-OR model allows uncertainity 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.

8.1.2 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(Cost | Harvest, Subsidy). The discrete parent is handled by explicit enumeration that is, specifying both P(Cost | Harvest, Subsidy) and P(Cost | Harvest, ¬ 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'.

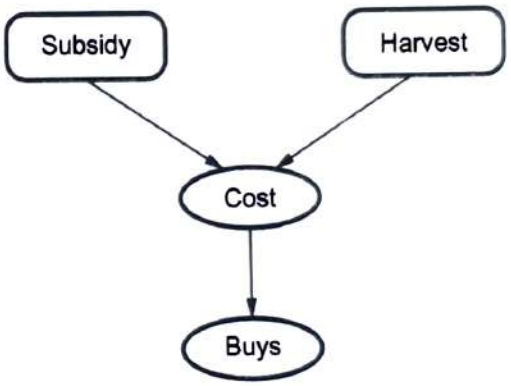


Fig. 8.1.6 A simple network with discrete variables (Subsidy and Buys) and continuous variables (Harvest and Cost).

8.1.3 Inferencing in Bayesian Networks

8.1.3.1 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,....., E_m and 'e' is the perticular observed event.

Y : The set of non-evidence variables $Y_1, Y_2, \ldots Y_k$ [Non-evidence variables are also called as hidden variables].

X : It the complete set of all the types of variables, where $\mathbf{X} = \{X\} \cup E \cup Y$.

- 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 Jcalls = true and Mcalls = true.

  The query is 'Has burglary occured ?'

  The probability distribution for this situation would be,

  P(Burglary | J calls = true, M calls = true) = < 0.284, 0.716 >

8.1.3.2 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 :

1) 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.
2) 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 explicitely. The time complexity for network with 'n' boolean variables is always $O(2^n)$ which is better than the $O(n\,2^n)$ required in simple inferencing approach (using posterior probability).
3) The drawback of the algorithm is, it keeps on evaluating repeated subexpression 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 = 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.

$Q(x_i) \leftarrow$ ENUMERATE-ALL (VARS[bn] e) return NORMALIZE (Q(x)).

Function ENUMERATE-ALL (vars, e) returns a real number.

if EMPTY ? (vars) then return 1.0

Y$\leftarrow$ FIRST (vars)

If Y has value y in e.

Then return P(y | parents(Y)) X ENUMERATE-ALL (REST(vars), e)

else return $\sum_y$ P(y | parents(Y)) X ENUMERATE-ALL (REST(vars), e_y)

where e_y is e extended with Y = y.

Example :

Consider query,

P(Burglary | J calls = true, M calls = true)

Hidden variables in the queries are $\rightarrow$ Earthquake and Alarm.

using the query equation.

$$\mathbf{P}(\text{Burglary} \mid j, m) = \alpha\ \mathbf{P}(\text{Burglary}, J, M) = \alpha \sum_e \sum_a \mathbf{P}(\text{Burglary}, e, a, j, m)$$

The semantics of Bayesian networks (equation 8.1.1) then gives us an expression in terms of CPT entries. For simplicity, we will do this just for Burglary = 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

$$P(b \mid j, m) = \alpha P(b) \sum_{e} P(e) \sum_{a} P(a \mid b, e) P(j \mid a) P(m, a) \quad \ldots (8.1.3)$$

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. 8.1.2, 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 %.

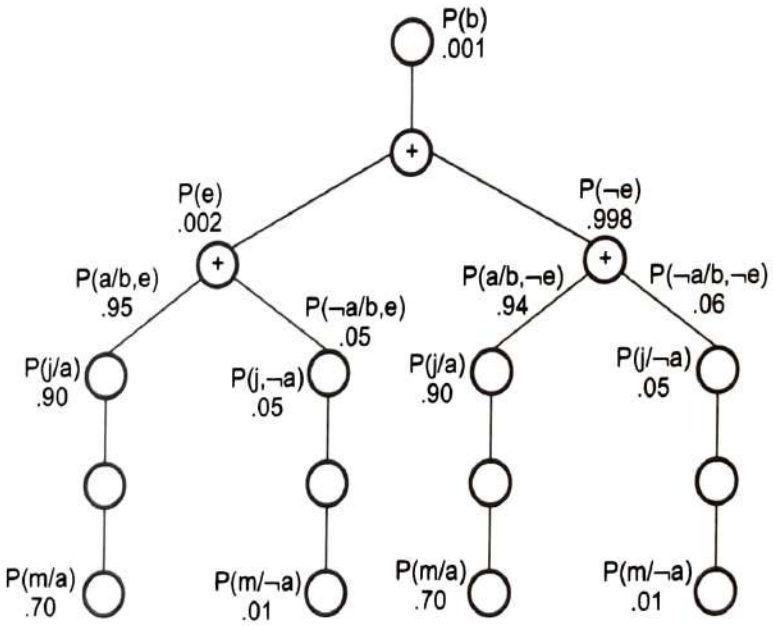


Fig. 8.1.7 The structure of the expression shown in equation 8.1.3

Note *In the Fig. 8.1.7, the evalution proceeds top to down, multiplying values along each path and summing at the "t" nodes. Observe that there is repetition of paths for j and m.*

8.1.3.3 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 $[P(b|j, m) = \alpha\ P(b) \sum_e P(e) \sum_a P(a|b, e)\ P(j|a)\ P(m|a)]$ 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 :

$$P(B|j, m) = \underbrace{\alpha\ P(B)}_{B} \sum_e \underbrace{P(e)}_{E} \sum \underbrace{P(a|B, e)}_{A} \underbrace{P(j|a)}_{J} \underbrace{P(m|a)}_{M}$$

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 $f_J(A)$.

iii) The factor for A is $P(a|B, e)$ which will be a $2\times2\times2$ 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.

$$f_{\bar{A}JM}(B, E) = \sum_a f_A(a, B, E) \times f_J(a) \times f_M(a)$$

$$= f_A(a, B, E) \times f_J(a) \times f_M(a) + f_A(\neg a, B, E) \times f_J(\neg a) \times f_M(\neg a)$$

The multiplication process used is called a printwise 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) :

$$P(B|j, m) = \alpha\ f_B(B) \times f_{\bar{E}\bar{A}JM}(B).$$

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 $Y_1, \ldots, Y_k$. Then we have

$$f(X_1, \ldots, X_j, Y_1, \ldots Y_k, Z_1, \ldots Z_l) = f_1(X_1, \ldots X_j, Y_1 \ldots Y_k)\, f_2(Y_1, \ldots Y_k, Z_1, \ldots Z_l).$$

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 × .2
T	F	.7	T	F	.8	T	T	F	.3 × .8
F	T	.9	F	T	.6	T	F	T	.7 × .6
F	F	.1	F	F	.4	T	F	F	.7 × .4
						F	T	T	.9 × .2
						F	T	F	.9 × .8
						F	F	T	.1 × .6
						F	F	F	.1 × .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.

$$f_J(A) \times f_M(A) \times \sum_e f_E(e) \times f_A(A, B, e) = f_J(A) \times f_M(A) \times f_{\bar{E}A}(A, B).$$

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 given below, the variable elimination algorithm is shown below :

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 $P(X_1, \ldots, X_n)$.

Factors ← [ ] : vars ← REVERSE (VARS [bn])

for each var in vars do

Factors ← [MAKE-FACTOR (var, e) factors]

if var is a hidden variable then

Factors ← SUM-OUT (var, factors)

returns NORMALIZE (POINTWISE-PRODUCT (factors)).

P(J calls, Burglary = 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 calls/Burglary = 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.*

8.1.3.4 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. 8.1.2 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.

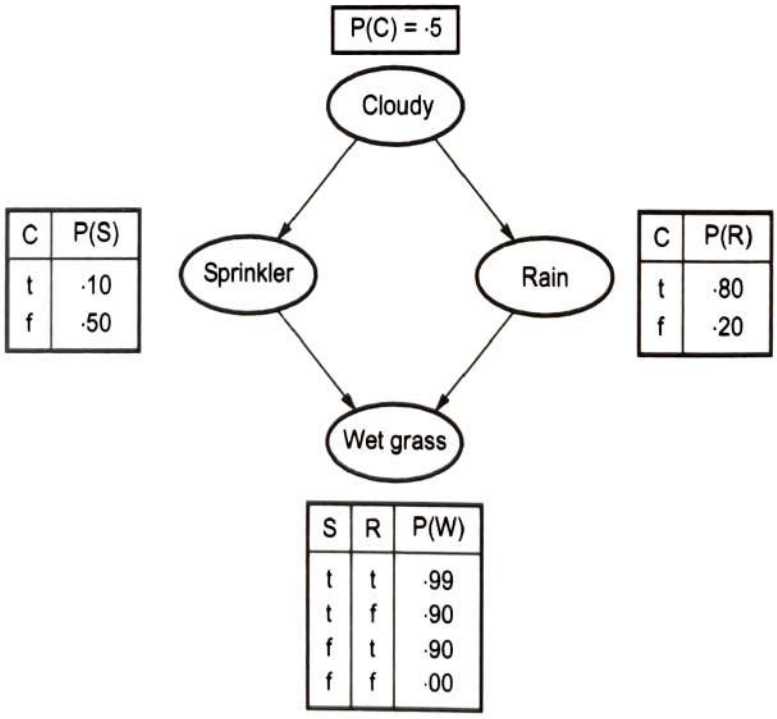


Fig. 8.1.8 (a) A multiply connected network with conditional probability tables

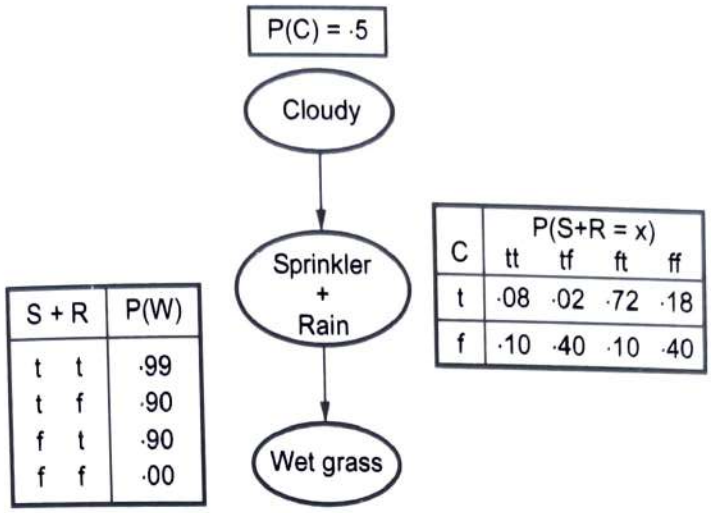


Fig. 8.1.8 (b) A clustered equivalent of the multiply connected network

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.

For example : The multiply connected network shown in following diagram 8.1.8 (a) can be converted into a polytree by combining the Sprinkler and Rain node into a cluster node called Sprinkler + Rain, as shown in diagram 8.1.8 (b). The two Boolean nodes are replaced by a meganode that takes on four possible values : TT, TF, FT, FF. The meganode has only one parent, the Boolean variable. Cloudy, so there are two conditioning cases.

Peculiarities of algorithm :

1) 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.
2) 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.
3) 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.

8.1.4 Approximate Inference in Bayesian Network

Because of the fact that exact inference in multiply connected network is intractable problem, we go for approximate inferencing.

For approximate inferencing randomize sampling algorithm (Monte Carlo Algorithm) is used. Its accuracy depends on number of samples generated. Monte Carlo Algorithm is widely used in problem areas where it is difficult to calculate exact quantities.

There are two **families** of Monte Carlo algorithm :

1) Direct sampling algorithm.
2) Markov chain sampling algorithm.

8.1.4.1 Direct Sampling Algorithm

1) The basic element in any sampling algorithm is the generation of samples from a known probability distribution.

   For example : An unbiased coin can be thought of as a random variable coin with values (heads, tails) and a prior distribution P(coin) = <0.5, 0.5>. Sampling from this distribution is exactly like flipping the coin : with probability 0.5 it will return heads, and with probability 0.5 it will return tails. Given a source of random numbers in the range [0, 1], it is a simple matter to sample any distribution on a single variable.

2) The simplest kind of random sampling process for Bayesian networks generates events from a network that has no evidence associated with it. The idea is to sample each variable in turn, in topological order.

3) The probability distribution from which the value is sampled is conditioned on the values already assigned to the variable's parents.

4) The sampling algorithm :

   Function PRIOR-SAMPLE (bn) returns an event sampled from the prior specified by bn.
   inputs : bn, a Bayesian network specifying joint distribution $\mathbf{P}(X_1, \ldots, X_n)$
   $X \leftarrow$ an event with n elements.
   for i = 1 to n do
   $x_i \leftarrow$ a random sample from $\mathbf{P}(X_i \mid \text{parent}(X_i))$
   return x.

5) **Applying operations of algorithm on the network in diagram 8.1.8 (a)** assuming an ordering [Cloudy, Sprinkler, Rain, WetGrass] :

   i) Sample from **P**(Cloudy) = <0.5, 0.5>; suppose this returns true.

   ii) Sample from **P**(Sprinkler | Cloudy = true) = <0.1, 0.9>; suppose this returns false.

   iii) Sample from **P**(Rain | Cloudy = true) = <0.8, 0.2>; suppose this returns true.

iv) Sample from P(WetGrass | Sprinkler = false, Rain = true) = <0.9, 0.1>; suppose this returns true.

In this case PRIOR-SAMPLE returns the **event [true, false, true, true].**

6) PRIOR-SAMPLE generates samples from the prior joint distribution specified by the nework. First, let $S_{ps}(x_1,....,x_n)$ be the probability that a specific event is generated by the PRIOR-SAMPLE algorithm. Just looking at the sampling process, we have,

$$S_{ps}(x_1,....,x_n) = \prod_{i=1}^{n} P(x_i | \text{parents}(X_i)).$$

Because each sampling step depends only on the parent values. This expression is also the probability of the event according to the Bayesian net's representation of the joint distribution. We have,

$$S_{ps}(x_1,....,x_n) = P(x_1,....,x_n)$$

7) In any sampling algorithm, the answers are computed by counting the actual samples generated. Suppose there are N total samples, and let $N(x_1,....,x_n)$ be the frequency of the specific event $x_1,....,x_n$. We expect this frequency to converge in the limit, to its expected value according to the sampling probability :-

$$\lim_{N \to \infty} \frac{N_{ps}(x_1,....,x_n)}{N} = S_{ps}(x_1,....,x_n) = P(x_1,....,x_n) \quad \ldots (8.1.4)$$

For example : Consider the event produced earlier : [true, false, true, true]. The sampling probability for this event is,

$$S_{ps}\text{ (true, false, true, true)} = 0.5 \times 0.9 \times 0.8 \times 0.9$$
$$= 0.324$$

Hence, in the limit of large N, we expect 32.4 % of the samples to be of this event.

8) Whenever we use an approximate equality ("≈"), we mean it in exactly this sense that the estimated probability becomes exact in the large sample limit. **Such an estimate is called consistent.**

For example : One can produce a consistent estimate of the probability of any partially specified event $x_1,....,x_n$, where $m \leq n$, as follows : -

$$P(x_1,....,x_m) \approx N_{ps}(x_1,....,x_m) / N \quad \ldots (8.1.5)$$

That is, the probability of the event can be estimated as the fraction of all complete events generated by the sampling process that match the partially specified event.

For example : If we generate 1000 samples from the sprinkler network, and 511 of them have, Rain = true, then the estimated probability of rain, written as P(Rain = true), is 0.511.

8.1.4.2 Rejection Sampling in Bayesian Networks

1) **Rejection sampling is a method for producing samples from a hard-to-sample distribution given an easy-to-simple distribution.**
2) It can be used to compute conditional probabilities ; that is, to determine $P(X \mid e)$.
3) The Rejection Sampling algorithm is,

function REJECTION-SAMPLING (x, e, bn, N) returns an estimate of $P(X \mid e)$

inputs : X, the query variable

e, evidence specified as an event.

bn, a Bayesian network.

N, the total number of samples to be generated.

local variables : N, a vector of counts over X, initially zero.

for j = 1 to N do

$X \leftarrow$ PRIOR-SAMPLE (bn)

if X is consistent with e is $N[x] \leftarrow N[x] + 1$ where x is the value of X in x.

return NORMALIZE (N[x]).

The rejection sampling algorithm for answering queries given evidence in a Bayesian network.

- **Working of algorithm :**
  - i) It generates samples from the prior distribution specified by the network.
  - ii) It rejects all those samples that do not match the evidence.
  - iii) Finally, the estimate $P(X = x \mid e)$ is obtained by counting how often $X = x$ occurs in the remaining samples.

4) Let $\mathbf{P}(X \mid e)$ be the estimated distribution that the algorithm returns. From the definition of the algorithm, we have,

$$P(X \mid e) = \alpha N_{ps}(X, e) = \frac{N_{ps}(X, e)}{N_{ps}(e)}$$

from equation (8.1.5) this becomes,

$$P(X \mid e) \approx \frac{P(x, e)}{P(e)} = P(x, e).$$

That is, **rejection sampling produces a consistent estimate of the rule probability.**

5) Applying operations of algorithm on the network in the diagram 8.1.8(a), let us assume that we wish to estimate **P**(Rain/Sprinkler = true), using 100 samples. Of the 100 that we generate, suppose 8 have Rain = true and 19 have Rain = false.

Hence,

P(Rain | Sprinkler = true) ≈ NORMALIZE

(<8.19>) = <0.296, 0.704 >

The true answer is <0.3, 0.7>

6) As more samples are collected, the estimate will converges to the true answer. The standard deviation of the error in each probability will be proportional to $1/\sqrt{n}$, where 'n' is the number of samples used in the estimate.

7) The biggest problem with rejection sampling is that it rejects so many samples ! The fraction of samples consistent with the evidence 'e' drops exponentially as the number of evidence variables grows, so the procedure is simply unusable for complex problems.

8.1.4.3 Likelihood Weighing in Bayesian Network

1) **Likelihood weighing avoids the inefficiency of rejection sampling by generating only events that are consistent with the evidence 'e'.**

2) The Likelihood Weighing algorithm is,

Function LIKELIHOOD-WEIGHING (X, e, bn, N) returns an estimate of P(X | e)

inputs : X, the query variable

e, evidence specified as an event

bn, a Bayesian network.

N, the total number of samples to be generated.

Local variables : W, a vector of weighted counts over X, initially zero.

for j = 1 to N do

X, w ← WEIGHTED-SAMPLE (bn)

W[X] ← W[x] + w where x is the value of X in x.

return NORMALIZE (W[X]).

Function WEIGHTED-SAMPLE (bn, e) returns an event and a weight

x ← an event with n elements; w ← 1.

for i = 1 to n do

if X, has a value x_i in e

then w ← wX $P(X_i = x_i | \text{parents}(X_i))$

else x_i ← a random sample from $P(X_i | \text{parents}(X_i))$

return X, w.

Note on likelihood weighing algorithm -

i) It fixes the values for the evidence variables E and samples only the remaining variables X and Y. This guarantees that each event generated is consistent with the evidence.

ii) Not all events are equal, however, before tallying the counts in the distribution for the query variable, each event is weighted by the likelihood that the event accords to the evidence, as measured by the product of the conditional probabilities for each evidence variable, given its parents.

iii) Intuitively, events in which the actual evidence appears unlikely should be given less weight.

3) Applying operations of algorithm on the network Fig. 8.1.8 (a), with the query **P**(Rain/Sprinkler = true, WetGrass = true). The process goes as follows :

i) The weight w is set to 1.0

ii) Now, an event is generated.

- Sample from **P**(Cloudy) = <0.5, 0.5>; suppose this returns true.
- Sprinkler is an evidence variable with value true. Therefore, we set
  $w \leftarrow w \times \mathbf{P}$ (Sprinkler = true | Cloudy = true) = 0.1
  Sample from **P**(Rain | Cloudy = true) = <0.8, 0.2>; suppose this returns true.
- WetGrass is an evidence variable with value true. Therefore, we set
  $w \leftarrow w \times$ P(Wet Grass = true | Sprinkler = true, Rain = true) = 0.099.

iii) Here WEIGHTED-SAMPLE returns the event [true, true, true, true] with weight 0.099, and this is tallied under Rain = true.

iv) The weight is low because the event describes a cloudy day, which makes the sprinkler unlikely to be on.

4) **Likelihood Weighting working :**

i) Examine the sampling distribution S_{ws} for WEIGHTED-SAMPLE.

ii) The evidence variables E are fixed with values 'e'.

iii) Call the other variables Z, that is, $Z = \{X\} \cup Y$.

iv) The algorithm samples each variable in Z, in given its parent values :

$$S_{ws}(Z, e) = \prod_{i=1}^{l} P(Z_i | \text{parents}(Z_i)) \quad \ldots (8.1.6)$$

Notice that Parents (Z_i) can include both hidden variables and evidence variables. Unlike the prior distribution P(Z), the distribution S_{ws} pays some attention to the evidence the sampled values for each Z_i will be influenced by evidence among Z_i's ancestors.

v) On the other hand, S_{ws} pays less attention to the evidence than does the true posterior distribution $P(Z|e)$, because the sampled values for each Z_i ignore evidence among Z_i's non ancestors.

vi) The likelihood weight w makes up for the difference between the actual and desired sampling distribution. The weight for a given sample 'x', composed from Z and 'e', is the product of the likelihood for each evidence variable given its parents (some or all of which may be among the Z_{is}) :

$$w(Z, e) = \prod_{i=1}^{m} P(e_i / \text{parents}(E_i)) \quad \ldots (8.1.7)$$

vii) Multiplying equations (8.1.4) and (8.1.5), we see that the weighted probability of a sample has the particularly convenient form,

$$S_{ws}(Z, e)\, W(Z, e) = \prod_{i=1}^{l} P(y_i | \text{parents}(Y_i))$$

$$\prod_{i=1}^{m} P(e_i | \text{parents}(E_i)) = P(y, e) \quad \ldots (8.1.8)$$

Because the two products cover all the variables in the network, allowing us to use equation (8.1.1) for the joint probability.

viii) It can be easy to show that likelihood weighting estimates are consistent. For any particular value x of X, the estimated posterior probability can be calculated as follows :

$$P(x|e) = \alpha \sum_{y} N_{ws}(x, y, e)\; w(x, y, e)$$

from LIKELIHOOD-WEIGHTING.

$$\approx \alpha' \sum_{y} S_{ws}(x, y, e)\; w(x, y, e)$$

from large N.

$$= \alpha' \sum_{y} P(x, y, e)$$ by equation (8.1.8)

$$= \alpha' P(x, e) = P(x|e)$$

Hence, likelihood weighting returns consistent estimates.

5) Performance of algorithm :

i) Likelihood weighting uses all the samples generated therefore it can be much more efficient than rejection sampling.

ii) It will, suffer a degradation in performance as the number of evidence variables increases.

iii) Because most samples will have very low weights and hence the weighted estimate will be dominated by the tiny fraction of samples that accord more than an infinitesimal likelihood to the evidence.

iv) The problem is exacerbated if the evidence variables occur late in the variable ordering, because then the samples will be the simulations that bear little resemblance to the reality suggested by the evidence.

8.1.4.4 Markov Chain Monte Carlo (MCMC) Algorithm for Inference in Bayesian Networks

Working of MCMC algorithm

1) MCMC generates each event by making a random change to the preceding event. It is therefore helpful to think of the network as being in a particular current state specifying a value for every variable.

2) The next state is generated by randomly sampling a value for one of the non - evidence variables X_i, conditioned on the current values of the variables in the Markov blanket of X_i.

3) MCMC therefore wonders randomly around the state space - the space of possible complete assignments flipping one variable at a time, but keeping the evidence variables fixed.

4) For example : Consider the query **P**(rain | Sprinkler = true, WetGrass = true) applied to the network of Fig. 8.1.8 (a). The evidence variables 'Sprinkler' and 'WetGrass' are fixed to their observed values and the hidden variables 'Cloudy' and 'Rain' are initialized randomly. Let us say to true and false respectively. Thus, the initial state is [true, true, false, true]. Now the following steps are executed repeatedly :

   a) Cloudy is sampled, given the current values of its Markov blanket variables : in this case, we sample from **P**(Cloudy | Sprinkler = true, Rain = false), (Shortly we will show how to calculate this distribution). Suppose the result is Cloudy = false. Then the new current state is [false, true, false, true].

   b) Rain is sampled, given the current values of its Markov blanket variables : In this case we sample from **P**(Rain | Cloudy = false, Sprinkler = true, WetGrass = true). Suppose this yields Rain = true. The new current state is [false, true, true, true].

5) Each state visited during this process is a sample that contributes to the estimate, for the query variable Rain. If the process visits 20 states where Rain is true and 60 states where Rain is false, then the answer to the query is NORMALIZE (<20, 60>) = <0.25, 0.75>).

6) The complete algorithm is shown as follows :-

Function MCMC-ASK (X, e, bn, N) returns an estimate of $P(X|e)$

local variables : N[X], a vector of counts over X, initially zero.

Z, the nonevidence variables in bn.

X, the current state of the network, initially copied from e.

initialize X with random values for the variables in Z.

for j = 1 to N do

$N[x] \leftarrow N[x] + 1$ where x is the value of X in x.

for each Z_i in Z do

sample the value of Z_i in X from $\mathbf{P}(Z_i|mb(Z_i))$ given the value of MB (Zi) in X return NORMALIZE (N[X]).

Why MCMC works

1) It should be noticed that the sampling process settles into a "dynamic equilibrium" in which the long-run fraction of time spent in each state is exactly proporitional to its posterior probability.

2) This remarkable property follows from the specific **transition probability** with which the process moves from one state to another, as defined by the conditional distribution given the Markov blanket of the variable being sampled.

3) Let $q(X \rightarrow X')$ be the probability that the process makes a transition from state X to X′. This transition probability defines what is called a **Markov chain** on the state space.

4) Now suppose that we run the Markov chain for 't' steps, and let $\pi_t(X)$ be the probability that the system is in state X at time 't'.

   - Similarly, let $\pi_{t+1}(X')$ be the probability of being in state X′ at time 't+1.'
   - Given $\pi_t(X)$
   - We can calculate $\pi_{t+1}(X')$ by summing for all states the system could be in at time 't'.
   - The probability of being in that state is the times the probability of making the transition to $X' \rightarrow \pi_{t+1}(X') = \sum_X \pi_t(X)\, q(X \rightarrow X')$

5) We will say that the chain has reached its **stationary distribution** if $\pi_t = \pi_{t+1}$.

   Call this stationary distribution π ; its defining equation is therefore,

$$\pi(X') = \sum_X \pi(X) \quad q(X \rightarrow X') \text{ for all } X' \qquad \ldots (8.1.9)$$

Under certain standard assumptions about the transition probability distribution q, there is exactly one distribution π satisfying this equation for any given q.

6) Above equation (8.1.8) can be read as saying that the expected "outflow" from each state (i.e., its current "population") is equal to the expected "inflow" from all the state.

One way to satisfy this relationship is, the expected flow between any pair of states is the same in both direction. This is the property of **detailed balance** :

$$\pi(X)\ q(X \to X') = \pi(X')\ q\ (X' \to X) \text{ for all } X, X'. \quad \ldots (8.1.10)$$

7) It can be shown that detailed balance implies stationarily simply by summing over X in equation (8.1.9). We have,

$$\sum_x \pi(X)\ q\ (X \to X') = \sum_x \pi(X')\ q\ (X' \to X) = \pi(X') \sum_x q\ (X' \to X) = \pi(X').$$

8) To show that the transition probability $q(X \to X')$ defined by the sampling step in MCMC-ASK satisfies the detailed balance equation with a stationary distribution equal to $P(X|e)$. (The true posterior distribution on the hidden variables).

This is done in two steps -

a) First, we will define a Markov chain in which each variable is sampled conditionally on the current values of all the other variables, and we will show that this satisfies detailed balance.

b) We will simply observe that, for Bayesian networks, doing that is equivalent to sampling conditionally on the variable's Markov blanket.

- Let X_i be the variable to be sampled, and let $\overline{X}_i$ be all the hidden variables other than X_i. Their values in the current state are x_i and $\overline{x}_i$. If we sample a new x_i' for X_i conditionally on all the other variables, including the evidence, we have ,

  $q(X \to X') = q(x_i, \overline{X}_i) \to (x_i', \overline{x}_i) = P(x_i' | \overline{x}_i, e)$.

- This transition probability is called **Gibbs sampler** and is a particularly convenient form of MCMC. Now we show that the Gibbs sampler is, in detailed balance with the true posterior.

  $\pi(X)\ q(X \to X') = P(X|e)\ P(x_i'|\overline{x}_i, e) = P(x_i, \overline{X}_i|e)\ P(x_i'|\overline{X}_i, e)$

  $= P(x_i|\overline{X}_i, e)\ P(\overline{X}_i|e)\ P(x_i'|\overline{X}_i, e)$ (using the chain rule of first term).

  $= P(x_i|\overline{X}_i, e)\ P(x_i', \overline{X}_i|e)$ (using the chain rule backwards)

  $= \pi(X')\ q\ (X' \to X)$.

- Note that, a variable is independent of all other variables given its Markov blanket; hence

  $P(x_i' + \overline{X}_i, e) = P(x' | mb(X_i))$

  where mb (X_i) denotes the values of the variables in X_i's Markov blanket, MB (X_i).

The probability of a variable, given its Markov blanket, is proportional to the probability of the variable given its parents times the probability of each child given its respective parents :-

$$P(x' \mid mb(X_i)) = \alpha\ P(x' \mid Parents(X_i)) \times \prod_{Y_j \in children(x_i)} P(y_j \mid Parents(Y_j)) \quad \ldots (8.1.11)$$

- Hence, to flip each variable X_i, the number of multiplications required is equal to the number of X_i's children.

9) We have discussed here simple variant of MCMC, namely the Gibbs sampler. In its most general form, MCMC is a powerful method for computing with probability models and many variants have been developed, including the simulated annealing algorithm presented.

8.2 Other Techniques for Uncertain Reasoning

GTU : Summer-18,19,20, Winter-19

There are two basic reasons why probability can fail :-

1) One common view is that probability theory is essentially numerical, whereas human judgemental reasoning is more "qualitative".
2) Probabilistic Reasoning did not scale up because of the exponential number of probabilities required in the full joint distribution.

8.2.1 Default Reasoning

- We can do qualitative reasoning using technique like default reasoning.
- Default reasoning treats conclusions not as "believed to a certain degree", but as "believed until a better reason is found to believe something else".
- We have discussed default reasoning in detail in chapter 7.

8.2.2 Rule-based

1) This approach hope to build on the success of logical rule-based systems, but add a sort of "Fudge factor" to each rule to accommodate uncertainty. These methods were developed in the mid-1970s and formed the basis for a large number of expert systems in medicine and other areas.
2) Rule-based systems emerged from early work on practical and intuitive systems for logical inference.
3) Logical systems in general, and logical rule-based systems in particular have three desirable properties :
   i) **Locality :** In logical systems, whenever we have a rule of the form $A \Rightarrow B$, we conclude B, given evidence A, without worrying about any other rules. In probabilistic systems, we need to consider all the evidence in the Markov blanket.

ii) **Detachment** : Once a logical proof is found for a proposition B, the proposition can be used regardless of how it was derived. That is, it can be detached from its justification. In dealing with probabilities, on the other hand, the source of the evidence for a belief is important for subsequent reasoning.

iii) **Truth functionality** : In logic, the truth of complex sentences can be computed from the truth of the components. Probability combination does not work this way, except under strong global independence assumptions.

4) There have been several attempts to devise uncertain reasoning schemes that retain these advantages. The idea is to attach degrees of belief to propositions and rules and to devise purely local schemes for combining and propagating those degrees of belief. The schemes are also truth functional.

For example : The degree of belief in $A \vee B$ is a function of the belief in A and the belief in B.

5) Problem associated with rule based methods :-

i) The properties of locality, detachment, and truth-functionality are simply not appropriate for uncertain reasoning.

ii) Let us look at truth functionality first. Let H_1, be the event that a fair coin flip comes up heads. Let T_1 be the event that the coin comes up tails on that same flip, and let H_2 be the event that the coin comes up heads on a second flip. Clearly, all three events have the same probability, 0.5, and so a truth functional system must assign the same belief to the disjunction of any two of them. But we can see that the probability of the disjunction depends on the events themselves and not just on their probabilities.

P (A)	P (B)	P (A ∨ B)
$P(H_1) = 0.5$	$P(H_1) = 0.5$	$P(H_1 \vee H_1) = 0.50$
	$P(T_1) = 0.5$	$P(H_1 \vee T_1) = 1.00$
	$P(H_2) = 0.5$	$P(H_1 \vee H_2) = 0.75$

It gets worse when we chain evidence together. Truth-functional systems have rules of the form $A \rightarrow B$ that allow us to compute the belief in B as a function of the belief in the rule and the belief in A. Both forward and backward-chaining systems can be devised. **The belief in the rule is assumed to be constant and is usually specified by the knowledge engineer.**

For example : as $A \rightarrow_{0.9} B$

Consider the WetGrass situation. If we wanted to be able to do both causal and diagnostic reasoning, we would need the two rules,

Rain → WetGrass and WetGrass → Rain

These two rules form a feedback loop : Evidence for Rain increases the belief in WetGrass, which in turn increases the belief in Rain even more. Clearly, uncertain reasoning systems need to keep track of the paths along which evidence is propagated.

Inter-causal reasoning (for explaining away) is also tricky. Consider what happens when we have the two rules,

Sprinkler → WetGrass and WetGrass → Rain.

Suppose we see that the Sprinkler is on. Chaining forward through our rules, this increases the belief that the grass will be wet, which in turn increases the belief that it is raining. But this is ridiculous; the fact that the sprinkler is on explains away the wet grass and should reduce the belief in rain. A truth functional system acts as if it also believes Sprinkler → Rain.

6) If task is restricted and rules are engineered carefully then truth-functional systems work well.

8.2.3 Ignorance

1) One area that we have not addressed so far is the question of ignorance, as opposed to uncertainty. Consider the flipping of a coin. If we know that the coin is fair, then a probability of 0.5 for heads is reasonable. If we know that the coin is biased, but we do not know which way, then 0.5 is the only reasonable probability. The two cases are different, yet probability seems not to distinguish them. The Dempster-Shafer theory uses interval-valued degrees of belief to represent an agent's knowledge of the probability of a proposition.

2) Representing ignorance using Dempster-Shafer theory :-

   i) The Dempster-Shafer theory is designed to deal with the distinction between **uncertainty** and **ignorance**.

   ii) Rather than computing the probability of a proposition, it computes the probability that the evidence supports the propositions.

   iii) This measure of belief is called a **belief function**, written Bel(X).

   iv) We return to coin flipping for an example of belief functions. Suppose a shady character comes up to you and offers to bet you ₹ 10 that this coin will come up heads on the next flip. Given that the coin might or might not be fair, what belief should you ascribe to the event that it comes up heads? Dempster-Shafer theory says that because you have no evidence either way,

you have to say that the belief Bel(Heads) = 0 and also that Bel(¬ Heads) = 0. This make Dempster-Shafer reasoning systems skeptical in a way that has some intuitive appeal.

v) Now suppose you have an expert at your disposal who testifies with 90 % certainty that the coin is fair (i.e he is 90 % sure that P(Heads) = 0.5). Then Dempster-Shafer theory gives Bel(Heads) = 0.9×0.5 = 0.45 and likewise Bel (¬ Heads) = 0.45. There is still a 10 percentage point "gap" that is not accounted for by the evidence.

vi) 'Dempster's rule' (Dempster, 1968) shows how to combine evidence to give new values for Bel, and Shafer's work extends this into a complete computational model.

vii) Problems associated with Dempster-Shafer theory :-

1. There is a problem in connecting beliefs to actions.
2. With probabilities, decision theory says that if P(Heads) = P(¬ Heads) = 0.5 then (assuming that winning ₹ 10 and losing ₹ 10 are considered equal magnitude opposites). The reasoner will be indifferent between the action of accepting and declining the bet.
3. The Dempster-Shafer reasoner has Bel(¬ Heads) = 0 and thus no reason to accept the bet, but then it also has Bel(Heads) = 0 and thus no reason to decline it. Thus, it seems that the Dempster-Shafer reasoner comes to the same conclusion about how to act in this case.
4. Unfortunately, Dempster-Shafer theory allows no definite decision in many other cases where probabilistic inference does yield a specific choice.

viii) One inter-pretation of Dempster-Shafer theory is that it defines a probability interval, the interval for Heads is [0,1] before our expert testimony and [0.45, 0.55] after. The width of the interval might help in deciding when we need to acquire more evidence. It can tell you that the expert's testimony will help you if you do not know whether the coin is fair, but will not help you if you have already learned that the coin is fair. However, there are no clear guidelines for how to do this, because there is no clear meaning for what the width of an interval means.

ix) In the Bayesian approach, this kind of reasoning can be done easily by examining how much one's belief would change if one were to acquire more evidence.

For example : Knowing whether the coin is fair would have a significant impact on the belief that it will come up heads, and detecting an asymmetric weight would have an impact on the belief that the coin is fair. A complete Bayesian model would include probability estimates for factors such as these,

allowing us to express our "ignorance" in terms of how our beliefs would change in the face of future information gathering.

8.2.4 Vagueness

1) Probability makes the same ontological commitment as logic :- That events are true or false in the world, even if the agent is uncertain as to which is the case. Researchers in fuzzy logic have proposed an ontology that allows vagueness :- That an event can be "Sort of" true. Vagueness and uncertainty are in fact orthogonal issues.

2) Representing vagueness : **(Fuzzy sets and fuzzy logic)** :

   i) Fuzzy set theory is used for specifying how well an object satisfies a vague description.

   ii) For example : Consider the proposition "Anil is tall". Is this true, if Anil is 5'10"? most people would hesitate to answer "true" or "false", preferring to say, "sort of".

   iii) Note that this is not a question of uncertainty about the external world. We are sure of Anil's height. The issue is that the linguistic term "tall" does not refer to a sharp demarcation of objects into two classes and there are degrees of tallness.

   iv) Due to this reason, fuzzy set theory is not a method for uncertain reasoning at all. Rather, fuzzy set theory treats Tall as a fuzzy predicate and says that the truth value of Tall(Anil) is a number between 0 and 1, rather than being just true or false.

   v) The name "fuzzy set" derives from the interpretation of the predicate as implicitly defining a set of its members - a set that does not have sharp boundaries.

   vi) **Fuzzy logic :**

   1. Fuzzy logic is a method for reasoning with logical expressions describing membership in fuzzy sets.
   2. For example : The complex sentence Tall(Anil) $\wedge$ Heavy(Anil) has a fuzzy truth value that is a function of the truth values of its components.
   3. The standard rules for evaluating the fuzzy truth, T, of a complex sentence are

      $T(A \wedge B) = \min(T(A), T(B))$

      $T(A \vee B) = \max(T(A), T(B))$

      $T(\neg A) = 1 - T(A)$

4. Fuzzy logic is therefore a truth-functional system - a fact that causes serious difficulties.
5. For example : Suppose that T (Tall(Anil)) = 0.6 and T (Heavy(Anil)) = 0.4. Then we have T (Tall(Anil)) $\wedge$ T (Heavy(Anil)) = 0.4. Which seems reasonable but we also get the result T (Tall(Anil)) $\wedge \neg$ (Tall(Anil)) = 0.4 which does not.
6. The problem arises from the inability of a truth-functional approach to take into account the correlations or anti-correlations among the component propositions.

vii) **Fuzzy control :**

1. Fuzzy control is methodology for constructing control systems in which the mapping between real-valued input and output parameters is represented by fuzzy rules.
2. Fuzzy control has been very successful in commercial products such as automatic transmissions, video cameras, and electric shavers.
3. These applications enjoy success may be because they have small rule bases, no chaining of inferences and tunable parameters that can be adjusted to improve the system's performance. The fact that they are implemented with fuzzy operators might be incidental to their success; the key is simply to provide a concise and intuitive way to specify a smoothly interpolated, real-world function.

viii) fuzzy logic can be represented in terms of probability theory. One idea is to view assertions such as "Anil is Tall" as discrete observations, made concerning a continuous hidden variable, Anil's actual height. The probability model specifies P(observer says Anil is tall/Height), using a probit distribution. A posterior distribution over Anil's height can then be calculated in the usual way, for example if the model is part of a hybrid Bayesian Network.

ix) Fuzzy predicates can also be given a probabilistic interpretation in terms of random sets - that is, random variables whose possible values are sets of objects.

For example : Tall is random set whose possible values are sets of people. The probability P(Tall = S_1), where S_1 is some particular set of people, is the probability that exactly the set would be identified as "tall" by an observer. Then the probability that "Anil is tall" is the sum of the probabilities of all the sets of which Anil is a member.

x) Both the hybrid Bayesian network approach and the random sets approach appear to capture aspects of fuzziness without introducing degrees of truth. But they can not handle proper representation of linguistic observations and continuous quantities that have been neglected by most outside the fuzzy community.

Answer in Brief

1. *Explain in detail probabilistic reasoning system with example.* **(Refer section 8.1.1)**
2. *Explain method of handling approximate inference in Bayesian networks.* **(Refer section 8.1.4)**
3. *Explain the need of fuzzy set and fuzzy logic with example.* **(Refer section 8.2.1)**
4. *Define Dempster-Shafer theory.*

Ans. : The Dempster-Shafer theory is designed to deal with the distinction between uncertainty and ignorance.

Rather than computing the probability of a proposition, it computes the probability the evidence that supports the proposition.

5. *Define : Baye's theorem.*

Ans. : In probability theory and applications, Baye's theorem (alternatively called as Baye's law or Bayes rule) links a conditional probability to its inverse.

$$P(b \mid a) = \frac{P(a \mid b)\, P(b)}{P(a)}$$

This equation is called as Baye's Rule or Baye's Theorem.

6. *What is reasoning by default ?*

Ans. : We can do qualitative reasoning using technique like default reasoning.

Default reasoning treats conclusions not as "believed to a certain degree", but as "believed until a better reason is found to believe something else".

7. *What are the logics used in reasoning with uncertain information ?*

Ans. : There are two approaches that can be taken for reasoning with uncertain information in which logic is used.

Non-monotonic logic is used in default reasoning process. Default reasoning also uses other type or logic called as default logic.

The second approach towards reasoning is vagueness which uses fuzzy logic. Fuzzy logic is a method for reasoning with logical expressions describing membership in fuzzy sets.

8. *Define prior probability.*

Ans. : The prior (unconditional) probability is associated with a proposition 'a'.

The prior probability 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 the prior priobility is written as,

P(Cavity = true) = 1 or P(cavity) = 0.1

9 State the types of approximation methods.

Ans. : For approximate inferencing randomize sampling algorithm (Monte Carlo Algorithm) is used. There are two approximation methods that are used in randomize sampling algorithm which are 1) Direct sampling algorithm and 2) Markov chain sampling algorithm.

In direct sampling algorithm samples are generated from known probability distribution. In Markov chain sampling each event is generated by making a random change to the preceding event.

10. What do you mean by hybrid Bayesian network ?

Ans. : A network with both discrete and continuous variables is called as hybrid Bayesian network. In hybrid Bayesian network, for representing the continuous variable its discretization is done in terms of intervals because it can have infinite values.

For specifying the hybrid network two kinds of distribution are specified. The conditional distribution for a continuous variable given discrete or continuous parents and the conditional distribution for a discrete variable given continuous parent.

11. Define computational learning theory.

Ans. : The computational learning theory is a mathematical field related to the analysis of machine learning algorithms.

The computational learning theory is used in the evaluation of sample complexity and computational complexity. Sample complexity targets the issue that, how many training examples are needed to learn a successful hypothesis ? The computational complexity evaluates that how much computational effort is needed to learn a successful hypothesis ?

In addition to performance bounds, computational learning theory also deals with the time complexity and feasibility of learning.

12. Give the full specification of Bayesian network.

Ans. : 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 :-

1) A set of random variables makes up the nodes of the network. Variables may be discrete or continuous.
2) 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.
3) Each node X_i, has a conditional probability distribution $P(X_i \mid Parents(X_i))$ that quantifies the effect of the parents on the node.
4) 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**.

8.3 University Questions with Answers

Summer - 18

Q.1 *Differentiate Fuzzy logic and Crisp logic.* **(Refer sections 8.2)** [3]

Q.2 *Discuss various defuzzification methods.* **(Refer section 8.2)** [4]

Q.3 *Discuss Bayesian network and its application.* **(Refer sections 8.1)** [4]

Summer - 19

Q.4 *Differentiate fuzzy logic and crisp logic. Also describe set operations on fuzzy and crisp logic.* **(Refer section 8.2)** [7]

Q.5 *Discuss various defuzzification methods.* **(Refer section 8.2)** [4]

Q.6 *Discuss Bayesian network and its application.* **(Refer section 8.1)** [4]

Winter - 19

Q.7 *Explain probabilistic inference in Bayesian networks with the help of a suitable example.* **(Refer section 8.1)** [4]

Q.8 *Explain the difference between Boolean and fuzzy set membership using a suitable example.* **(Refer section 8.2)** [3]

Summer - 20

Q.9 *What is the importance of fuzzy logic ? How do you perform union, intersection and complement operation on the fuzzy sets ?* **(Refer section 8.2)** [4]

❑❑❑

9 Game Playing

Syllabus

Overview, MiniMax, Alpha-Beta Cut-off, Refinements, Iterative deepening.

Contents

9.1 What is Game ? - The Adverserial Search

GTU : Summer-12, 14, 15, 18, Winter-12, 15, 17

- **In adversarial search problem environment is multiagent, competitive where in the agent's goals are in coflict.** Adversarial search problems are commonly known as **games**.
- Mathematical game theory a branch of economics views any multiagent environment as a game, provided that, impact of each agent on the others is "significant" regardless of whether the agent are cooperative or competitive.
- The term game means a sort of conflict in which n individuals or groups (known as players) participate.
- Game theory denotes games of strategy.
- **John Von Neumann** is acknowledged as **father of game theory.** Neumann defined game theory in 1928 and established the mathermatical framework for all subsequent theoretical developments.
- Game theory allows decision-makers (players) to cope with other decision-maker (players) who have different purposes in mind. In other words, players determine their own strategies in terms of the strategies and goal of their opponent.
- Games are integral attribute of human beings. Games engage the intellectual faculties of humans.
- If computers are to mimic people they should be able to play games.

Game playing has close relation to intelligence and it has well-defined states and rules.

9.2 Applications of Game Theory

- Applications of game theory are wide-ranging. **Von Neunmann** and **Morgenstern** indicated the utility of game theory by linking with ecomomic behavior.

1. Economic models

- For markets of various commodities with differing numbers of buyers and sellers, fluctuating values of supply and demand, seasonal and cyclical variations, analysis of conflicts of interest in maximizing profits and promoting the widest distribution of goods and services.

2. Social sciences

- The n-person game theory has interesting uses in studying the distribution of power in legislative procedures, problems of majority rule, individual and group decision making.

3. Epidemiologists

- Make use of game theory, with respect to immunization procedures and methods of testing a vaccine or other medication.

4. Military strategists

- Turn to game theorey to study conflicts of interest resolved through "battles" where the outcome or payoff of a war game is either victory or defeat.

9.3 Definition of Game

1. A game **has at least two** players. **Solitaire is not considered a game by game theory.**

   The term "solitaire" is used for single-player games of concentration.

2. An instance of a game begins with a player, choosing from a set of specified (game rules) alternatives. **This choice is called a move.**

3. After first move, the new situation determines which player to make next move and alternatives available to that player.

   i) In many board games, the next move is by other player.

   ii) In many multi-player card games, the player making next move depends on who dealt, who took last trick, won last hand, etc.

4. The moves made by a player may or may not be known to other players. Games in which all moves of all players are known to everyone are called **games of perfect information.** For example,

   i) Most board games are games of perfect information.

   ii) Most card games are not games of perfect information.

5. Every instance of the game must end.

6. When an instance of a game ends, each player receives a payoff. A **payoff** is a value associated with each player's final situation. A zero-sum game is one, in which elements of payoff matrix sum to zero. In typical zero-sum game :

   i) Win = 1 point,

   ii) Draw = 0 point,

   iii) Loss = – 1 point

9.4 Game Theory

- Game theory does not prescribe a way or say how to play a game. Game theory is a set of ideas and techniques for analyzing conflict situations between two or more parties. The outcomes are determined by their decisions.

General Game Theorem

- In every two players, game like zero sum, non-random, perfect knowledge game there exists a perfect strategy guaranteed to at least result in a tie game.

The frequently used terms in game theory : -

- The term "**game**" means a sort of conflict in which n individuals or groups (known as players) participate.
- A list of "**rules**" stipulates the conditions under which the game begins.
- A game is said to have "**perfect information**" if all moves are known to each of the palyers involved.
- A "**strategy**" is a list of the optimal choices for each player at every stage of a given game.
- A "**move**" is the way in which game progresses from one stage to another, beginning with an initial state of the game to the final state.
- The total number of moves constitute the entirety of the game.
- The **payoff** or **outcome**, referes to what happens at the end of a game.
- **Minimax : The least good of all good outcomes.**
- **Maximin : The least bad of all bad outcomes.**

- **The important and basic game theory theorem is the mini-max theorem. This theorem says,**

  "If a minimax of one player corresponds to a maximin of the other player, then that outcome is the best that both players can hope for."

9.5 Relevance of Game Theory and Game Playing

How relevant the game theory is to mathematics, computer science and economics is shown in the figure below :

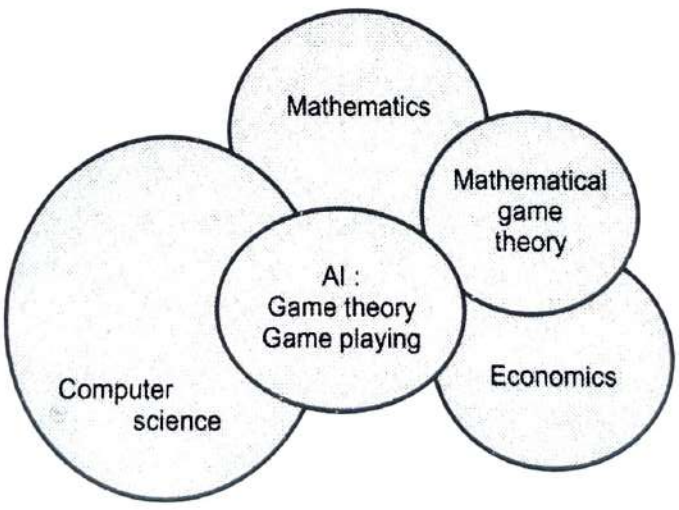


Fig. 9.5.1 Game theory and its relevance to other fields

9.6 Game Playing

Characteristics of Game Playing : -

1. There is always an "unpredictable" opponent
   - Opponent introduces uncertainty.
   - Opponent also wants to win.
2. Time limit

Games are always played in time constraint environment. Therefore is needs to handle time efficiently.

9.7 Types of Games

1. Based on chance

i) Deterministic (not involving chance)

For example -

Chess, Checkers, Tic-tac-toe

ii) Non-deterministic (can involve chance)

For example -

Backgammon, Monopoly.

2. Based on information

i) Perfect information -

Here all moves of all players are known to everyone.

For example -

Chess, Checker, Tic-tac-toe.

ii) Imperfect information -

Here all moves are not known to everyone.

For example -

Bridge, Pocker, Scrabble.

3. General zero-sum games

Players must choose their strategies simultaneously, neither knowing what the other player is going to do.

For example -

If you play a single game of chess with someone, one person will lose and one person will win. The win (+1) added to the loss (– 1) equals zero.

4. Constant - sum game

Here the algebraic sum of the outcomes are always constant, though not necessarily zero.

It is strategically equivalent to zero-sum games.

5. Non-zero - sum game

Here the algebraic sum of the outcomes are not constant. In these games, the sum of the payoffs are not the same for all outcomes.

They are not always completely solvable but provide insights into important areas of inter-dependent choice.

In these games, one player's losses do not always equal another player's gains.

The non-zero-sum games are of two types : -

i) Negative sum games (Competitive) -

Here nobody really wins, rather everybody loses.

Example - A war or a strike.

ii) Positive sum games (Co-operative) -

Here all players have one goal that they contribute together.

Example - An educational game, building blocks, or a science exhibit.

6. N-person game

It involves more than two players.

Analysis of such games is more complex than zero-sum games.

Conflicts of interest are less obvious.

9.8 Formal Representation of a Game as a Problem

9.8.1 A Game is Essentially a Kind of a Search Problem !

Game is formally defined with following four components : -

1. The initial state, which includes the board position and identifies the player to move.
2. A successor function, which returns a list of (move, state) pairs, each indicating a legal move and the resulting state.
3. A terminal test, which determines when the game is over. States where the game has ended are called as **terminal states.**
4. A utility function (also called an objective function or payoff function), which gives a numeric value for the terminal states. In chess, the outcome is a win, loss

or draw, with values +1, – 1, or 0. Some game have a wider variety of possible outcomes; the payoffs in backgammon range from + 192 to – 192.

9.8.2 Mixed Strategies

- A players's strategy in a game is a complete plan of action for whatever situation might arise. It is a complete algorithm for playing the game, telling a player what to do for every possible situation throughout the game.
- **A pure strategy** provides a complete definition of how a player will play a game. In perticular, it determines the move a player will make for any situation they could face. A player's strategy set, is the set of pure strategies available to that player.
- **A mixed strategy** is an assignment of a probabiblity to each pure strategy. This allows for a player to randomly select a pure strategy. Since probabilities are contineous, there are infinitely many mixed strategies available to a player, even if their strategy set is finite.
- A mixed stretegy for a player is a probability distribution, on the set of his pure strategies.

9.8.3 The Game Tree

The initial state and the legal moves for each side, define the game tree for the game.

- **Description of the game tree** [Refer Fig. 9.8.1 on next page].

1. Root node -

Represents board configuration and decision, required as to what is the best single next move.

If my turn to move, then the root is labeled a MAX node indicating it is my turn;

Otherwise it is labeled a MIN, node to indicate it is my opponent's turn.

2. Arcs -

Represent the possible legal moves for the player that the arcs emanate from.

3. At each level, the tree has nodes that are all MAX or all MIN.

4. Since moves alternate, the nodes at level 'i' are of opposite kind from those at level i + 1.

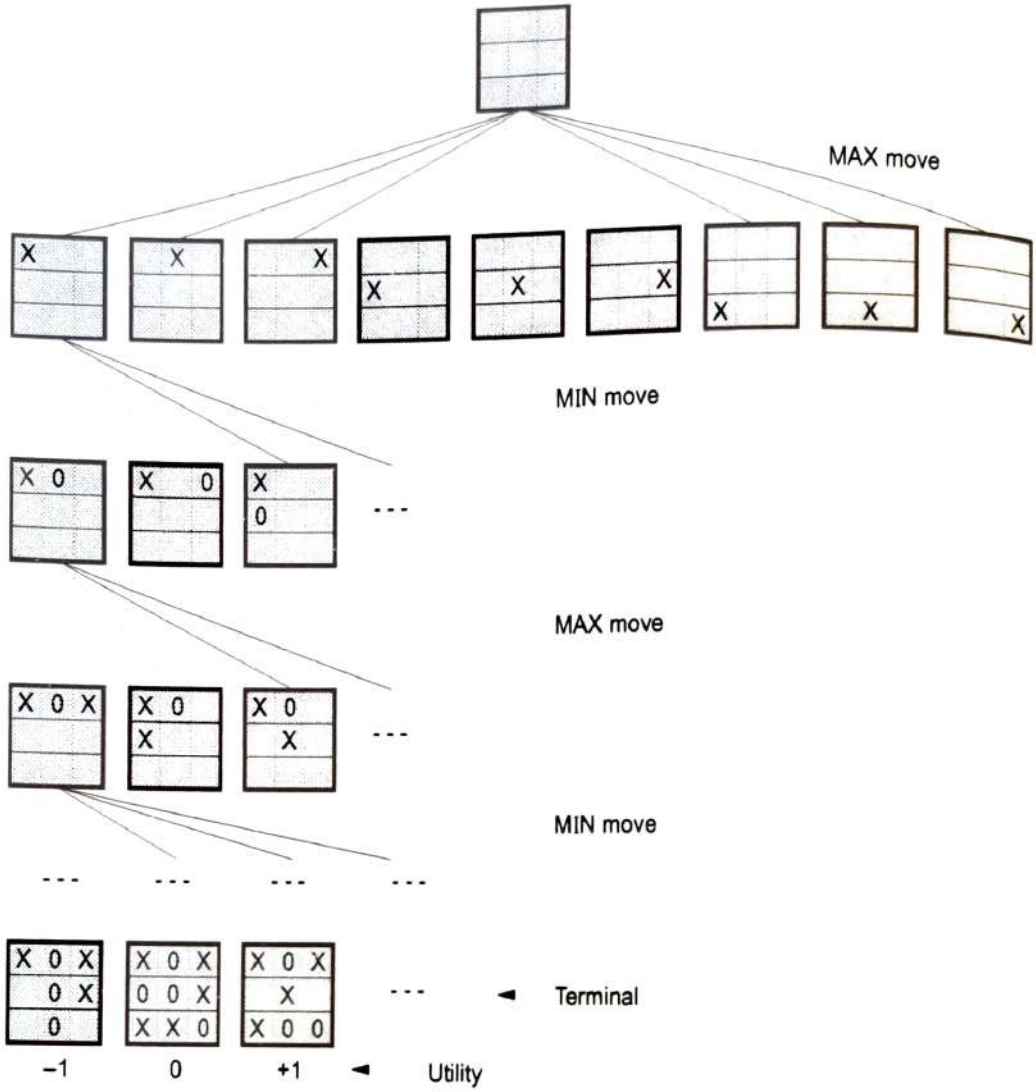


Fig. 9.8.1 Example of game tree

9.9 Game Strategy

9.9.1 Concept of Strategy

A player's strategy in a game, is a complete plan of action for whatever situation might arise. It is the complete description of how one will behave under every possible circumstances. You need to analyze the game mathematically and create a table with "outcomes" listed for each strategy.

A Two Player Strategy Table.

Players Strategies	Player A Strategy 1	Player A Strategy 2	Player A Strategy 3	etc.
Player B strategy 1	Tie	A wins	B wins	...
Player B strategy 2	B wins	Tie	A wins	...

Player B strategy 3	A wins	B Wins	Tie	...
etc.	...	...	...	...

1. Above is a partial tree for the game of tic-tac-toe.
2. The top node (root) is the initial state and MAX (player 1) moves first, placing X in an empty square.
3. The rest of the search tree shows alternate moves for MIN (player 2) and MAX.
4. Terminal states are assign utilities according to the rules of game.

9.9.2 Optimal Strategies

It is a techniques which always leads to superior solution than any other strategy, as opponent is playing in perfect manner.

Roughly speaking, an optimal strategy leads to outcomes that are at least as good as any other strategy when one is playing an infallible opponent.

Mini-Max Value

The minimax value for a given game tree is determined by optimal strategy by evaluating minimax value of each node.

Mini-Max Theorem

Players adopt those strategies which will maximize their gains, while minimizing their losses. Therefore the solution is, the best each player can do for him/herself in the face of opposition of the other player.

Determining Optimal Strategy

1. Given a game tree the optimal strategy can be determined by examining the minimax value of each node.
2. The minimax value of a node is the utility (for player called MAX) of being in the corresponding state, assuming that both players play optimally from this stage to the end of the game.
3. The minimax value of a terminal state is just its utility.
4. Given a choice, MAX will perfer to move to a state of maximum value, whereas MIN prefers a state of minimum value.

9.10 Mini-Max Algorithm

GTU : Winter-12,15,18,19, Sumemr-14,18,19,20

The minimax algorithm computes the minimax decision from the current state. It is used as a searching technique in game problems. The minimax algorithm performs a complete depth-first exploration of the game-tree.

9.10.1 The Algorithm

1. The start node is MAX (player 1) node with current board configuration.
2. Expand nodes down (play) to some depth of look-ahead in the game.
3. Apply evaluation function at each of the leaf nodes.
4. "Back up" values for each non-leaf nodes until computed for the root node.
5. At MIN (player 2) nodes, the backed up value is the minimum of the values associated with its children.
6. At MAX nodes, the backed up value is the maximum of the values associated with its children.

Note :

The process of "backing up" values gives the optimal strategy, that is, both players assume that your opponent is using the same static evaluation function as you are.

9.10.2 Properties of Mini-Max

1. Minimax provides a complete solution for finite tree.
2. Minimax provides optimal strategy against an optimal opponent.
3. The time complexity is $O(b^m)$.
4. The space complexity is O(bm) for depth-first exploration where algorithm generates all successor at once, or O(m) for an algorithm that generate successors one at a time.

9.10.3 Problem Associated with Mini-Max

This algorithm explores whole search space. If we have a game with huge search spaces it will take long time.

In such cases then exact solution is completely infeasible.

9.10.4 Game Playing with Mini-Max-Tic-Tac-Toe [Noughts and Crosses] Example

1. Assume that two players named MIN and MAX who are playing the game.
2. MAX is playing first.
3. As you can see initially MAX has nine possbile moves.
4. Play alternates between MAX and MIN until we reach leaf nodes corresponding to terminal states such that one player has 3 in a row or all the squares are filled.
5. The number on each leaf node indicates the utility value of the terminal state from the point of view of MAX.
6. High values are assumed to be good for MAX and bad for MIN.
7. It is MAX's job to make use of search tree to determine best move.
8. Static evaluation. Criteria - '+ 1' for a win, '0' for draw.

9.10.5 Example to Show How Mini-Max Algorithm Works

Consider game of tic-tac-toe with the initial state -

0	0	X
X		0
		X

Step 1 -

Start : MAX (player 1) moves (MAX is making X)

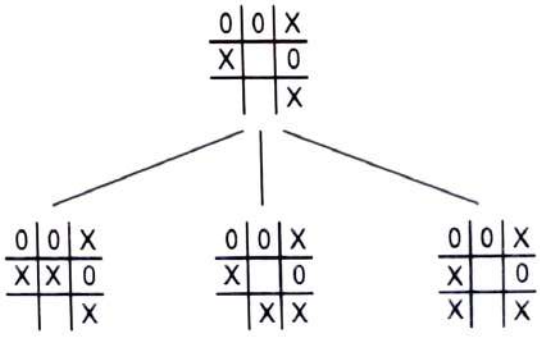


Fig. 9.10.1 Max playing (X)

Step 2 -

Next : MIN (player 2) moves.

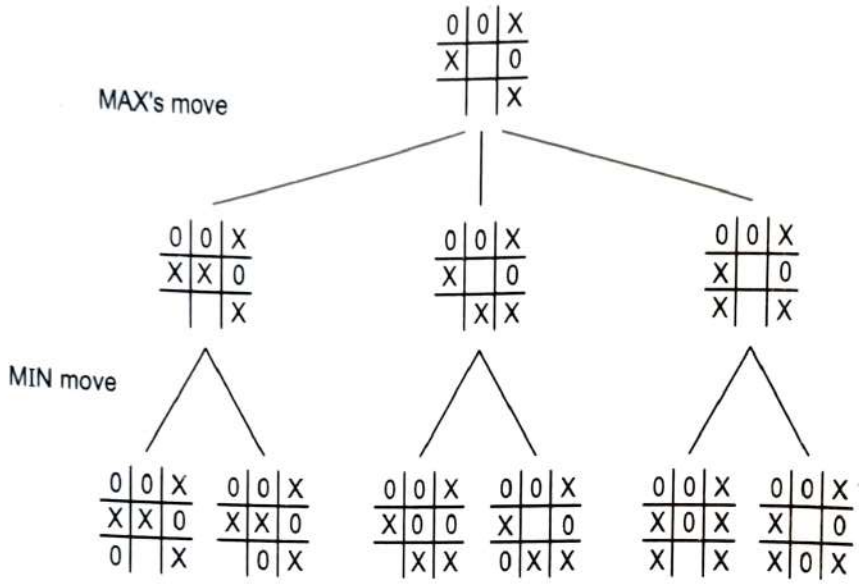


Fig. 9.10.2 Min playing (0)

Step 3 -

Again : MAX's moves

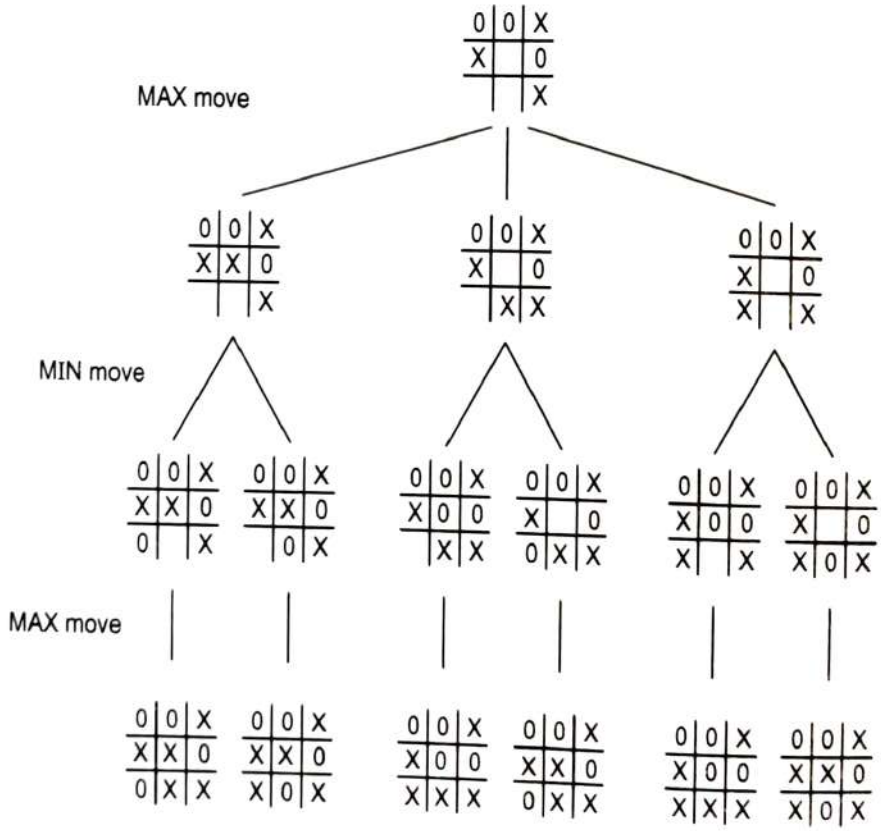


Fig. 9.10.3 Max playing (X)

Step 4 -

'+ 1' for a win, '0' for a draw. Criteria '+ 1' for a win, '0' for a draw.

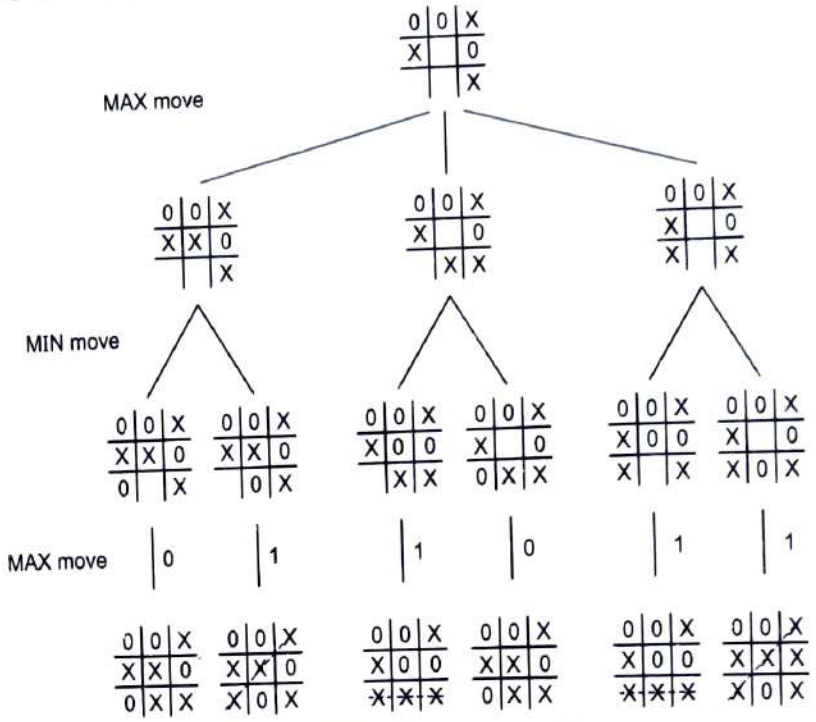


Fig. 9.10.4 Min playing (0)

Step 5 -

Level by level, on the basis of opponents turn UP : One level

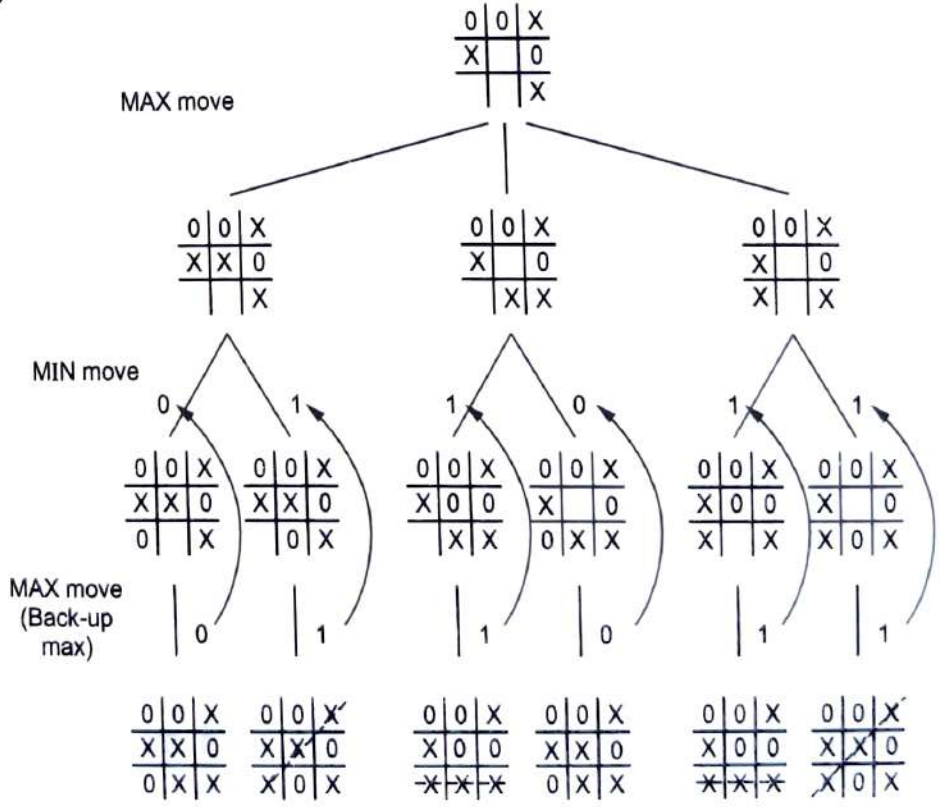


Fig. 9.10.5 One level up

Step 6 -

UP : Two levels

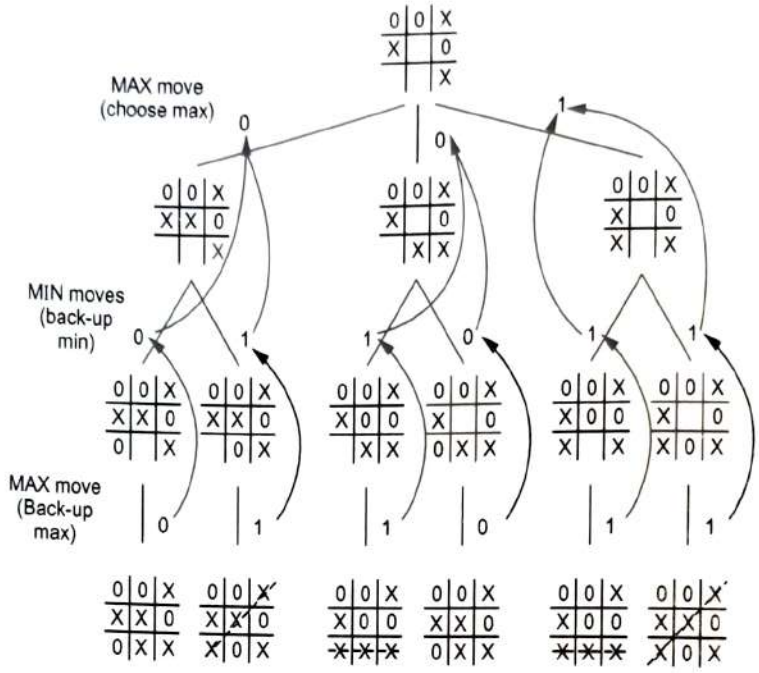


Fig. 9.10.6 One level up

Step 7 - Choose best move which is maximum

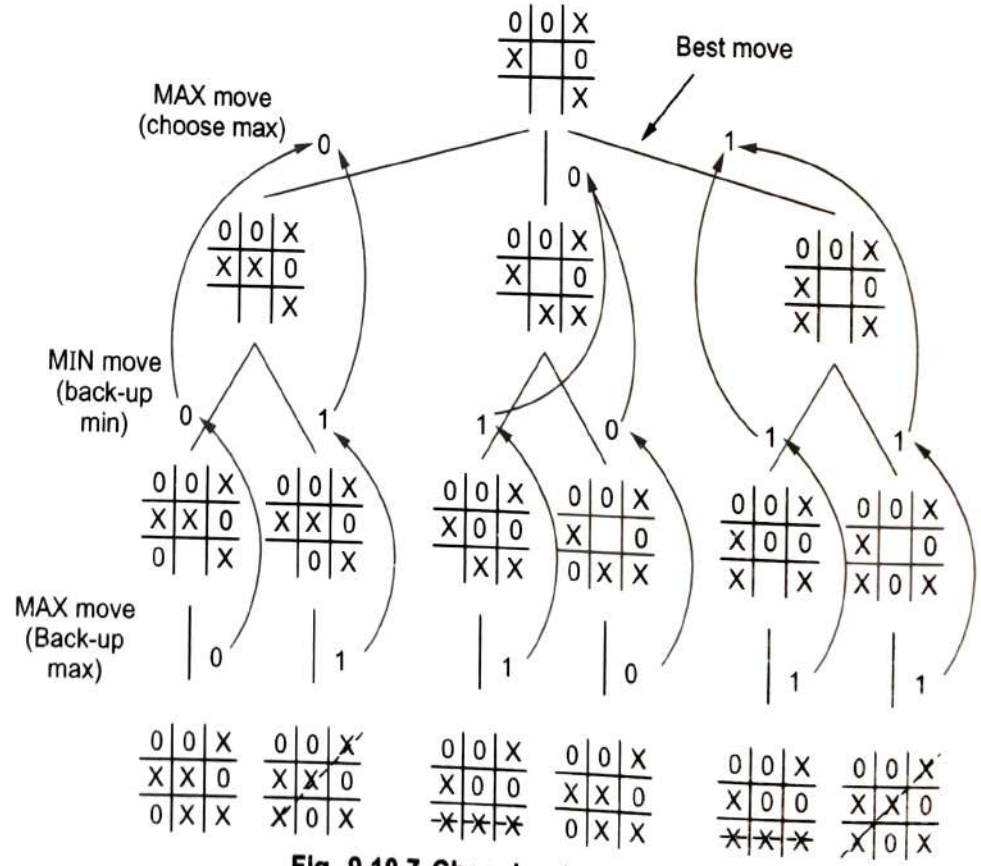


Fig. 9.10.7 Choosing best move

9.10.6 The 'made-up' Games and Concept of 'ply'

- For study purpose we can form a game a tree which is constructed up to certain level (i.e. upto certain depth). This is called as a **made-up game.** The term ply refers to depth of the tree.
- For example - ply 4 is level at depth 4 **below the root node.**

Example

A two ply game tree, for game of tic-tac-toe.

1. Assume that two players named MIN and MAX who are playing the game.
2. MAX is playing first.
3. The possible moves for MAX at the root node are labeled a_1, a_2 and a_3.
4. The possible replies to a_1 for MIN are b_1, b_2, b_3 and so on.
5. This particular game ends after one move each by MAX and MIN.
6. In game parlance, we say that this tree is one which, moves deep consisting of two-half-moves, each of which is called a **ply**.

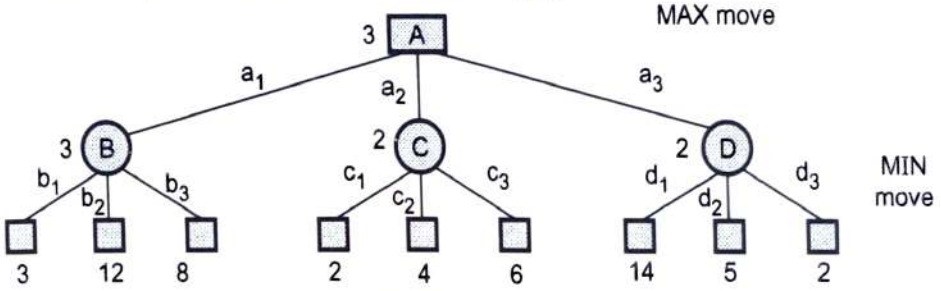


Fig. 9.10.8 Game tree

Example

A three-ply game tree, for game of tic-tac-toe

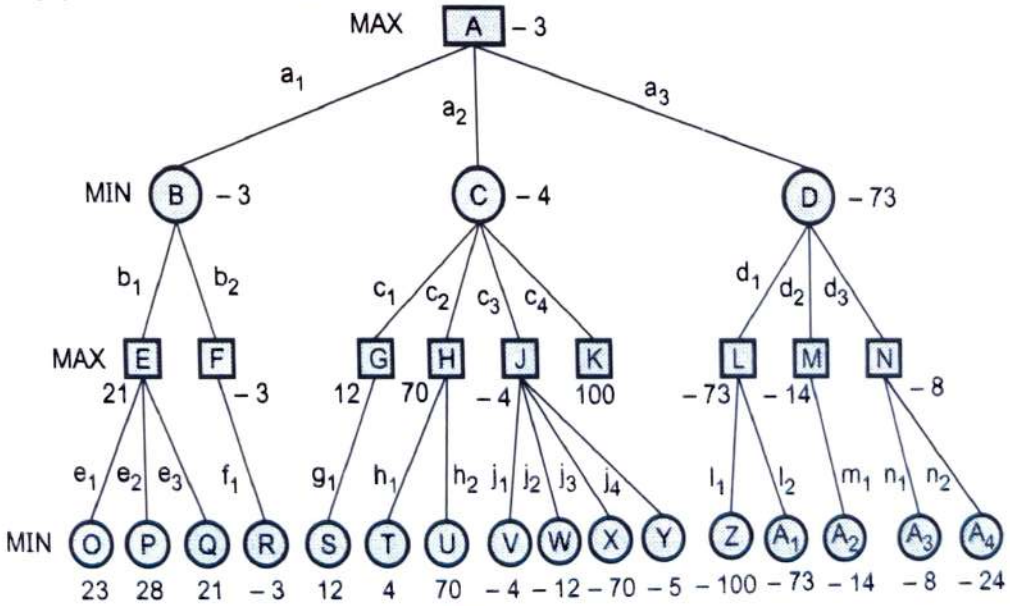


Fig. 9.10.9 Three ply game tree

9.10.7 Mini-Max Algorithm for Playing Multiplayer Games

1. There are many multiplayer games like cricket, football, etc.
2. The multiplayer game can be played with minimax concept.
3. Here the single value for each node is replaced with a vector of values. For example, in a three-player game with players A, B, and C a vector (V_A, V_B, V_C) is associated with each node.
4. For terminal states, this vector gives the utility of the state from each player's viewpoint. (In two player, zero-sum games, the two-element vector can be reduced to a single value because the values are always opposite.
5. In this implementation UTILITY function return a vector of utilities.
6. For non-terminal states we can calculate utility function value as explain below -

Consider following diagram,

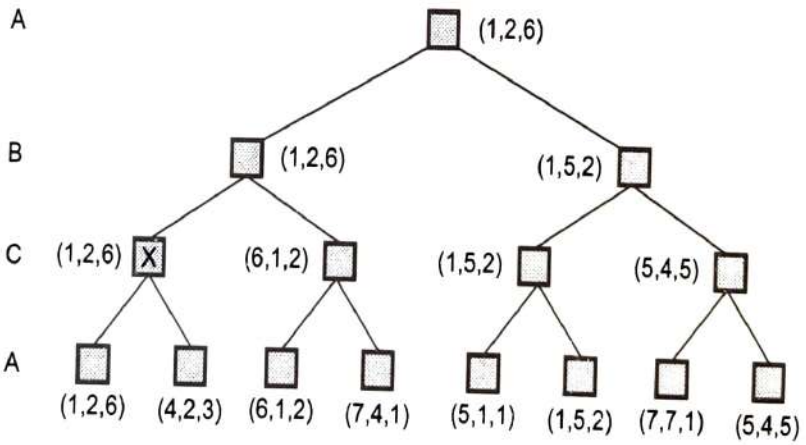


Fig. 9.10.10 Utility function value calculation

7. For non-terminal states vector values should be calculated as explained.

   Consider the node marked X in the game tree shown in above diagram. In this state, player C choose what to do. The two choices lead to terminal states with utility vector ($V_A = 1$, $V_B = 2$, $V_C = 6$) and ($V_A = 4$, $V_B = 2$, $V_C = 3$). Since 6 is bigger than 3, C should choose the first move. This means that if state X is reached subsequent play will lead to a terminal state with utilities ($V_A = 1$, $V_B = 2$, $V_C = 6$). Hence the backed-up value of X is this vector.
8. In general, the backed-up value of a node 'n' is the utility vector of whichever successor has the highest value for the player choosing at 'n'.
9. Multiplayer games usually involve alliance, whether formal or informal, among the players. Alliances are made and broken as the game proceeds.

9.11 Alpha-Beta Pruning

GTU : Summer-12,18, Winter-17,18,19

9.11.1 Motivation for α - β Pruning

1. The problem with minimax algorithm search is that the number of game states it has to examine is exponential in the number of moves.
2. α-β proposes to compute the correct minimax algorithm decision without looking at every node in the game tree.

α-β Pruning Example :

Step 1 :

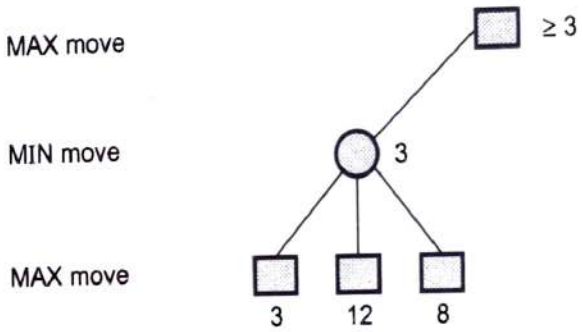


Fig. 9.11.1 Pruning example (i)

Step 2 :

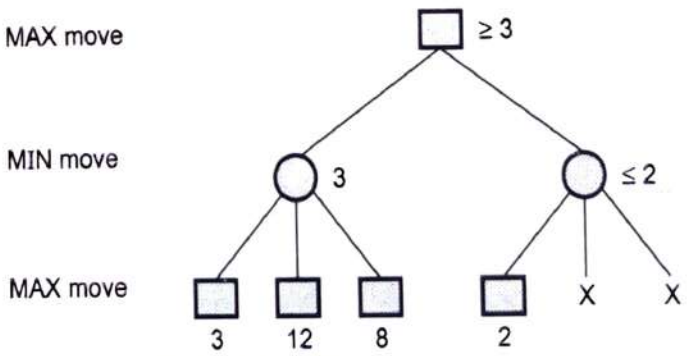


Fig. 9.11.2 Pruning example (ii)

Step 3 :

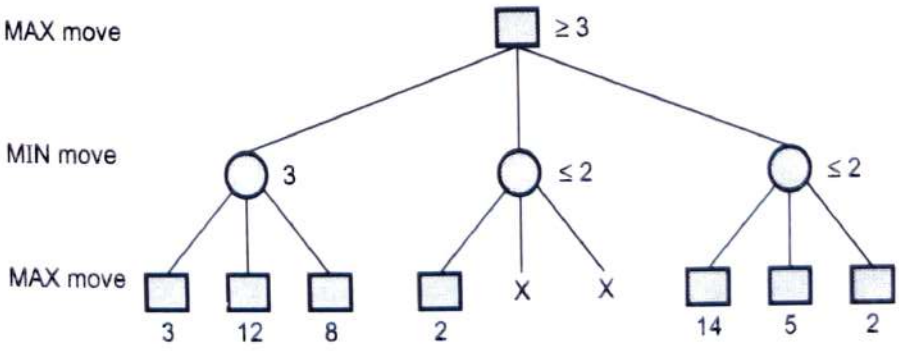


Fig. 9.11.3 Pruning example (iii)

9.11.2 Steps in Alpha-Beta Pruning

1. MAX player cuts off search when he knows MIN-player can force a provably bad outcome.
2. MIN player cuts of search when he knows MAX-player can force provably good (for MAX) outcome.
3. Applying an alpha-cutoff means we stop search of a particular branch because we see that we already have a better opportunity elsewhere.
4. Applying beta-cutoff means we stop search of a particular branch because we see that the opponent already has a better opportunity elsewhere.
5. Applying both forms is alpha-beta pruning.

9.11.3 Alpha Cutoff

It may be found that, in the current branch, the opponent can achieve a state with a lower value for us than one achievable in another branch. So the current branch is one that we will certainly not move the game. Search of this branch can be safely terminated.

For example -

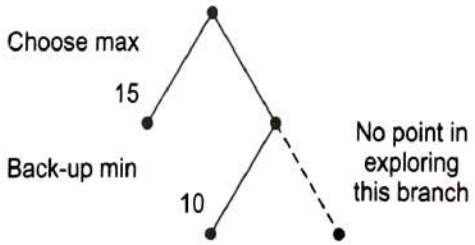


Fig. 9.11.4 Alpha cutoff

9.11.4 Beta-Cutoff

It is just the reverse of alpha-cutoff.

It may also be found, that in the current branch, we would be able to achieve a state which has a higher value for us than one of the opponent can hold us to in another branch. The current branch can be identified as one that the opponent will certainly not move the game. So search in this branch can be safely terminated.

For example - (Refer Fig. 9.11.5)

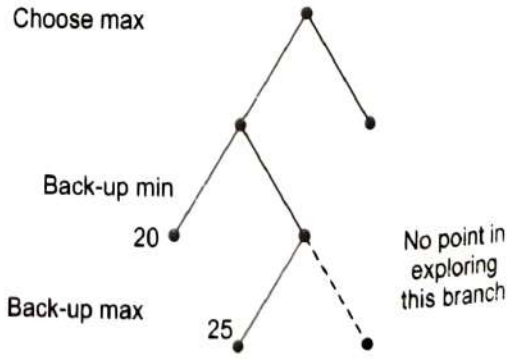


Fig. 9.11.5 Beta cutoff

9.11.5 Algorithm of Alpha-Beta Pruning

/*

alpha is the best score for max along the path to state.

beta is the best score for min along the path to state.

*/

If the level is the top level, let alpha = – infinity, beta = + infinity.

If depth has reached the search limit apply static evaluation function to state and return result.

If player is max :

Until all of state's children are examined with ALPHA-BETA or until alpha is equal to or greater than beta :

Call ALPHA-BETA (child, min, depth + 1, alpha, beta);

note result

Compare the value reported with alpha; if reported value is larger reset alpha to the new value.

Report alpha

If player is min :

Until all of state's children are examined with ALPHA-BETA or until alpha is equal to or greater than beta :

Call ALPHA-BETA, (child, max, depth + 1, alpha, beta);

note result.

Compare the value reported with beta; if reported value is smaller, reset beta to the new value.

Report beta.

9.11.6 Example of Alpha-Beta Pruning (Upto 3rd Ply)

1. In a game tree, each node represents a board position where one of the player gets to choose a move.
2. For example, look at node C in Fig. 9.11.6, as well as look at its left child.
3. We realize that if the players reach node C, the minimizer can limit the utility to 2. But the maximizer can get utility 6 by going to node B instead, so he would never let the game reach C. Therefore we don't even have to look at C's other children.

4. Initially at the root of the tree there is no gurantee about what values the maximizer and minimizer can achieve. So beta is set to ∞ and alpha to – ∞.
5. Then as we move down the tree, each node starts with beta and alpha values passed down from its parent.
6. If it's a maximizer node, then alpha is increased if a child value is greater than the current alpha value. Similarly, at a minimizer node, beta may be dereased. This procedure is shown in Fig. 9.11.6.

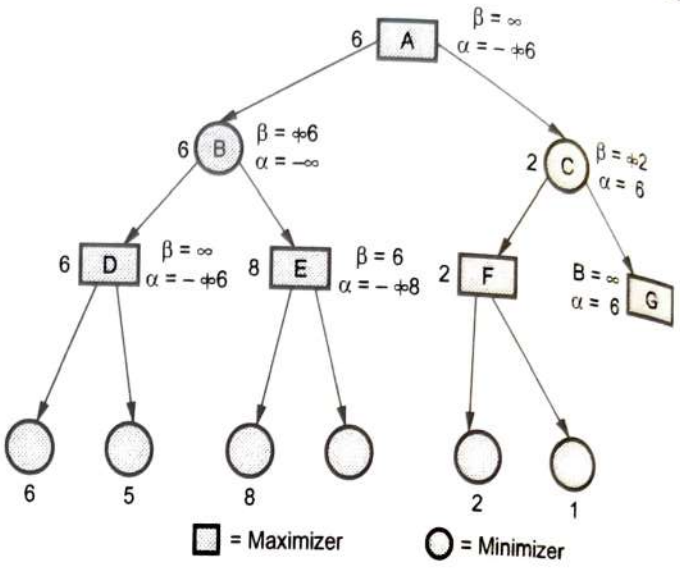


Fig. 9.11.6 Alpha beta pruning

7. At each node, the alpha and beta values may be updated as we iterate over the node's children. At node E, when alpha is updated to a value of 8, it ends up exceeding beta.
8. This is a point where alpha-beta pruning is required we know the minimizer would never let the game reach this node, so we don't have to look at its remaining children.
9. In fact, pruning happens exactly when alpha becomes greater than or equal to beta - that is, when the alpha and beta lines hit each other in the node value.

9.12 Refinements and Heuristic for Cutting Off Search

9.12.1 Evaluation Functions

1. An evaluation function returns an estimate of the expected utility of the game from a given position, just as the heuristic functions return an estimate of the distance to the goal.
2. It should be clear that the performance of a game-playing program is dependant on quality of its evaluation function. An inaccurate evaluation function will guide an agent toward positions that turn out to be lost.

3. Designing evaluation function :

a) The evaluation function should order the terminal states in the same way as the true utility function; otherwise, an agent using it might select suboptimal moves even if it can see ahead all the way to the end of the game.

b) The computation must not take too long !

c) For nonterminal states, the evaluation function should be strongly correlated with the actual chances of winning.

9.12.2 Imperfect and Real-time Decisions

- The minimax algorithm generates the entire game search space, whereas the alpha-beta algorithm allow us to prune large parts of it. However, alpha-beta still has to search all the way to terminal states for at least a portion of the search space. This depth is usually not practical, because moves must be made in a reasonable amount of time-typically a few minutes at most.
- For making search faster we can make a heuristic evaluation function that can be applied to states in the search. It will effectively turn nonterminal nodes into terminal leaves. In other words, the suggestion is to alter minimax or alpha-beta in two ways; the utility function is replaced by a heuristic evaluation function, which gives an estimate of the positions utility, and the terminal test is replaced by a cutoff test that decides when to apply heuristic function.

9.12.3 Cutting Off Search

1. Cutting off search is simple approach for searching faster.
2. The cutting off search is applied to limit the depth.

   If Cutoff-Test (s, depth) then return E(S) problem in cutting off search. (Where E is evaluation function).
3. Cutoff test might be applied in adverse condition.
4. It may stop search before allowable time.
5. Iterative deepening go until time is elapsed.
6. **Quiescent position -**

   a) If a position looks "dynamic", don't even bother to evaluate it.

   b) Instead, do a small secondary search until things calm down. For example - After capturing a piece, things look good, but this would be misleading if opponent was about to capture, right back.

   c) In general such factors are called **continuation heuristics.**

   d) A **quiescent position** is a position which is unlikely to exhibit wild swings (huge changes) in value in near future.

   e) Consider following example -

Here [Refer Fig. 9.12.1] now since the tree is further explored, the values, which is passed to A, is 6. Thus the situation calms down. This is called as **waiting for quiescence.** This helps in avoiding the **horizon effect of a drastic change of values.**

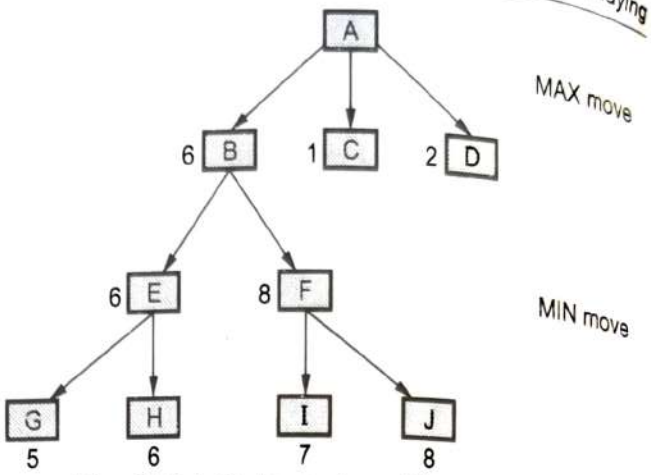


Fig. 9.12.1 Quiescent position example

7. The horizon effect -

A potential problem that arises in game tree search (of a fixed depth) is the horizon effect, which occurs when there is a drastic change in value, immediately beyond the place where the algorithm stops searching. Consider the tree show in Fig. 9.12.2 (a). It has nodes A, B, C and D. At this level since it is a maximizing ply, the value which will passed up at A is 5.

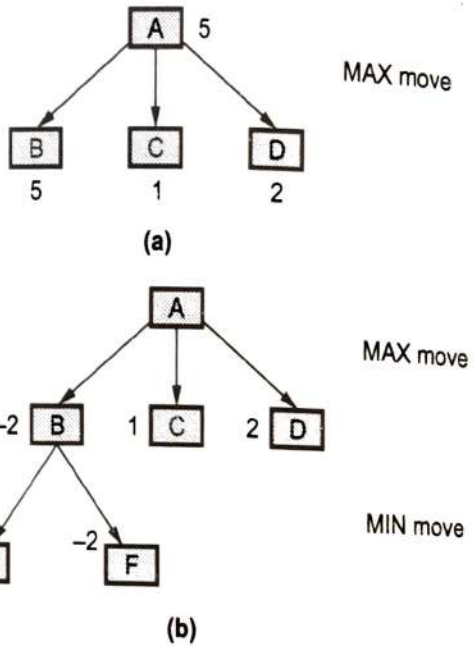


Fig. 9.12.2 Horizon effect example

Suppose node B is examined one more level as shown in Fig. 9.12.2 (b), then we see that because of a minimizing ply, value at B is – 2 and hence the value passed to A is 2. This results in a drastic change in the situation. There are two **proposed solutions** to this problem

a) Singular extension -

Which expands a move that is clearly better than all other moves. Its cost is higher with branching factor 1.

b) Secondary search -

One proposed solution is to examine the search space, beyond the apparently best one, to see that if something is looming just over the horizon. In that case we can revert to second-best move. Obviously then the second-best move has the same problem, and there isn't time to search beyond all possible acceptable moves.

9.12.4 Forward Pruning

1. It is another method for searching faster.
2. In this, some moves at a given node are pruned immediately without further consideration. Clearly, most humans playing chess only consider a few moves from each position (at least consciously).
3. Unfortunately, the approach is rather dangerous because there is no guarantee that the best move will not be pruned away.
4. This can be disasterous if applied near the root, because it may happen that, often the program will miss some "obvious" moves.
5. Forward pruning can be used safely in special siturations. For example, when two moves are symmetric or otherwise equivalent, only one of them need be considered - or for nodes that are deep in the search tree.

9.13 Games with Chance

Games with a certain element of chance are often more interesting than those without chance, and many games involve rolling dice, tossing a coin or something similar.

- In the real world many situation infront of us are unpredictable. It is also observed in many games.

  Example : Dice rolling, backgammon.

- Some time, in the game, imperfect information is available.

  Example : Cards, dominose, etc.

- In game with chance, we can introduce probabilities to our search diagrams and calculate minimax solutions as in the normal games.
- We add 1 more level in game tree i.e. the level of chance nodes.
- Chance nodes have as many successors as outcomes of the random element.
- Minimax with element of chance

  1) di (i = 1, ..., n) – Outcomes from the chance nodes.

  2) P (d_i) - Probability of d_i;

  3) S (N, d_i) - Moves from position N for outcome d_i;

  4) If N is MAX : Utility (N) = $\sum_{i=1}^{n} P(d_i) \max_{s \in S(N, di)}$ utility (s)

  5) If N is MIN :

  Utility (N) = $\sum_{i=1}^{n} P(d_i) \min_{s \in S(N, d_i)}$ utility (s)

- The utility is not computed by using just the terminal values. Therefore values assigned to win, loss and draw affect the choice of moves.
- Time complexity increases (n outcomes from the chance nodes) to $O(b^d n^d)$.
- Alpha-Beta pruning is more complicated in this game trees.

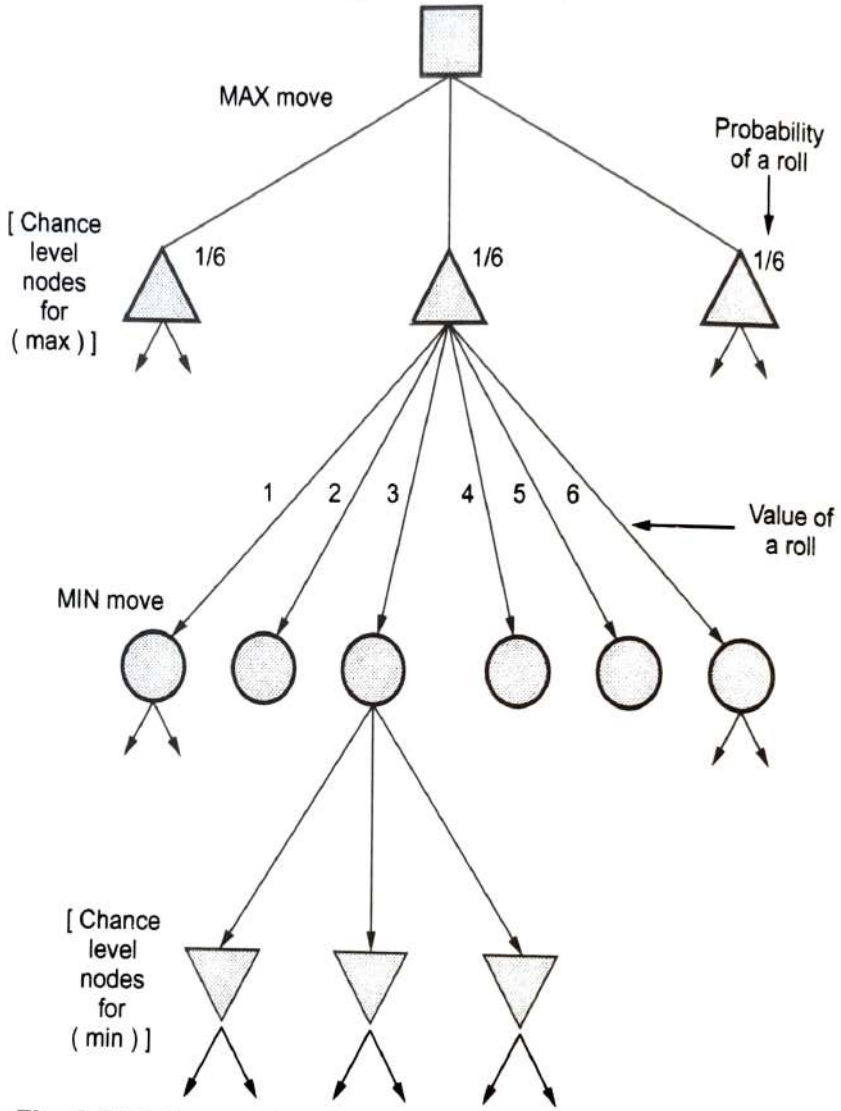


Fig. 9.13.1 A game tree for a backgammon (Game of chance)

Answer in Brief

1. *How can minimix also be extended for game of chance ?* **(Refer section 9.13)**
2. *Explain perfect decisions in game playing. Give example.* **(Refer section 9.9)**
3. *Explain minimax algorithms and how it works for game of tic-tac-toe.* **(Refer section 9.10)**
4. *Explain minimax search procedure with example upto 3rd ply.* **(Refer section 9.10)**
5. *Explain minimax procedure for game playing. Is this DFS or BFS ? How it can be modified to be used by a program playing three-four player game ?* **(Refer section 9.10)**
6. *How minimax procedure can be modified to play multipalyer game ?* **(Refer section 9.10)**

7. *Describe alpha-beta pruning using example. Show game tree upto 3rd ply.* **(Refer section 9.11)**
8. *Describe minimax procedure and alpha-beta pruning.* **(Refer section 9.11)**
9. *Explain alpha-beta pruning algorithm with example.* **(Refer section 9.11)**
10. *Write a short note on quiescent position and secondary search.* **(Refer sections 9.11 and 9.12)**
11. *Explain horizon effect.* **(Refer sections 9.11 and 9.12)**
12. *Explain minimax search algorithm for two player game.* **(Refer section 9.10)**
13. *Also explain alpha-beta pruning.* **(Refer section 9.10)**
14. *Explain the nature and scope of artificial intelligence. Why game playing problems are considered as AI problems ?* **(Refer sections 9.1 and 9.8)**
15. *Is the minimax procedure a depth-first or breadth-first search procedure ?* **(Refer section 9.10)**
16. *Why does the search in game-playing programs always proceed forward from the current position rather than backward from a goal state ?* **(Refer section 9.10)**

9.14 University Questions with Answers

Summer - 12

Q.1 *Explain the Alpha-Beta cutoff procedure in game playing.* **(Refer section 9.11)** [7]

Winter - 12

Q.2 *Explain minmax procedure for game playing with any example.* **(Refer section 9.10)** [7]

Summer - 14

Q.3 *Consider the game tree of Fig. 1 in which the static scores are from first player's point of view. Suppose the first player is maximizing player. Applying mini-max search, show the backed-up values in the tree. What move will the MAX choose ? If the nodes are expanded from left to right, what nodes would not be visited using alpha-beta pruning.* **(Refer section 9.10)** [3]

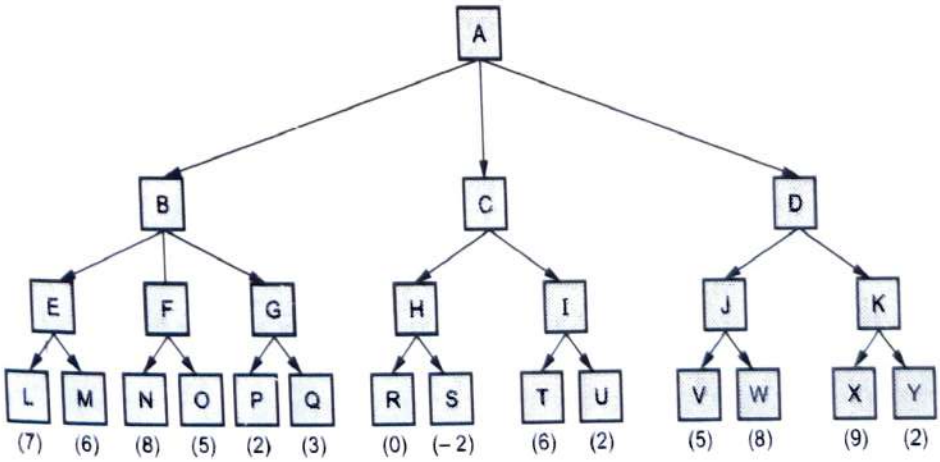


Fig. 1 Game Tree

Winter - 15

Q.4 *Explain Min-Max search procedure with an example.* **(Refer section 9.10)** [7]

Winter - 17

Q.5 *Explain alpha-beta cut off search with an example. State a case when to do alpha pruning.* **(Refer section 9.11)** [7]

Q.6 *Discuss min-max search method.* **(Refer section 9.11)** [3]

Summer - 18

Q.7 *Consider the game tree given in Fig. 2, in which the evaluation function values are shown below each leaf node for the max player. Assume that the root node corresponds to the minimizing player. Assume that the search always visits children left-to-right. Compare the backed-up values computed b the minimax algorithm by writing values at the appropriate nodes in the tree given.* **(Refer section 9.10)** [3]

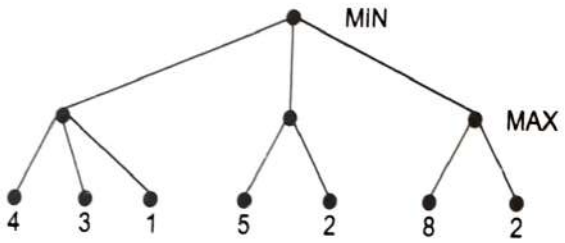


Fig. 2

Q.8 *For the game tree given in Fig. 2, which nodes will not be examined by the alpha-alpha pruning algorithm ? Show the process of alpha-beta pruning to justify your answer.* **(Refer sections 9.10 and 9.11)** [7]

Winter - 18

Q.9 *Discuss alpha-beta cutoffs procedure in game playing.* **(Refer section 9.11)** [4]

Q.10 *Explain min max procedure in game playing.* **(Refer section 9.10)** [7]

Summer - 19

Q.11 *Discuss min-max search method with an example.* **(Refer section 9.10)** [7]

Winter - 19

Q.12 *Explain the MiniMax search procedure for game playing using suitable example. What is the significance of Alpha and Beta cut-offs ?* [7]

(Refer sections 9.10 and 9.11)

Summer - 20

Q.13 *Simulate the working of Tic-tac-toe problem with Minimax technique.* [7]

(Refer section 9.10.5)

❑❑❑

10 Planning in AI Agent World

Syllabus

The Blocks World, Components of a Planning System, Goal Stack Planning, Nonlinear Planning Using Constraint Posting, Hierarchical Planning, Reactive Systems, Other Planning Techniques.

Contents

10.1 Blocks World

GTU : Summer-16

- Inorder to compare the variety of methods of planning, we should find it useful to look at all of them in a single domain that is complex enough that the need for each of the mechanisms is apparent yet simple enough that easy-to follow examples can be found.
- There is a flat surface on which blocks can be placed.
- There are a number of square blocks, all of the same size.
- They can be stacked one upon the other.
- There is robot arm that can manipulate the blocks

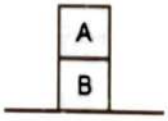
Why Use the Blocks world as an example ?

The blocks world is chosen because :

- It is sufficiently simple and well behaved.
- It is easily understood
- Yet still provides a good sample environment to study planning :
  - Problems can be broken into nearly distinct subproblems.
  - We can show how partial solutions need to be combined to form a realistic complete solution.

Actions of the robot arm

- UNSTACK(A,B)



- STACK(A,B)
- PICKUP(A)
- PUTDOWN(A)
- Notice that the robot arm can hold only one block at a time.

Predicates

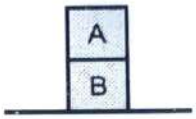
Inorder to specify both the conditions under which an operation may be performed and the results of performing it, we need the following predicates :

- ON(A,B)
- ONTABLES(A)
- CLEAR(A)

- HOLDING(A)
- ARMEMPTY

Simple Position

- State S0



- ON(A, B, S0) ^ ONTABLE(B, S0)^CLEAR(A, S0)

If we execute UNSTACK(A,B) in state S0, then the resulting state S1 : HOLDING(A,S1)^CLEAR(B,S1).

To enable the complete situation to be described, we provide a set of rules called **frame axioms**, that describe components of the state that are not affected by each operator.

- ONTABLE(x,s) -> ONTABLE(z, DO(UNSTACK(x,y),s))
- DO is a function that specifies for a given state and a given action, the new state that results from the execution of the action.

Robot - problem solving system (STRIPS)

- List of new predicates that the operator causes : ADD, DELETE
- PRECONDITIONS list contains those predicates that must be true for the operator to be applied.

STRIPS style operators for BLOCKs World

- STACK(x,y)
  - P : CLEAR(y)^HOLDING(x)
  - D : CLEAR(y)^HOLDING(x)
  - A : ARMEMPTY^ON(x,y)
- PICKUP(x)
  - P : CLEAR(x) ^ ONTABLE(x) ^ARMEMPTY
  - D : ONTABLE(x) ^ ARMEMPTY
  - A : HOLDING(x)

A simple search tree

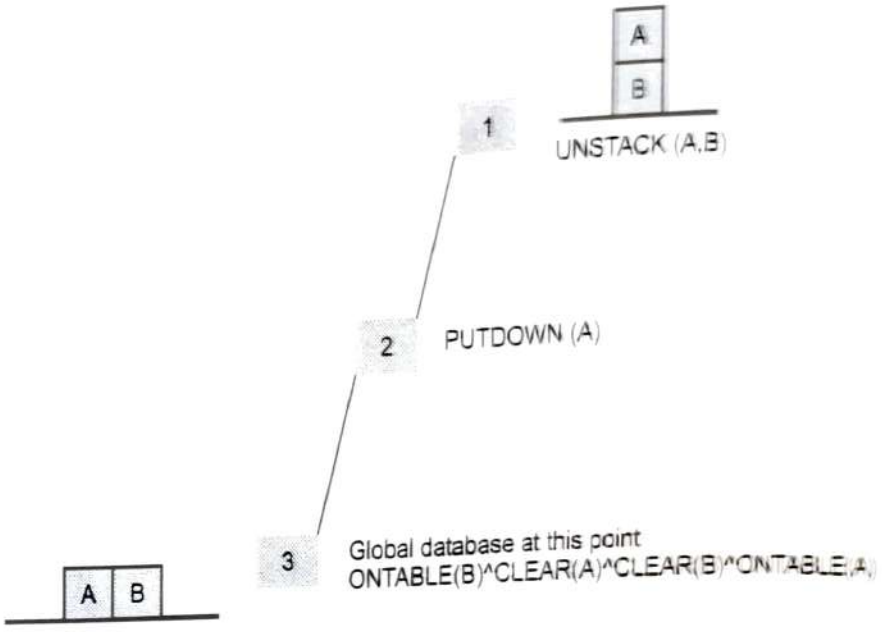


Fig. 10.1.1

10.2 Planning

GTU : Summer-14

The methods which focus on ways of decomposing the original problem into appropriate subparts and on ways of recording and handling interactions among the subparts as they are detected during the problem-solving process are often called as planning. Planning refers to the process of computing several steps of a problem-solving procedure before executing any of them.

What does planning involve ?

Planning problems are hard problems :

- They are certainly non-trivial.
- Solutions involve many aspects that we have studied so far :
  - Search and problem solving strategies.
  - Knowledge representation schemes.
  - Problem decomposition - breaking problem into smaller pieces and trying to solve these first.

We have seen that it is possible to solve a problem by considering the appropriate form of knowledge representation and using algorithms to solve parts of the problem and also to use searching methods.

Search in planning

Search basically involved moving from an *initial state* to a *goal state*. Classic search techniques could be applied to planning state in this manner :

A* Algorithm

- Best first search,

Problem decomposition

- *Synthesis*, Frame Problem.

AO* Algorithm

- Split problem into *distinct* parts.

Heuristic reasoning

Ordinary search backtracking can be hard so introduce reasoning to devise heuristics and to control the backtracking.

The first major method considered the solution as a search from the initial state to a goal state through a state space. There are many ways of moving through this space by using operators and the *A* algorithm* described the best first search through a graph. This method is fine for simpler problems but for more realistic problems it is advisable to use problem decomposition. Here the problem is split into smaller sub problems and the partial solutions are then *synthesised*. The **danger** in this method occurs when certain paths become *abortive* and have to be discarded. How much of the partial solution can be saved and how much needs to be recomputed.

The *frame problem* - deciding which things change and which do not - gave some guidance on enabling us to decide on what stays the same and on what changes as we go from state to state. If the problem concerned designing a robot to build a car then mounting the engine on the chassis would not affect the rear of the car now-a-days.

The *AO* algorithm* enabled us to handle the solution of problems where the problem could be split into distinct parts and then the partial solutions reassembled. However difficulties arise if parts interacted with one another. Most problems have some interaction and this implies some thought into the ordering of the steps; for example if the robot has to move a desk with objects on it from one room to another; or to move a sofa from one room to another and the piano is near the doorway. The thought process involved in recombining partial solutions of such problems is known as planning. At this point we run into a discussion about the role of the computer in the design of a plan as to how we can solve a problem. It is extremely unlikely at this stage that a computer will actually solve the problem unless it is a game and here some interaction with a person is needed.

Generally the computer is used to decide upon or to offer words of wisdom on the best method of approaching the solution of a problem. In one sense this can be interpreted as a simulation; if we are considering the handling of queues at an airport or a post office there are no actual people and so we can try a range of possibilities; likewise in a traffic control problem there are no cars or aircraft piling up at a junction or two miles over an airport. Once the computer based investigation has come up with the best solution we can then implement it at the real site.

This approach assumes that there is continuity in the way of life. It cannot budget for rapid change or revolution. How can this approach cater for unexpected events such as a faulty component or a spurious happening such as two items stuck together or a breakdown in the path somewhere vital such as in a camera reel. When a fault or some unrecognizable state is encountered it is not necessary to restart for much of what has been successfully solved is still useful. Consider a child removing the needles from a partially completed knitted sweater. The problem lies in restarting from a dead end and this will need some backtracking. This method of solution is pursued to reduce the level of complexity and so to ensure successful handling we must introduce reasoning to help in the backtracking required to cater for faults. To assist in controlling backtracking many methods go backward from the goal to an initial state.

- **Components of a planning system**

1. Choose the best rule to apply next based on the best available heuristic information.
2. Apply the chosen rule to compute the new problem state that arises from its application.
3. Detect when a solution has been found.
4. Detect when an almost correct solution has been found and employ special techniques to make it totally correct.
5. Detect dead ends so that they can be abandoned and the system's effort directed in more fruitful directions.

1. Choose the rules to apply

- The most widely used technique for selecting an appropriate rules to apply is first to isolate a set of differences between desired goal state and then to identify those rules that are relevant to reducing those differences.
- If several rules, a variety of other heuristic information can be exploited to choose among them.

For example, if we wish to travel by car to visit a friend, then

- The first thing to do is to fill up the car with fuel.

- If we do not have a car then we need to acquire one.
- The largest difference must be tackled first.

2. Applying rules

- In simple systems, applying rules is easy. Each rule simply specified the problem state that would result from its application.
- In complex systems, we must be able to deal with rules that specify only a small part of the complete problem state.
- One way is to describe, for each action, each of the changes it makes to the state description.
- A number of approaches to this task have been used.

Green's Approach (1969)

Basically this states that *we note the changes to a state produced by the application of a rule.* Consider the problem of having two blocks *A* and *B* stacked on each other (*A* on top). Then we may have an initial state S_0 which could be described as :

$$ON(A, B, S_0) \wedge$$
$$ONTABLE(B, S_0) \wedge$$
$$CLEAR(A, S_0)$$

If we now wish to UNSTACK(*A*, *B*). We express the operation as follows :

$$[CLEAR(x,s) \wedge ON(x, y, s)] \rightarrow$$
$$[HOLDING(x, DO(UNSTACK(x,y),s)) \wedge$$
$$CLEAR(y,DO(UNSTACK(x,y),s))]$$

where *x*, *y* are any blocks, *s* is any state and DO() specifies that a new state results from the given action.

The result of applying this to state S_0 to give state S_1 then we get :

$HOLDING(A, S_1)$ $CLEAR(B, S_1)$.

There are a few problems with this approach :

The frame problem

- In the above we know that *B* is still on the table. This needs to be encoded into *frame axioms* that describe components of the state *not* affected by the operator.

The qualification problem

- If we resolve the frame problem the resulting description may still be inadequate. Do we need to encode that a block cannot be placed on top of itself ? If so should this attempt *fail* ?

If we allow failure things can get complex - do we allow for a lot of unlikely events ?

The ramification problem

- After unstacking block *A*, previously, how do we know that *A* is no longer at its initial location ? *Not only is it hard to specify exactly what does not happen (frame problem) it is hard to specify exactly what does happen.*

State Description Approach

In this approach for each action, a change it makes to the state is described, everything else remains unchanged.

The main **advantage** of this method is single mechanism (that is resolution) can perform all operations that are required for state description. The major **disadvantage** is the number of axioms that are required becomes very large if problem-state description is complex.

STRIPS Approach (1971)

STIPS proposed another approach :

- Basically each operator has three lists of predicates associated with it :
  - a list of things that become TRUE called ADD.
  - a list of things that become FALSE called DELETE.
  - a set of prerequisites that must be true before the operator can be applied.
- Anything not in these lists is assumed to be unaffected by the operation.
- This method initial implementation of STRIPS - has been *extended* to include other forms of reasoning/planning (*e.g.* Nonmonotonic methods, Goal Stack Planning and even Nonlinear planning - We will see later.)

Consider the following example in the Blocks World and the fundamental operations :

STACK

- Requires the arm to be holding a block A and the other block B to be clear. Afterwards the block A is on block B and the arm is empty and these are true - ADD; The arm is not holding a block and block B is not clear; predicates that are false DELETE;

UNSTACK

- Requires that the block A is on block B; that the arm is empty and that block A is clear. Afterwards block B is clear and the arm is holding block A - ADD; The arm is not empty and the block A is not on block B - DELETE;

We have now greatly reduced the information that needs to be held. If a new attribute is introduced we do not need to add new axioms for existing operators. Unlike in Green's method we remove the state indicator and use a database of predicates to indicate the current state

Thus if the last state was :

ONTABLE(B) ^ ON(A,B) ^ CLEAR(A)

after the unstack operation the new state is

ONTABLE(B) ^ CLEAR(B) ^ HOLDING(A) ^ CLEAR(A)

3. Detecting Progress

The final solution can be detected if,

- We can devise a predicate that is true when the solution is found and is false otherwise.
- Requires a great deal of thought and requires a proof.

4. Detecting a solution

- A planning system has succeeded in finding a solution to a problem when it has found a sequence of operators that transforms the initial problem state into the goal state.
- How will it know when this has been done ?
- In simple problem-solving systems, this question is easily answered by a straightforward match of the state descriptions.
- One of the representative systems for planning systems is, predicate logic. Suppose that as a part of our goal, we have the predicate P(x). To see whether P(x) is satisfied in some state, we ask whether we can prove P(x) given the assertions that describe that state and the axioms that define the world model.

5. Detecting Dead Ends

- As a planning system is searching for a sequence of operators to solve a particular problem, it must be able to detect when it is exploring a path that can never lead to a solution.
- The same reasoning mechanisms that can be used to detect a solution can often be used for detecting a dead end.
- If the search process is **reasoning forward** from the initial state, it can prune any path that leads to a state from which the goal state cannot be reached.
- If search process is **reasoning backward** from the goal state, it can also terminate a path either because it is sure that the initial state cannot be reached or because little progress is being made. In backward reasoning each goal is decomposed into subgoals. Each of these subgoals may lead to a set of additional subgoals. The major disadvantage is that every subgoal can lead to multiple subgoals making a problem harder than original one.

Detecting false trails is also necessary :

- E.g. A* search - if insufficient progress is made then this trail is aborted in favour of a more hopeful one.
- Sometimes it is clear that solving a problem one way has reduced the problem to parts that are harder than the original state.
- By moving back from the goal state to the initial state it is possible to detect conflicts and any trail or path that involves a conflict can be pruned out.
- Reducing the number of possible paths means that there are more resources available for those left.

Supposing that the computer teacher is ill at a school there are two possible alternatives :

- Ttransfer a teacher from mathematics who knows computing or
- Bring another one in.

Possible Problems :

- If the mathematics teacher is the only teacher of mathematics the problem is not solved.
- If there is no money left the second solution could be impossible.

If the problems are nearly decomposable we can treat them as decomposable and then patch them, how ? Consider the final state reached by treating the problem as decomposable at the current state and noting the differences between the goal state and the current state, and the goal state and the initial state and use appropriate Means End analysis techniques to move in the best direction. Better is to work back in the path leading to the current state and see if there are options. It may be that one optional path could lead to a solution whereas the existing route led to a conflict. Generally this means that some conditions are changed prior to taking an optional path through the problem.

Another approach involves putting off decisions until one has to, leaving decision making until more information is available and other routes have been explored. Often some decisions need not be taken as these nodes are never reached.

Repairing an Almost Correct Solution

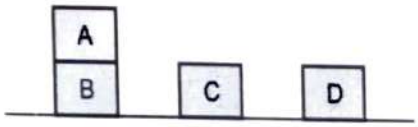
- The kinds of techniques we are discussing are often useful in solving nearly decomposable problems.
- One good way of solving such problems is to assume that they are completely decomposable, proceed to solve the sub-problems separately, and then check that when the sub-solutions are combined, they do in fact yield a solution to the original problem.

10.3 Goal Stack Planning

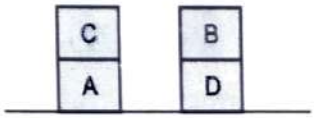
- In this method, the problem solver makes use of a single stack that contains both goals and operators that have been proposed to satisfy those goals.
- The problem solver also relies on a database that describes the current situation and a set of operators described as PRECONDITION, ADD, and DELETE lists.
- The goal stack planning method attacks problems involving conjoined goals by solving the goals one at a time, in order.
- A plan generated by this method contains a sequence of operators for attaining the first goal, followed by a complete sequence for the second goal etc.
- At each succeeding step of the problem solving process, the top goal on the stack will be pursued.
- When a sequence of operators that satisfies it is found, that sequence is applied to the state description, yielding new description.
- Next, the goal that is then at the top of the stack is explored and an attempt is made to satisfy it, starting from the situation that was produced as a result of satisfying the first goal.
- This process continues until the goal stack is empty.
- Then as one last check, the original goal is compared to the final state derived from the application of the chosen operators.
- If any components of the goal are not satisfied in that state, then those unsolved parts of the goal are reinserted onto the stack and the process is resumed.

Goal stack planning

- To start with goal stack is simply :
- On (C, A)^ON (B, D)^ONTABLE(A)^ONTABLE(D)



Start :
ON(B, A)^ONTABLE(A)^
ONTABLE(C)
^ONTABLE(D)
^ARMEMPTY



Goal : ON(C, A)^ON(B, D)^
ONTABLE(A)^ONTABLE(D)

- This problem is separate into four subproblems, one for each component of the goal.
- Two of the subproblems ONTABLE(A) and ONTABLE(D) are already true in the initial state.
- Alternative 1: Goal stack :
- ON (C, A)
- ON (B, D)
- ON (C, A)^ON(B, D)^OTAD
- Alternative 2 : Goal stack :
- ON (B, D)
- ON (C,A)
- ON (C,A)^ON(B,D)^OTAD

***Exploring Operators**

- Pursuing alternative 1, we check for operators that could cause ON(C,A)
- Of the 4 operators, there is only one STACK. So it yields :
- **STACK(C,A)**
- **ON(B,D)**
- **ON(C,A)^ON(B,D)^OTAD**
- Preconditions for STACK(C,A) should be satisfied, we must establish them as subgoals :
- **CLEAR(A)**
- **HOLDING(C)**
- **CLEAR(A)^HOLDING(C)**
- **STACK(C,A)**
- **ON(B,D)**
- **ON(C,A)^ON(B,D)^OTAD**
- Here we exploit the Heuristic that if HOLDING is one of the several goals to be achieved at once, it should be tackled last.
- Next we see if CLEAR(A) is true, it is not. The only operator that could make it true is UNSTACK(B, A). This produces the goal stack :
- **ON(B,A)**
- **CLEAR(B)**
- **ON (B,A)^CLEAR(B)^ARMEMPTY**

- **UNSTACK(B,A)**
- **HOLDING(C)**
- **CLEAR(A)^HOLDING(C)**
- **STACK(C,A)**
- **ON(B,D)**
- **ON(C,A)^ON(B,D)^OTAD**
- We see that we can pop predicates on the stack till we reach HOLDING(C) for which we need to find suitable operator.
- The operators that might make HOLDING(C) true : PICKUP(C) and UNSTACK(C,x). Without looking ahead, since we cannot tell which of these operators is appropriate, we create two branches of the search tree corresponding to the following goal stacks :

ALT1 :	ALT2 :
ONTABLE(C)	ON(C,X)
CLEAR(C)	CLEAR(C)
ARMEMPTY	ARMEMPTY
ONTABLE(C) ^CLEAR(C)^ARMEMPTY	ON(C,X)^CLEAR(C)^ARMEMPTY
PICKUP(C)	UNSTACK(C,X)
CLEAR(A)^HOLDING (C)	CLEAR(A)^HOLDING (C)
STACK(C, A)	STACK(C,A)
ON(B, D)	ON(B,D)
ON(C,A)^ON(B,D)^OT AD	ON(C,A)^ON(B,D)^OT AD

Table 10.3.1 Goal stack

***Choosing Alternative**

- How should our program choose now between alternative 1 and alternative 2 ?
- We can tell that picking up C(alt 1) is better than unstacking it because it is not currently on anything. So to unstack it, we would first have to stack it. This would be waste of effort.
- But how could the program know that ?
- If we pursue the alternative 1, the top element on the goal stack is ONTABLE(C) which is already satisfied, so we pop it off. CLEAR(C) is also satisfied and is popped off.

- The remaining precondition of PICKUP(C) is ARMEMPTY which is not satisfied since HOLDING(B) is true.So we apply the operator STACK(B,D). This makes the goal stack :
- **CLEAR(D)**
- **HOLDING(B)**
- **CLEAR(D)^HOLDING(B)**
- **STACK(B,D)**
- **ONTABLE(C)^CLEAR(C)^ARMEMPTY**
- **PICKUP(C)**
- **CLEAR(A)^HOLDING(C)**
- **STACK(C,A)**
- **ON(B,D)**
- **ON(C,A)^ON(B,D)^OTAD**

***Complete plan**

1. UNSTACK(C,A)
2. PUTDOWN(C)
3. PICKUP(A)
4. STACK(A,B)
5. UNSTACK(A,B)
6. PUTDOWN(A)
7. PICKUP(B)
8. STACK(B,C)
9. PICKUP(A)
10. STAKC(A,B)

10.4 Non-linear Planning using Constraint Posting GTU : Summer-18

- Difficult problems cause goal interactions,
- The operators used to solve one subproblem may interfere with the solution to a previous subproblem.
- Most problems require an interwined plan in which multiple subproblems are worked on simultaneously.
- Such a plan is called nonlinear plan because it is not composed of a linear sequence of complete subplans.

- Let us reconsider the **SUSSMAN ANOMALY.**
- Problems such as this one require subproblems to be worked on simultaneously.
- Thus a nonlinear plan using heuristics such as :

1. Try to achieve ON(A,B) clearing block A putting block C on the table.
2. Achieve ON(B,C) by stacking block B on block C.
3. Complete ON(A,B) by stacking block A on block B.

Constraint posting has emerged as a central technique in recent planning systems (e.g. MOLGEN and TWEAK)

Constraint posting builds up a plan by :

- Suggesting operators,
- Trying to order them and
- Produce bindings between variables in the operators and actual blocks.

The initial plan consists of *no* steps and by studying the goal state ideas for the possible steps are generated. There is *no order or detail* at this stage. Gradually more detail is introduced and constraints about the order of subsets of the steps are introduced until a *completely ordered* sequence is created. In this problem means-end analysis suggests two steps with end conditions ON(A,B) and ON(B,C) which indicates the operator STACK giving the layout shown below where the operator is preceded by its preconditions and followed by its post conditions :

CLEAR(B)	CLEAR(C)
*HOLDING(A)	*HOLDING(B)
STACK(A,B)	STACK(B,C)
ARMEMPTY	ARMEMPTY
ON(A,B)	ON(B,C)
CLEAR(B)	CLEAR(C)
¬ HOLDING(A)	¬ HOLDING(B)

NOTE

- There is no order at this stage.
- Unachieved preconditions are starred (*).
- Both of the HOLDING preconditions are unachieved since the arm holds nothing in the initial state.
- Delete postconditions are marked by (¬).

Many planning methods have introduced heuristics to achieve goals or preconditions. The TWEAK planning method brought all these together under one formalism. Other methods that introduced/used the following heuristics are mentioned in brackets in the following section.

***Constraint Posting**

- The idea of constraint posting is to build up a plan by incrementally hypothesizing operators, partial orderings between operators, and binding of variables within operators.
- At any given time in the problem-solving process, we may have a set of useful operators but perhaps no clear idea of how those operators should be ordered with respect to each other.
- A solution is a partially ordered, partially instantiated set of operators to generate an actual plan, we convert the partial order into any of a number of total orders.

***Constraint posting versus state space search**

State Space Search

- Moves in the space :

  Modify world state via operator

- Model of time :

  Depth of node in search space

- Plan stored in :

  Series of state transitions

Constraint Posting Search

- Moves in the space :
  - Add operators
  - Order operators
  - Bind variables
  - Or Otherwise constrain plan
- Model of Time :
  - Partially ordered set of operators
- Plan stored in :
  - Single node

The *Tweak* planning method involves the following heuristics :

Step Addition

- Creating new steps (GPS).

Promotion

- Constraining a step to go before another step (Sussman's HACKER).

Declobbering

- Placing a new step between two steps to revert a precondition (NOAH, NONLIN).

Simple Establishment

- Assigning a value to a variable to ensure a precondition (TWEAK).

Separation

- Preventing variables being assigned certain values (TWEAK).

Now lets look at the effect of each heuristic.

We must try and achieve the preconditions of the STACK operation above. We could try picking up the respective blocks :

CLEAR(A)	CLEAR(C)
ONTABLE(A)	ONTABLE(B)
*ARMEMPTY	*ARMEMPTY
PICKUP(A)	PICKUP(B)
ONTABLE(A)	¬ ONTABLE(B)
¬ ARMEMPTY	¬ ARMEMPTY
HOLDING(A)	HOLDING(B)

At the moment there is no plan as the postconditions of this set could negate the preconditions of the first (STACK) plan so we must introduce order :

- If the eventual plan contains a PICKUP then STACK step then.
- HOLDING preconditions would need to be satisfied by some other steps.
- *Solve* this by enforcing ordering by introducing constraints whenever step addition is employed.
- In this case we need to state that a PICKUP step should precede a corresponding STACK step. That is to say

  PICKUP(A) ← STACK(A,B)

  PICKUP(B) ← STACK(B,C)

This gives four steps partially ordered and four unachieved conditions.

- *CLEAR(A) - Block A is not clear in initial state.
- *CLEAR(B) - Although block B is clear in initial state STACK(A,B) with postcondition CLEAR(B) might precede the step with *CLEAR(B) precondition.
- Two *ARMEMPTY - initial state makes ARMEMPTY but PICKUP step has ARMEMPTY and could again precede this step.

We can use the *promotion* heuristic to force one operator to precede another so that the postcondition of one operator STACK(A,B) does not negate the precondition CLEAR(B) of another operator PICKUP(B). This ordering is represented by

PICKUP(B) ← STACK(A,B)

We can use promotion to achieve one of the ARMEMPTY preconditions :

Making PICKUP(B) precede PICKUP(A) ensures that the arm is empty and all the conditions for PICKUP(B) are met.

This is written

PICKUP(B) ← PICKUP(A).

Unfortunately a postcondition of the first operator is that the arm becomes not empty, so we need to use the *declobbering* heuristic to achieve the preconditions of the second operator PICKUP(A).

Declobbering

- PICKUP(B) asserts ¬ ARMEMPTY.
- But if we insert a step between PICKUP(B) and PICKUP(A) to reassert ARMEMPTY then we can achieve the precondition.
- STACK(B,C) can do this so we *post another constraint :*
  PICKUP(B) ← STACK(B,C) ← PICKUP(A)

We still need to achieve CLEAR(A) :

The appropriate operator is UNSTACK(x,A) by *step addition*. This leads to the following set of conditions

*CLEAR(x)
*ON(x,A)
*ARMEMPTY

UNSTACK(x,A)

ON(x,A)
¬ ARMEMPTY
HOLDING(x)
CLEAR(A)

The variable x can be bound to the block C by the *simple establishment* heuristic since C is on A in the initial state. The preconditions CLEAR(C) and ARMEMPTY are negated by STACK(B,C) and by PICKUP(B) or PICKUP(A) however.

So we must introduce three orderings by *promotion* to ensure the operator UNSTACK(C,A).

UNSTACK(C,A) ***leftarrow*** **STACK(B,C)**

UNSTACK(C,A) ***leftarrow*** **PICKUP(A)**

UNSTACK(C,A) ***leftarrow*** **PICKUP(B)**

Promotion involves adding a step and this clobbers one of the preconditions of PICKUP(B) viz ARMEMPTY, always a potential problem with this heuristic.

However all is not lost as there is an operator, PUTDOWN that has the required postcondition and given that the operator UNSTACK(C,A) had generated the precondition for it of HOLDING(C) we can produce an extra operator successfully

HOLDING(C)

PUTDOWN(C)

¬ HOLDING(C)

ONTABLE(C)

ARMEMPTY

This operator *declobbers* the operator PICKUP(B) yielding the sequence

UNSTACK(C,A) ← PUTDOWN(C) ← PICKUP(B)

This yields the final sequence :

1. UNSTACK(C,A)
2. PUTDOWN(C)
3. PICKUP(B)
4. STACK (B,C)
5. PICKUP(A)
6. STACK(A,B)

Let us finish this section by looking at the formal form of the TWEAK algorithm :

1. Initialize S to be the set of propositions in the goal state.
2. Repeat
   a. Remove some unachieved proposition P from S.
   b. Achieve P by using one of the heuristics.
   c. Review all the steps, including additional steps to find all unachieved preconditions, add these to S the set of unachieved preconditions until the set S is empty.
3. Complete the plan by converting partial orders into a total order performing all necessary instantiations, binding of the variables.

*Modal Truth Criterion

- A proposition P is necessarily true in a state S if and only if two conditions hold: there is a state T equal or necessarily previous to S in which P is necessarily asserted; and for every step C possibly before S and every proposition Q possibly co-designating with P which C denies, there is a step W necessarily between C and S which asserts R, a proposition such that R and P co-designate whenever P and Q co-designate.
- Modal truth criterion tells us when a proposition is true.

10.5 Hierarchical Planning

- In order to solve hard problems, a problem solver may have to generate long plans.
- It is important to be able to eliminate some of the details of the problem until a solution that addresses the main issues is found.
- Then an attempt can be made to fill the appropriate details.
- Early attempts to do this involved the use of macro operators.
- But in this approach, no details were eliminated from actual descriptions of the operators.
- As an example, suppose you want to visit a friend in Europe but you have a limited amount of cash to spend. First preference will be find the airfares, since finding an affordable flight will be the most difficult part of the task. You should not worry about getting out of your driveway, planning a route to the airport etc, until you are sure you have a flight.
- The assignment of appropriate criticality value is critical to the success of this hierarchical planning method.
- Those preconditions that no operator can satisfy are clearly the most critical.

- Example, solving a problem of moving robot, for applying an operator, PUSH-THROUGH DOOR, the precondition that there exist a door big enough for the robot to get through is of high criticality since there is nothing we can do about it if it is not true.

ABSTRIPS

- ABSTRIPS actually planned in a hierarchy of abstraction spaces, in each of which preconditions at a lower level of abstraction were ignored.
- ABSTRIPS approach is as follows :
  - First solve the problem completely, considering only preconditions whose criticality value is the highest possible.
  - These values reflect the expected difficulty of satisfying the precondition.
  - To do this, do exactly what STRIPS did, but simply ignore the preconditions of lower than peak criticality.
  - Once this is done, use the constructed plan as the outline of a complete plan and consider preconditions at the next-lowest criticality level.
  - Because this approach explores entire plans at one level of detail before it looks at the lower-level details of any one of them, it has been called **length-first approach.**

10.6 Reactive Systems

- The idea of reactive systems is to avoid planning altogether, and instead use the observable situation as a clue to which one can simply react.
- A reactive system must have access to a knowledge base of some sort that describes what actions should be taken under what circumstances.
- A reactive system is very different from the other kinds of planning systems we have discussed because it chooses actions one at a time.
- It does not anticipate and select an entire action sequence before it does the first thing.
- Example is a thermostat. The job of the thermostat is to keep the temperature constant.
- Reactive systems are capable of surprisingly complex behaviours.
- The main advantage reactive systems have over traditional planners is that they operate robustly in domains that are difficult to model completely and accurately.
- Reactive systems dispense with modeling altogether and base their actions directly on their perception of the world.

- Another advantage of reactive systems is that they are extremely responsive, since they avoid the combinatorial explosion involved in deliberative planning.
- This makes them attractive for real time tasks such as driving and walking.

Thermostat

- Is an example for reactive systems.
- Its job is to keep the temperature constant inside a room.
- One might imagine a solution to this problem that requires significant amounts of planning, taking into account how the external temperature rises and falls during the day, how heat flows from room to room, and so forth.
- Real thermostat uses simple pair of situation-action rules :
  1. If the temperature in the room is k degrees above the desired temperature, then turn the airconditioner on.
  2. If the temperature in the room is k degrees below desired temperature, then turn the airconditioner off.

10.7 Other Planning Techniques

- Triangle tables [Fikes et al.,1972] - It provides a way recording the goals that each operator is expected to satisfy as well as the goals that must be true for its correct execution. If some exceptional situation occurs during execution of the plan, then a table is provided that stores the information which is required to patch the plan.
- Metaplanning [Stefik,1981 a] - It is a technique for reasoning about the problem being solved, which also provides a way organizing the planning process.
- Macro-operators [Fikes and Nilsson, 1971] - It allows a planner to build new operators which represent commonly used sequences of operators.
- Case-based planning [Hammond, 1986] - It makes use of already existing plan to develop new plans there by providing reusability of existing plans.

Answer in Brief

1. *Explain various planning techniques ?* **(Refer sections 10.2, 10.3, 10.4 and 10.5)**
2. *What is regression ?* **(Refer section 10.2)**
3. *What are differences and similarities between problem solving and planning ?* **(Refer section 10.2)**
4. *Explain the concept of planning with state-space search with suitable example.* **(Refer section 10.2)**
5. *Write algorithms for TWEAK and STRIPS.* **(Refer section 10.2)**
6. *Explain goal stack planning method.* **(Refer section 10.3)**

7. *Explain various ways of applying rules in complex system.* **(Refer section 10.4)**
8. *Explain non-linear planning method.* **(Refer section 10.4)**
9. *Explain heirarchical planning with example.* **(Refer section 10.5)**
10. *Write short note on learning methods.* **(Refer section 10.2)**
11. *Design the operators and specify their respective precondition (P), delete (D) and add lits.* **(Refer section 10.2)**
12. *Consider the following representation on from blocks world.*

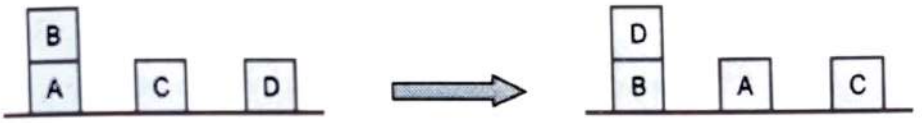
    Start : $ON(B, D) \wedge ON(C, B) \wedge ONTABLE(D) \wedge ONTABLE(A)$.

    Goal : $ON(A, B) \wedge ON(C, D) \wedge ONTABLE(B) \wedge ONTABLE(D)$.

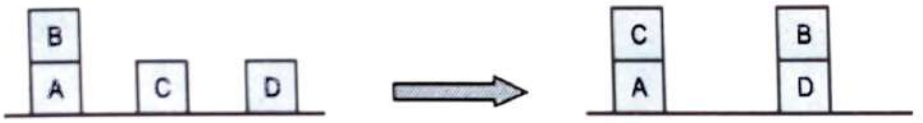
    i) Show how STRIPS would solve this problem.

    ii) Did these processes produce optimum plans and if not justify how it can be done ?

    (Refer section 10.2)
13. *Explain goal stack planning. Solve the following using goal stack planning.* **(Refer section 10.2)**



14. *Solve the following using goal stack planning.* **(Refer section 10.2)**



15. *Consider the problem of swapping the contents of two registers A and B. Suppose that the single operator ASSIGN (x, v, iv, 0v) is available which assigns the values v, which is stored in location 1v to location 2 which previously contained the value 0v.*

    ASSIGN (x, v, 1v, 0v)

    P : CONTAINS (1v, v) ^ CONTAINS (x, 0v)

    D : CONTAINS (x, 0v)

    P : CONTAINS (x, 1v)

    Assume that there is atleast one additional register C available.

    a) What should STRIPS do with this problem 2 ?

    b) How might you design a program to solve this problem ? **(Refer section 10.2)**
16. *Write a note on reactive system.* **(Refer section 10.6)**

10.8 University Questions with Answers

Summer - 14

Q.1 *Explain goal stack planning using suitable example.* **(Refer section 10.2)** **[7]**

Summer - 16

Q.2 *Explain steps of natural language processing.* **(Refer section 10.1)** [7]

Q.3 *Write a short note on : Hopfield networks.* **(Refer section 10.1)** [7]

Summer - 18

Q.4 *Discuss nonlinear planning using constraint posting with exampl.* **(Refer section 10.4)** [7]

Winter - 18

Q.5 *Discuss goal stack planning.* **(Refer section 10.3)** [4]

❑❑❑

11 Statistical Learning Methods

Syllabus

Statistical Learning Methods - Statistical Learning, Learning with Complete Data, Learning with Hidden Variables ; EM Algorithm.

Contents

11.1 Introduction

Uncertain reasoning methods can be used for learning from observations. Agents can handle uncertainity by using the methods of probability and decision theory. For this, first they should learn their probabilistic theories of the world from experience.

11.1.1 Statistical Learning

In statistical learning, data are evidence, that is instantiation of some or all of the random variables describing the domain.

The hypothesis are probabilistic theories of how the domain works, including logical theories as a special case.

For example -

Suppose there are five kinds of bags of candies :

10 % are h_1 : 100 % cherry candies

20 % are h_2 : 75 % cherry candies + 25 % lime candies

40 % are h_3 : 50 % cherry candies + 50 % lime candies

20 % are h_4 : 25 % cherry candies + 75 % lime candies

10 % are h_5 : 100 % lime candies.

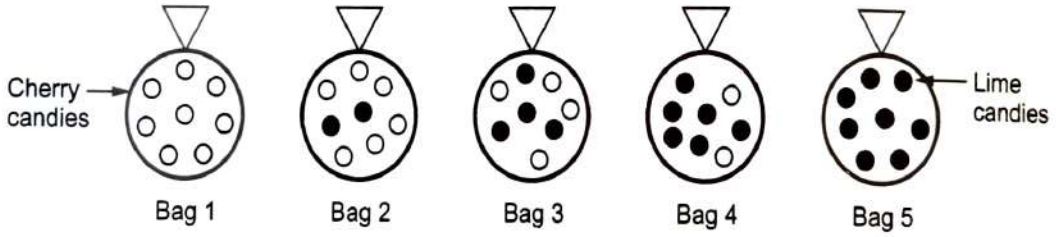


Fig. 11.1.1 The candy bags example

Then we observe candies drawn from some bag :

Fig. 11.1.2 Lime candies drawn from some bag

What kind of bag is it ? What flavour will the next candy be ?

Despite of its apparent triviality, this scenario serves to introduce many of the major issues. The agent really does need to infer a theory of its world, albeit a very simple one.

11.1.2 Bayesian Learning

1) It calculates the probability of each hypotheses, given the data and makes the predictions on that basis.

2) Predictions are made by using all the hypotheses, weighted by their probabilities, rather than a single "best" hypothesis. This way learning is reduced to probabilistic inference.

3) Let D represent all the data with observed value d, then the probability of each hypothesis obtained by Bayes' Rule -

$$P(h_i|d) = \alpha P(d|h_i) P(h_i) \quad \ldots (11.1.1)$$

If we want to make predicition about an unknown quantity X, then we have,

$$P(X|d) = \sum_i P(X|d, h_i) P(h_i|d)$$

$$= \sum_i P(X|h_i) P(h_i|d) \quad \ldots (11.1.2)$$

Where it is assumed that each hypothesis determines a probability distribution over X.

4) Equation (11.1.2) shows that predictions are weighted averages over the predictions of the individual hypotheses.

5) The important quantities in Bayesian learning are the hypothesis prior - $P(h_i)$ and the likelihood of the data under each hypothesis - $P(d|h_i)$.

6) The basic characteristic of Baysian learning is that "True hypothesis dominates the Bayesian prediction".

7) For any fixed prior that does not rule out the true hypothesis, the posterior probability of any false hypothesis will eventually vanish, simply because the probability of generating "uncharacteristic" data indefinitely is vanishingly small.

8) The Bayesian prediction is optimal whether the data set be small or large.

9) For real learning problems, the hypothesis space is usually very large or infinite.

10) Approximation in Bayesian learning : -

   i) A prediction can be made on the basis of single most probable hypothesis, h_i, that maximizes $P(h_i|d)$. This is called as maximum a posteriori or MAP hypothesis.

   ii) Predictions made according to an MAP hypothesis h_{map} are approximately Bayesian to the extent that $P(X|d) \approx P(X|h_{map})$.

   iii) Finding MAP hypothesis is often much easier than Bayesian learning, because it requires solving an optimization problem instead of a large summation.

iv) Overfitting Trade-offs :

a) Overfitting can occur when the hypothesis space is too expressive, so that it contains many hypotheses that fit the data set well.

b) Rather than placing an arbitrary limit on the hypotheses to be considered, Bayesian and MAP learning methods use the prior to penalize complexity.

c) Typically, more complex hypothesis have a lower prior probability-in part because there are usually many more complex hypotheses than simple hypotheses.

d) On the other hand, more complex hypotheses have a greater capacity to fit the data.

v) Hence, the hypothesis prior embodies a trade-off between the complexity of a hypothesis and its degree of fit to the data.

vi) If H contains only deterministic hypothesis, then in that case, $P(d|h_i)$ is 1 if h_i is consistent and 0 otherwise. Looking at equation (11.1.1) we see that h_{MAP} will then be the simplest logical theory that is consistent with the data. Therefore, maximum a posteriori learning provides a natural embodiment of Ockham's razor.

vii) a) Another trade-off between complexity and degree of fit is obtained by taking the logarithm of equation (11.1.1).

b) Choosing h_{MAP} to maximize $P(d|h_i)\ P(h_i)$ is equivalent to minimizing

$$-\log_2 P(d|h_i) - \log_2 P(h_i)$$

c) Using the connection between information encoding and probability we see that the $-\log_2 P(h_i)$ term equals the number of bits required to specify the hypothesis h_i.

d) $\log_2 P(d/h_i)$ is the additional number of bits required to specify the data given the hypothesis.

e) To see this, consider that no bits are required if the hypothesis predicts the data exactly as with h_5 and the string of lime candies and $\log_2 1 = 0$.

f) MAP learning is choosing the hypothesis that provides maximum compression of the data.

g) The same task is addressed more directly by the minimum description length or MDL, learning method, which attempts to minimize the size of hypothesis and data encodings rather than work with probabilities.

viii) a) Another approximation is provided by assuming a uniform prior over the space of hypotheses. In that case, MAP learning reduces to choosing an h_i that maximizes $P(d|H_i)$. This is called a **maximum-likelihood (ML)** hypothesis, h_{ML}.

b) Maximum-likelihood learning is very common in statistics, a discipline in which many researchers destruct the subjective nature of hypothesis priors. It is reasonable approach when there is no reason to prefer one hypothesis over another a priori. For example, when all hypotheses are equally complex.

c) It provides a good approximation to Bayesian and MAP learning when the data set is large, because the data swamps the prior distribution over hypotheses, but it has problems (all we shall see) with small data sets.

11.1.3 Learning with Complete Data

11.1.3.1 Maximum-Likelihood Parameter Learning : (Discrete Models)

Statistical learning methods have important task which is parameter learning with complete data. Parameter learning task involves finding the numerical parameters for a probability model whose structure is fixed.

Data is said to be complete when each data points contains values for every variable in the mode.

Consider candy-bag example : -

New manufacturer : Then the lime/Cherry proportions is completely unknown.

Parameter : $\theta \in [0, 1]$ (proportion of cherry)

Hypothesis : h_θ

Assumption : All proportions equally likely a priori.

BN's variables : Flavour $\in$ {Cherry, Lime}

N unwrapped candies : C cherries and l = N – C limes.

Likelihood of this particular data set :

$$P(d|h_\theta) = \prod_{j=1}^{N} P(d_j|h_\theta) = \theta^c (1 - \theta)^l$$

- Finding the maximum-likelihood hypothesis h_{ML} is then equivalent to maximising the log-likelihood :

$$L(d|h_\theta) = \log P(d|h_\theta)$$

$$= \sum_{j=1}^{N} \log P(d_j|h_\theta) = c \log \theta + l \log (1 - \theta)$$

To that end : 1) differentiate L with respect to θ and

2) set the resulting expression to 0.

$$\frac{dL(d|h_\theta)}{d\theta} = \frac{c}{\theta} - \frac{l}{1-\theta} = 0 \Rightarrow \theta = \frac{c}{c+l} = \frac{c}{N}$$

- Previous result is obvious, but the process is important : -
  1) Write down the expression for the likelihood of the data as a function of the parameter(s).
  2) Write down the derivative of the log-likelihood with respect to each parameter.
  3) Find the parameter values such that the derivatives are zero.

The last step can be tricky, using iterative solution algorithms or numerical optimisation.

- **Problem with ML learning :**

If some events have never been observed, h_{ML} assigns them 0 probability.

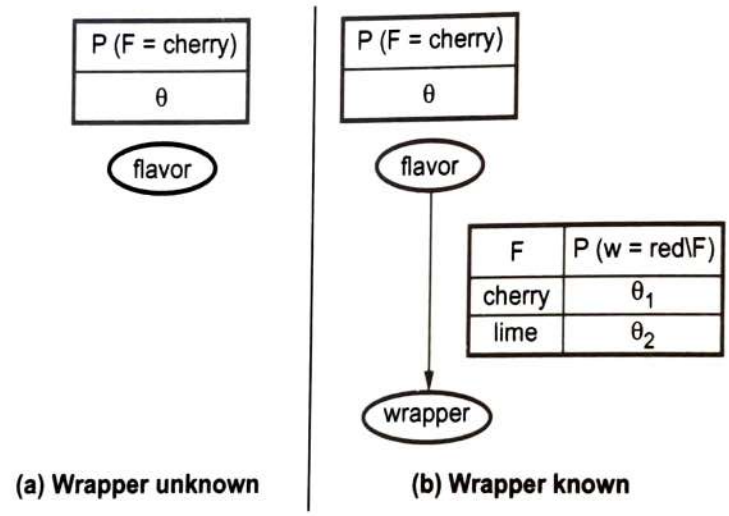


Fig. 11.1.3 Problem in ML Learning

New pb(b) : Wrappers red or blue. Colors selected probabilistically (see new BN)

$P(\text{Flavor} = \text{Cherry}, \text{Wrapper} = \text{green} \mid h_{\theta,\theta_1,\theta_2})$

$= P(\text{Flavor} = \text{cherry} \mid h_{\theta,\theta_1,\theta_2})\ P(\text{Wrapper} = \text{green} \mid \text{Flavor} = \text{Cherry}, h_{\theta,\theta_1,\theta_2})$

$= \theta \cdot (1 - \theta_1)$

Experiment :

$$N = c + l = (r_c + g_c) + (r_l + g_l)$$

Likelihood :

$$P(d|h_{\theta,\theta_1,\theta_2}) = \theta^c (1-\theta)^l \cdot \theta_1^{r_c} (1-\theta_1)^{g_c} \cdot \theta_2^{r_l} (1-\theta_2)^{g_l}$$

log-likelihood :

$$L = [c\log\theta + l\log(1-\theta)] + [r_c\log\theta_1 + g_c\log(1-\theta_1)] + [r_l\log\theta_2 + g_l\log(1-\theta_2)]$$

Derivatives :

$$\frac{dL}{d\theta} = \frac{c}{\theta} - \frac{l}{1-\theta} = 0 \qquad \Rightarrow \theta = \frac{c}{c+l}$$

$$\frac{dL}{d\theta_1} = \frac{r_c}{\theta_1} - \frac{g_c}{1-\theta_1} = 0 \qquad \Rightarrow \theta = \frac{r_c}{r_c + g_c}$$

$$\frac{dL}{d\theta_2} = \frac{r_1}{\theta_2} - \frac{g_l}{1-\theta_2} = 0 \qquad \Rightarrow \theta = \frac{r_l}{r_l + g_l}$$

With complete data, ML parameter learning for a BN decomposes into separate learning problems (one per parameter).

11.1.3.2 Naive Bayes Models

1) This is the most common Bayesian network model used in machine learning.
2) In this model, the "class" variable C (which is to be predicted) is the root and the "attribute" variables X_i are the leaves.
3) The model is "naive" because it assumes that the attributes are conditionally independent of each other, given the class.
4) Assuming Boolean variables the parameters are,

$$\theta = P(C = \text{true}), \theta_{i1} = P(X_i = \text{true} \mid C = \text{true}),$$

$$\theta_{i2} = P(X_i = \text{true} \mid C = \text{false})$$

The maximum - likelihood parameter values are found in exactly the same way as shown in Fig. 11.1.3 (b).

5) Once the model has been trained in this way, it can be used to classify new examples for which the class variable C is unobserved. With observed attribute values, $x_1, x_2, \ldots x_n$, the probability of each class is given by,

$$P(C \mid x_1, x_2, \ldots x_n) = \alpha \; P(C) \prod_i P(x_i \mid C)$$

6) A deterministic prediction can be obtained by choosing the most likely class.
7) The method learns fairly well but not as well as decision-tree learning; this is presumably because the true hypothesis - which is a decision tree - is not representable exactly using a naive Bayes model.
8) Naive Bayes learning do well in a wide range of applications; the boosted version is one of the most effective general-purpose learning algorithm.
9) Naive Bayes learning scales well to very large problems : with n Boolean attributes, there are just 2n+1 parameters, and no search is required to find h_{ML}, the maximum - likelihood naive Bayes hypothesis.

10) Naive Bayes learning has no difficulty with noisy data and can give probabilistic predictions when appropriate.

11.1.3.3 Maximum-Likelihood Parameter Learning : (Continuous Models)

1) Continuous probability model such as the linear-Gaussian model is used for maximum - likelihood parameter learning.

2) Because continuous variables are ubiquitous in real-world applications, it is important to know how to learn continuous models from data.

3) The principles for maximum likelihood learning are identical to those of the discrete case.

4) a) Let us begin with a very simple case : Learning the parametes of a Gaussian density function on a single variable.

b) That is, the data are generated as follows : -

$$P(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

The parameters of this model are the mean μ and the standard deviation σ. (Notice that the normalizing "constant" depends on σ, so we cannot ignore it.)

c) Let the observed values be $x_1, \ldots, x_N$. Then the log likelihood is,

$$L = \sum_{j=1}^{N} \log \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x_j-\mu)^2}{2\sigma^2}}$$

$$= N(-\log \sqrt{2\pi} - \log \sigma) - \sum_{j=1}^{N} \frac{(x_j-\mu)^2}{2\sigma^2}$$

d) Setting the derivatives to zero as usual, we obtain,

$$\frac{\delta L}{\delta \mu} = -\frac{1}{\sigma^2} \sum_{j=1}^{N} (x_j - \mu) = 0 \qquad \Rightarrow \mu = \frac{\sum_j x_j}{N}$$

$$\frac{\delta L}{\delta \sigma} = -\frac{N}{\sigma} + \frac{1}{\sigma^3} \sum_{j=1}^{N} (x_j - \mu)^2 = 0 \quad \Rightarrow \sigma = \sqrt{\frac{\sum_j (x_j - \mu)^2}{N}}$$

5) The maximum-likelihood value of the mean is the sample average and the maximum-likelihood value of the standard deviation is the square root of the sample variance. Again, these are comforting results that confirm "commonsense" practice.

6) a) Now consider a linear Gaussian model with one continuous parent X and a continuous child Y.

b) Y has a Gaussian distribution whose mean depends linearly on the value of X and whose standard deviation is fixed.

c) To learn the conditional distribution $P(Y|X)$, we can maximize the conditional likelihood.

$$P(y|x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(y-(\theta_1 x+\theta_2))^2}{2\sigma^2}}$$

d) Here, the parameters are θ_1, θ_2 and σ. The data are a collection of (x_j, y_j) pairs, as shown in Fig. 11.1.4.

e) Using the usual methods, we can find the maximum-likelihood values of the parameters.

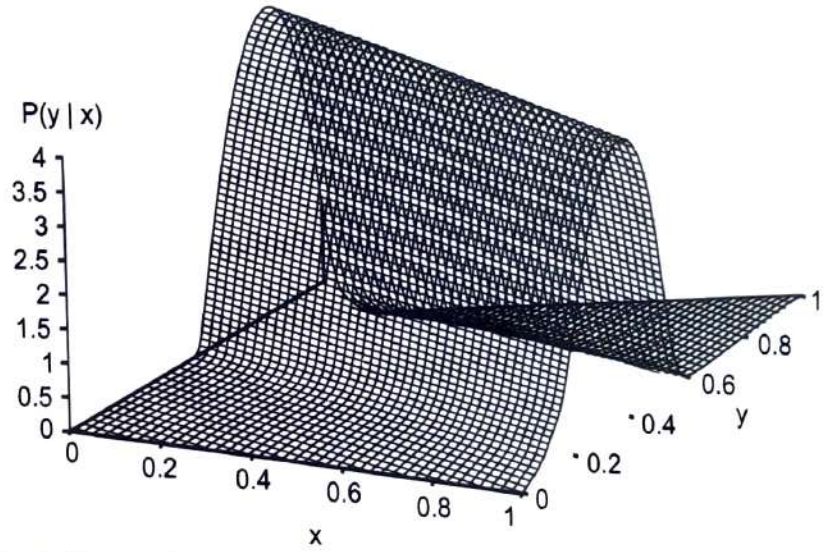


Fig. 11.1.4 (a) A linear Gaussian model described as $y = \theta_1 x + \theta_2$ plus Gaussian noise with fixed variance

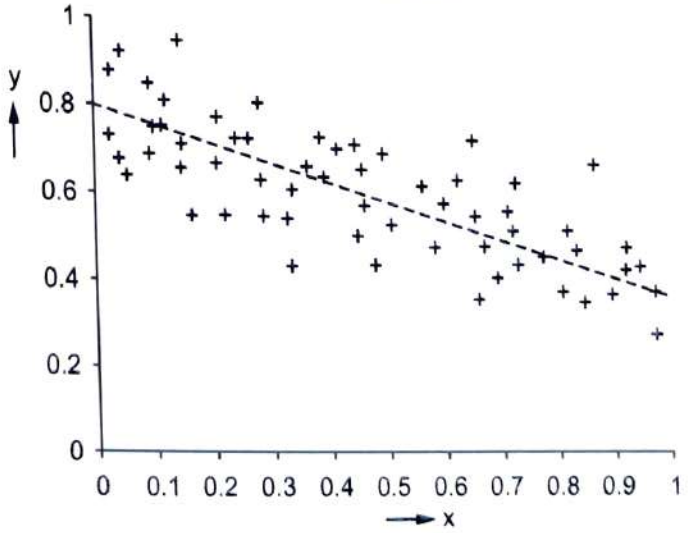


Fig. 11.1.4 (b) A set of 50 data points generated from model in Fig. 11.1.4 (a)

f) Here we want to make a different point. If we consider just the parameters θ_1 and θ_2 that define the linear relationship between x and y, it becomes clear that maximizing the log-likelihood with respect to these parameters is the same as minimizing the numerator in the exponent of above equation.

$$E = \sum_{j=1}^{N} (y_j - (\theta_1 x_j + \theta_2))^2$$

g) The quantity $(y_j - (\theta_1 x_j + \theta_2))$ is the error for (x_j, y_j) that is, the difference between the actual value y_i and the predicted value $(\theta_1 x_j + \theta_2)$. So E is the well-known sum of squared errors.

h) This is the quantity that is minimized by the standard linear regression procedure.

i) Minimizing the sum of squared errors gives the maximum - likelihood straight line model, provided that the data are generated with Gaussian noise of fixed variance.

11.1.3.4 Bayesian Parameter Learning

ML learning is simple, but not appropriate for small data sets.

Example -

If only cherries have been observed, $h_{ML} \rightarrow \theta = 1.0$

- Bayesian Approach :
  - It uses hypothesis prior over possible values of the parameters.
  - Update of this distribution is used as the data arrives.
- Candy example with Bayesian view : -
  - θ : unknown value of a variable Θ.
  - Hypothesis prior : P (Θ). (Continuous over [0, 1] and integrating to 1).

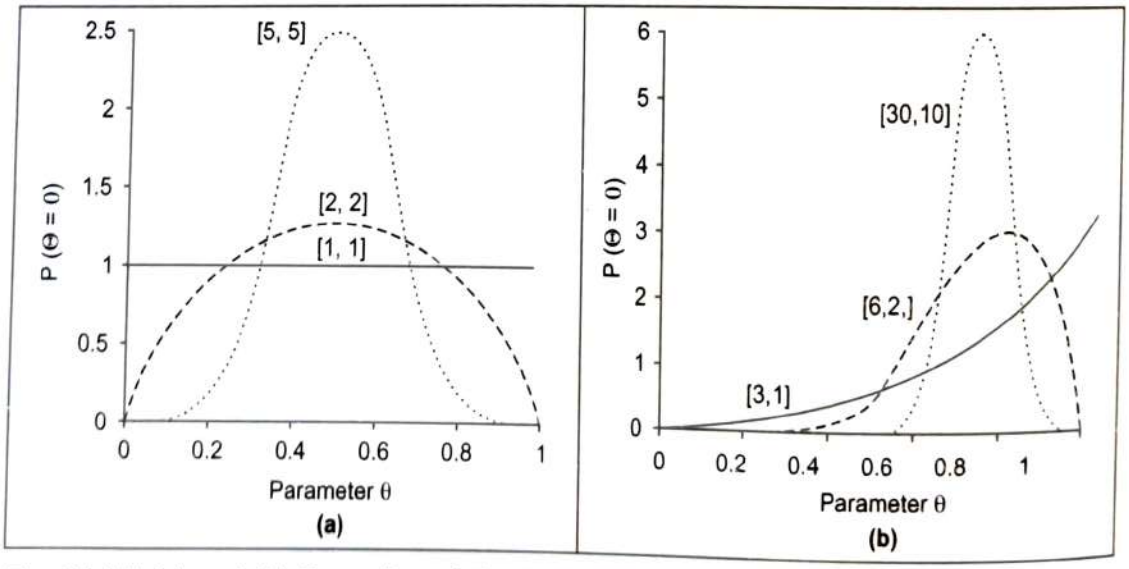


Fig. 11.1.5 (a) and (b) Examples of the beta [a, b] distribution for different values of [a, b]

- Candidates :
  beta distributions, defined by 2 hyperparameters a and b

  such that :

  $$\text{beta}[a, b](\theta) = \alpha\, \theta^{a-1} (1-\theta)^{b-1}$$

- Nice property of the beta family :

If Θ has prior beta [a, b] and a data point is observed, then the posterior for Θ is also a beta distribution.

Beta family :

A beta family is called as the **conjugate prior** for the family of distributions for a Boolean variable.

$$\begin{aligned} P(\theta \mid D_1 = \text{Cherry}) &= \alpha\, P(D_1 = \text{Cherry} \mid \theta)\, P(\theta) \\ &= \alpha' \theta \cdot \text{beta}[a, b](\theta) = \alpha'\, \theta \cdot \theta^{a-1} (1-\theta)^{b-1} \\ &= \alpha'\, \theta^{a} (1-\theta)^{b-1} = \text{beta}[a+1, b](\theta) \end{aligned}$$

Note : a and b are virtual counts (starting with beta [1, 1]).

With wrappers : 3 parameters. It need to specify $P(\theta, \theta_1, \theta_2)$.

Assuming parameter independence : $P(\theta, \theta_1, \theta_2) = P(\theta)\, P(\theta_1)\, P(\theta_2)$

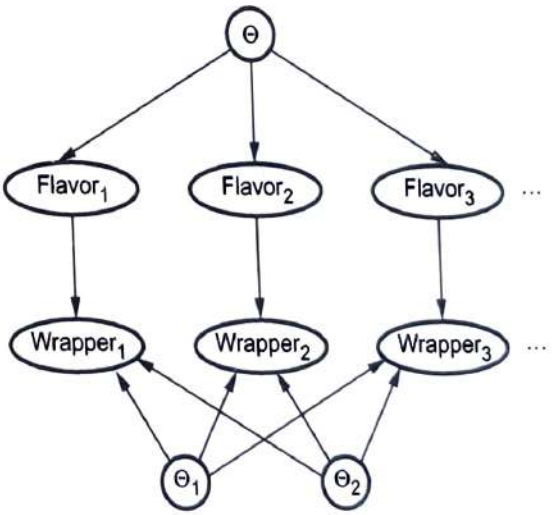


Fig. 11.1.6 A Bayesian network that corresponds to a Bayesian learning process. Posterior distributions for the parameter variables Θ, Θ_1, Θ_2 can be inferred from their prior distributions and the evidence in the flavour$_i$, wrapper$_i$ variables

- **View of Bayesian learning as inference in a BN.**

Variables : Unknown parameters + variables describing each instance.

$$P(\text{Flavor}_i = \text{Cherry} \mid \Theta = \theta) = \theta$$

$$P(\text{Wrapper}_i = \text{red} \mid \text{Flavor}_i = \text{Cherry}, \Theta_1 = \theta_1) = \theta_1$$

11.1.3.5 Learning Bayes Net Structures

Often the Bayes net structures are easy to get from expert knowledge. But sometimes the causality relationships are debatable.

For example :

Smoking ⇒ Cancer ?

too Much TV ⇒ Bad at school ?

- To search for a good model :
  - Start with a linkless model and add parents to each node, then learn parameters and measure accuracy of the resulting model.
  - Start with an initial guess of the structure, and use hill-climbing or simulated annealing to make modifications (reversing, adding, or deleting arcs).

Note : To avoid cycles, many algorithms use an ordering over nodes to constrain arcs' orientations.

- Two ways for deciding when a good structure has been found : -
  1) Test whether the conditional independence assertions implicit in the structure, are satisfied in the data.

     $$P(Fri \mid Sat, Bar \mid WillWait) = P(Fri \mid Sat \mid WillWait)\ P(Bar \mid WillWait)$$

     It requires an appropriate statistical test (with appropriate threshold).
  2) Measure the degree to which the proposed model explains the data. The ML hypothesis would give a fully connected network.
- Here we need to penalize complexity : -
  a) MAP (MDL) approach : Substracts a penalty from the likelihood of each structure.
  b) Bayesian approach : places a joint prior over structures and parameters.

     It too many structures are there then use sampling rather than exact methods.

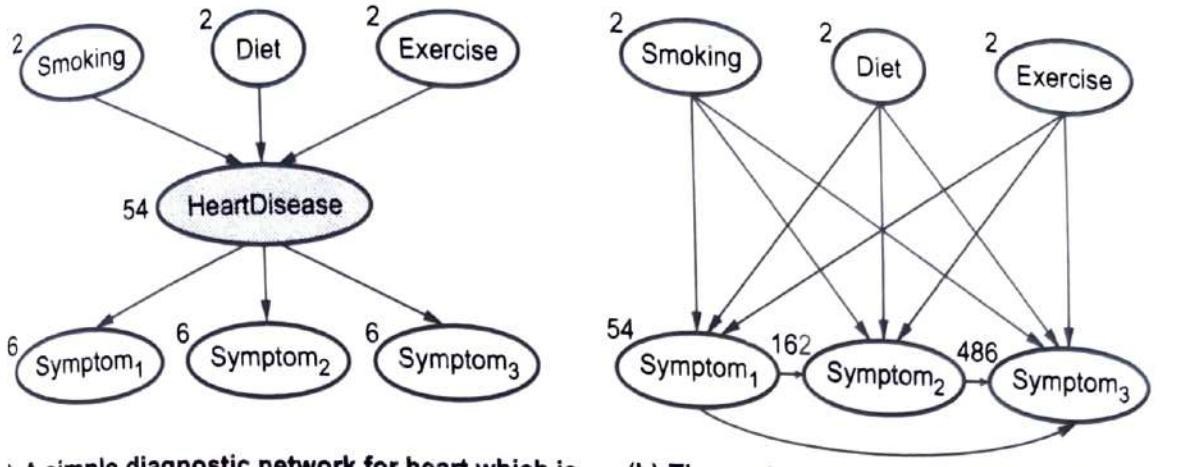
11.1.4 Learning with Hidden Variables

11.1.4.1 Introduction

Many real world problems have hidden (or latent) variables. There is no training data variable. The model cannot be built without training data and inference can not be done without model. So we have a problem in hand that-How to learn models with hidden variables ?

For example :

Example : Medical records include observed symptoms, treatment applied and outcome of the treatment, but rarely the disease.



(a) A simple diagnostic network for heart which is assumed to be a hidden variable. Each variable has three possible values. Each veriable is labeled with the number of independent parameters in its conditional distribution; the total number is 78

(b) The equivalent network with heart disease removed. Notice that symptom variables are no longer conditionally independent given their parents. This network requires 708 parameters

Fig. 11.1.7

Question : Why include the disease in the model ?

- Each node has 3 values : None, moderate, severe.
- Model with hidden variable has 78 parameters.
- Model without hidden variable has 708 parameters.
- Hidden variables allow simpler models.

11.1.4.2 Unsupervised Clusering : Learning Mixtures of Gaussians

Unsupervised clustering is the problem of discerning multiple categories in a collection of objects.

It is "unsupervised" because no "category" label is given.

Example :

- "Red giant" or "white dwarfs" are categories created according to characteristics of stars (spectrum, size ...).
- **Linnaean** taxonomy of organisms : species, genera, orders ...
- **Probabilistic mixture models :**
  - Consider a mixture model of the form

$$P(X|\pi,\theta) = \sum_{h=1}^{k} \Pi_h P(X \mid \theta_h)$$

a) $P(X \mid \theta_h)$ is the h^{th} mixing component.

b) Π_h is the mixing weight.

– Mixing component :

a) $P(X \mid \theta_h)$ is a density function with parameter θ_h.

b) Example : Gaussians, multinomials, Bernoulli.

c) Each component corresponds to one cluster.

– Mixing weight :

a) Forms a probability distribution over components, $\sum_h \Pi_h = 1$.

b) Relative proportion of each cluster.

- **The mixture model learning problem : -**

– Given : Data $\chi = \{X_1, \ldots, X_n\}$

– Assume :

a) χ is generated by a mixture model.

b) k is the number of components in the mixture model.

c) $P(X \mid \theta_h)$ is from a known (exponential) family.

– A point X is generated as follows :

a) Sample component h with probability Π_h.

b) Sample X with probability $P(X \mid \theta_h)$

– Problem : Find (π, θ) that maximizes

$$P(\chi \mid \pi, \theta) = \prod_{i=1}^{n} P(X_i \mid \pi, \theta) = \sum_{i=1}^{n} \log P(X_i \mid \pi, \theta)$$

– (π^*, θ^*) best explains the observed data.

This can be used as a model for the data domain.

- **EM algorithm for mixture models :**
  - Assume χ is generated from a mixture of Gaussians.
  - Each point X generated from one component.
    a) It is unknown that which component generated X.
    b) It is unknown what the (μ_h, Σ_h) of each component is
  - If component assignments were known,
    a) It can estimate (μ_h, Σ_h) using ML estimates.
    b) It can estimate Π_h using ML estimate.
  - If parameters $\{\Pi_h, (\mu_h, \Sigma_h)\}$ were known

a) It can estimate (probabilistic) component assignments $P(h|x)$.

- Learning mixture models

a) Pretend component parameters $\{\Pi_h, (\mu_h, \Sigma_h)\}$ are known.

b) Estimate component assignments $P(h|x)$ for every point.

c) Estimate parameters $\{\Pi_h, (\mu_h, \Sigma_h)\}$ based on current assignments.

d) Repeat till convergence.

- **EM Algorithm :**
  - **Initialize :** component parameters $\{(\Pi_h, (\mu_h, \Sigma_h))\}$
  - **E-step :** compute $P(h|x_i)$

    Bayes Rule, $P(h|x_i) = P_{hi} = \propto \Pi_h P(x_i/\mu_h, \Sigma_h)$
  - **M-step :** compute component parameters

$$P_h = \sum_i P_{hi}$$

$$\Pi_h = \frac{P_h}{\sum_{h'} P_{h'}}$$

$$\mu_h = \sum_i P_{hi} X_i / P_h$$

$$\sum_h = \sum_i P_{hi} (X_i - \mu_h) (X_i - \mu_h)^T / P_h$$

  - **Increase log-likelihood** at every iteration.
  - **It converges to local minimum** (mostly).
  - **Issues :**

    a) Degenerate solutions can give high likelihood.

    b) Local minima (or saddle point) may be very bad.

- **Learning mixtures of Gaussians : -**

The question is that what probability distribution has generated the data ?

The answer is,

Clustering → mixture distribution P : k components, each being a distribution.

$$P(X) = \sum_{i=1}^{k} P(C = i) P(X | C = i)$$

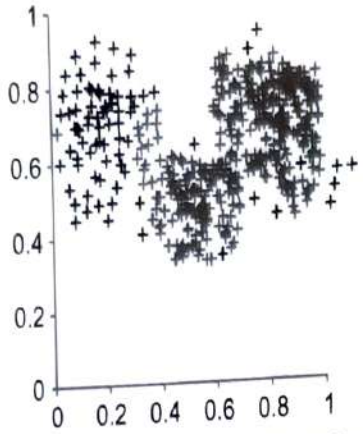
Data point generation : -

1) Choose a component, then

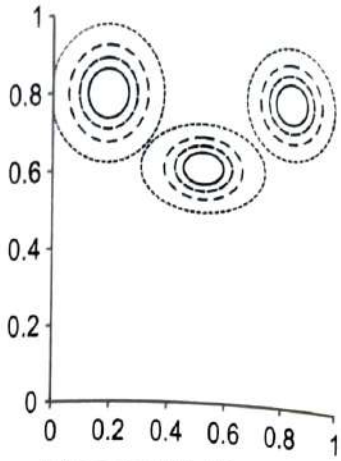
2) Sample from that component.

For continuous data : a multivariate Gaussian for each component is used (mixture of Gaussians family).

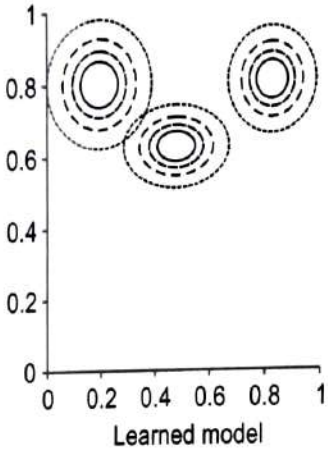
- (Original mixture of Gaussians used for generating the 500 points).



(a) 500 datapoints in two dimensions, suggesting the presence of three clusters



(b) A Gaussian mixture model with three components. The weigths (left to right) are 0.2, 0.3 and 0.5. The data in 11.1.8 (a) were generated from this



(c) The model reconstructed by EM from the data in 11.1.8 (b)

Fig. 11.1.8

Parameters (for each component) :

$$w_i = P(C = i), \mu_i, \Sigma_i$$

Problems in learning :

- We do not know from which component each data point comes.
- With the parameters, we could compute $P(C|X)$.

Basic idea of EM, is iterate until convergence :

- Pretend we know the parameters.
- Infer $P(C|X)$ for each data point.
- Refit each component i to data. Points (weighted by $P(C = i|X)$)

- **For mixtures of Gaussians :**

1) E-step : Compute

$$P_{ij} = P(C = i \mid X_j)$$

$$= \alpha\ P(X_j \mid C = i)\ P(C = i)$$

Define, $P_i = \Sigma_i\ P_{ij}$

2) M-step :

$$\vec{\mu}_i \leftarrow \frac{1}{P_i} \sum_j P_{ij}\ X_j$$

$$\Sigma_i \leftarrow \frac{1}{P_i} \sum_j P_{ij}\ (X_j - \vec{\mu}_i)\ (X_j - \vec{\mu}_i)^T$$

$$w_i \leftarrow P_i$$

- E-step computes the expected values P_{ij} of hidden indicator variables.
- M-step finds parameter values maximizing the log-like lihood of the data.

Z_{ij} (= 1 if datum X_i generated by i^{th} component)

- Comments :
  - Log-likelihood of learned model exceeds that of original model !
  - EM increases the log-likelihood at each iteration. (This fact can be proved in general). In certain conditions, EM can be proven to reach a local maximum.
  - EM resembles a gradient-based hill-climbing (with no step size).
- Bad Behaviours :
  - A Gaussian component covers only one data point.

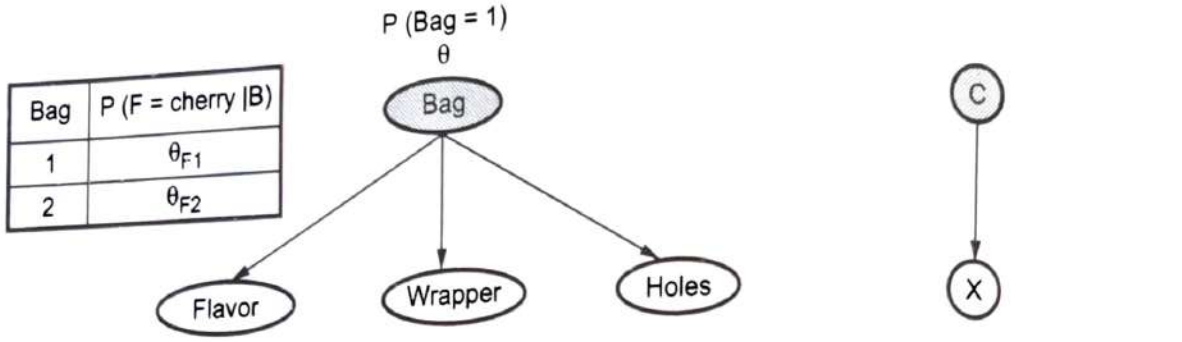
    (Varience $\rightarrow$ 0, likelihood $\rightarrow \infty$)
  - 2 components are merged.

    (same data points, same mean and varience).
- Solutions :
  - MAP version of EM (with priors on parameters), or
  - restart a component with random parameters.
- About initialisation :
  - Reasonable values, and
  - Random ones components have to be distinct !!!).

11.1.4.3 Learning BN with Hidden Variables

Problem :

- 2 bags of candies mixed together.
- Candies have 3 features : Flavour, Wrapper, Hole (see below Fig. 11.1.9).
- Distribution in each bag : Naive Bayes model.



Bag	P (F = cherry |B)
1	θ_{F1}
2	θ_{F2}

(a) A mixture model for candy. The proportions different flavours, wrappers and numbers of holes depend on the bag, which is not observed

(b) Bayesian network for a gaussian mixture. The mean and covariance of the observable variables X depend on the component C.

Fig. 11.1.9

Can we recover composition of the 2 bags ?

Let's apply EM.

1000 data samples generated with parameters : -

$$\theta = 0.5, \theta_{F1} = \theta_{W1} = \theta_{H1} = 0.8, \theta_{F2} = \theta_{W2} = \theta_{H2} = 0.3$$

	W = Red		W = Green	
	H = 1	H = 0	H = 1	H = 0
F = Cherry	273	93	104	90
F = Lime	79	100	94	167

Initially : $\theta^{(0)} = 0.6$,

$$\theta_{F1}^{(0)} = \theta_{W1}^{(0)} = \theta_{H1}^{(0)} = 0.6,$$

$$\theta_{F2}^{(0)} = \theta_{W2}^{(0)} = \theta_{H2}^{(0)} = 0.4$$

For θ : Cannot count candies from bags 1 and 2 $\rightarrow$ expected counts :

$$\theta^{(1)} = \frac{\hat{N}\,(\text{Bag} = 1)}{N}$$

$$= \frac{1}{N} \sum_{j=1}^{N} P(\text{Bag} = 1 \mid \text{Flavor}_j, \text{Wrapper}_j, \text{holes}_j)$$

$$= \frac{1}{N} \sum_{j=1}^{N} \frac{P(f_j \mid \text{Bag}=1)\, P(W_j \mid \text{Bag}=1)\, P(h_j \mid \text{Bag}=1)\, P(\text{Bag}=1)}{\sum_i P(f_j \mid \text{Bag}=i)\, P(W_j \mid \text{Bag}=i)\, P(h_j \mid \text{Bag}=i)\, P(\text{Bag}=i)}$$

With the 273 red-wrapped cherry candies :

$$\frac{273}{1000} \cdot \frac{\theta_{F1}^{(0)} \theta_{W1}^{(0)} \theta_{H1}^{(0)} \theta^{(0)}}{\theta_{F1}^{(0)} \theta_{W1}^{(0)} \theta_{H1}^{(0)} \theta^{(0)} + \theta_{F2}^{(0)} \theta_{W2}^{(0)} \theta_{H2}^{(0)} (1 - \theta^{(0)})} \simeq 0.22797$$

$$\theta^{(1)} = 0.6124$$

For other parameters, as θ_{F1}, the expected count of cherry candies from bag 1 is :

$$\sum_{j:\, \text{Flavor}_j = \text{Cherry}} P(\text{Bag} = 1 \mid \text{Flavor}_j = \text{Cherry}, \text{wrapper}_j, \text{holes}_j)$$

Again, with a Bayes Net algorithm, we get :

$$\theta^{(1)} = 0.6124$$

$$\theta_{F1}^{(1)} = 0.6648 \qquad \theta_{F2}^{(1)} = 0.3886$$

$$\theta_{W1}^{(1)} = 0.6483 \qquad \theta_{W2}^{(1)} = 0.3817$$

$$\theta_{H1}^{(1)} = 0.6558 \qquad \theta_{H2}^{(1)} = 0.3827$$

Log-likelihood increases from – 2044 to – 2021 (likelihood $\times 10^{10}$)

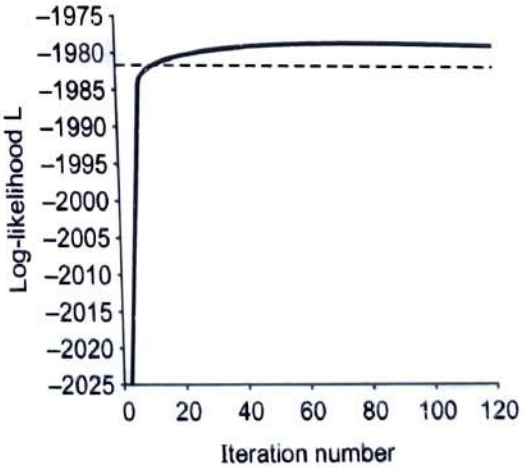


Fig. 11.1.10 Graph showing the log-likelihood of the data L, as a function of them iteration. Graph for the Bayesian network in Fig. 11.1.9 (a)

- After 10 iterations :
  - Better likelihood than original model.
  - Very slow progress.

    (gradient-based algorithm to speed-up learning).
- Bayesian network learning with hidden variables : -
  - Repeat : do inference, update parameters (based on inference).
  - Only local posterior probabilities are needed for each parameter.

General case :

$$\theta_{ijk} = P(X_i = X_{ij} \mid P_{a_i} = Pa_{ik})$$

$$\leftarrow \hat{N}(X_i = X_{ij}, P_{a_i} = pa_{ik}) \mid \hat{N}(P_{a_i} = pa_{ik})$$

- Learning HMMs :
  - Unrolled dynamic Bayesian network.
  - Estimate transition probabilities from observation sequences.
  - The model does not change $\theta_{ijt} = \theta_{ij}$.
  - Initialize transition probabilities θ_{ij}.
  - Estimate the number of times a transition happens.
- E-step, can be solved by inference.
  - Estimate the parameters of the HMM.
- M-step, simple ML estimates.
  - Baum-Welch Algorithm

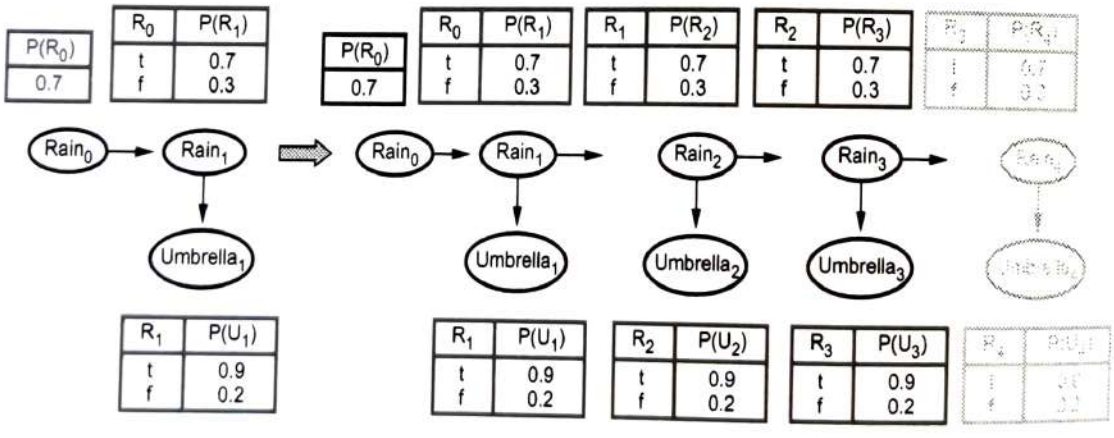


Fig. 11.1.11 An unrolled dynamic Bayesian network that represents a hidden Markov model

11.1.4.4 General Form of EM Algorithm

- Compute expected values of hidden variables.
- Recompute parameters.
- Model has known and hidden components (X, Z).
- The log-likelihood of the complete model L(X, Z).
  - But Z is a random variable.
  - For a given $\theta^{(t)}$, $P(Z = z|x, \theta^{(t)})$ is known.

With observed variable x, hidden variables Z and parameters $\vec{\theta}$, EM algorithm computes,

$$\theta^{(t+1)} = \text{argmax}_\theta \sum_z P(Z = z|x,\theta^{(t)})\ L(x, Z = z|\theta)$$

E-step : Summation (computes the expected log-likelihood of the "completed" data)

M-step : Maximisation (of this log-likelihood).

[**Note :** Important step : Identifying hidden variables.]

Computing the expectations :

- May very straight forward, example - Mixture models.
- May need exact inference algorithms, example - Bayes nets.
- May need approximate inference such as Gibbs sampling.

- Learning BN with hidden variables :

Here there are 2 combined difficulties,

1) Simple case : List of hidden variables known.

   Example - learn the structure knowing that HeartDisease exists.

2) With no or incomplete knowledge of hidden variables : One has to learn structure with possibility to add or remove variables.

Note : When inventing variables, the algorithm does not know what it represents (can not call it HeartDisease ...)

But a human expert may be able to interpret the network and identify variables.

As with full observability, a complexity penalty should be used (to avoid fully connected BNs).

Upto now, process with two loops : -

Outer loop : Structural search.

Inner loop : EM (+ gradient ?), including NP-hard problem of computing posteriors in a BN.

A more practical approach is structural EM : -

EM with possibility to update structure as well as parameters.

11.2 Neural Networks

GTU : Summer-18,19,20, Winter-18,19

Introduction

A neuron is a cell in the brain whose principle function is the collection, processing and dissemination of electrical signals.

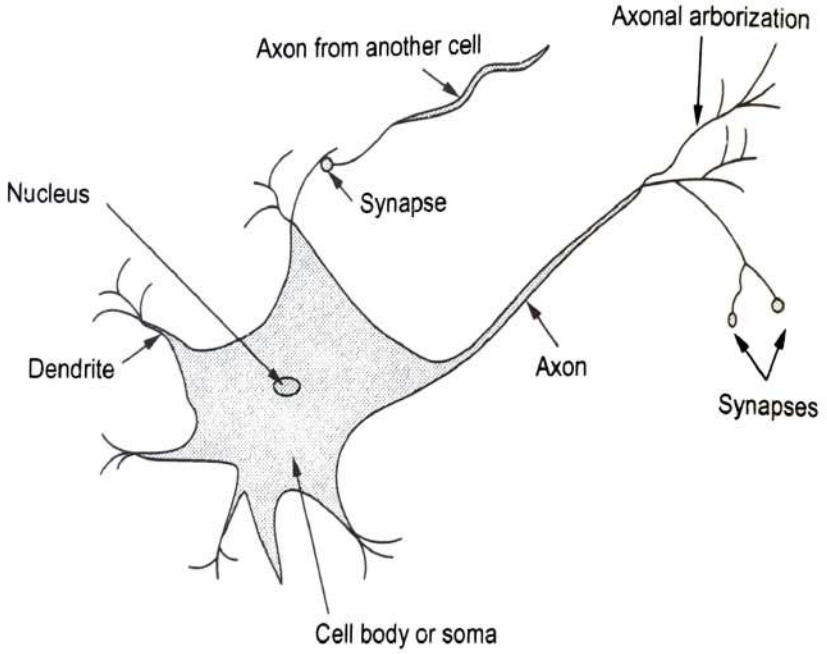


Fig. 11.2.1 Schematic diagram of a typical neuron

Note *[The brain has 10^{11} neurons of greater than 20 types, 10^{14} synapses, 1 ms - 10 ms cycle time. The signals are noisy "spike trains" of electrical potential. This method works for learning. The models have been developed based on neurons.]*

The brain's information-processing capacity is thought to emerge primarily from networks of such neurons. For this reason, some of the earliest AI work aimed to create artificial neural networks.

- McCulloch - Pitts (1943)

It is the basis of neural networks which is mathematical model of neuron as shown in following Fig. 11.2.2.

Roughly speaking, it "fires" when a linear combination of its inputs exceeds some threshold.

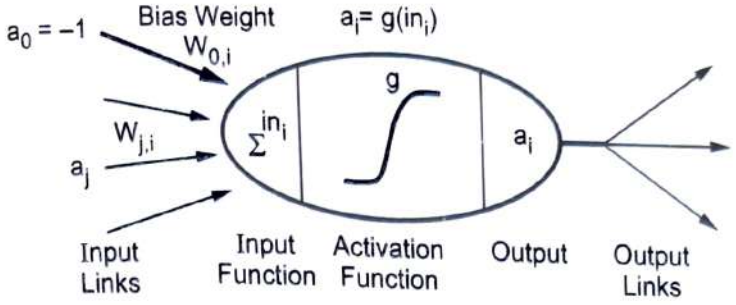


Fig. 11.2.2 Mathematical model of Neuron

The output is a "squashed" linear function (mostly weighted sum) of the inputs. A gross oversimplification of real neurons, but its purpose is to develope understanding of what networks of simple units can do.

- **Various parts in neural networks : -**
  - i) Neural networks are composed of nodes or units connected by directed links.
  - ii) A link from unit j to unit i serves to propagate the activation a_j from j to i.
  - iii) Each link also has a numeric weight $W_{j,i}$ associated with it, which determines the strength and sign of the connection.
  - iv) Each unit i first computes a weighted sum of its inputs :

$$in_i = \sum_{j=0}^{n} W_{j,i}\, a_j$$

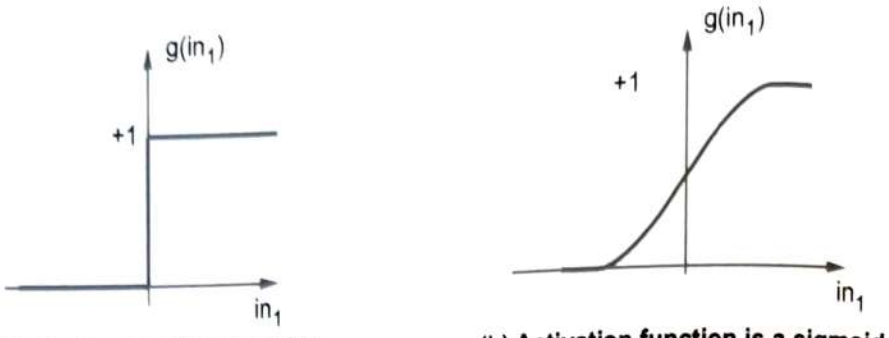
  - v) Then it applies an activation function g to this sum to derive the output :

$$a_i = g(in_i) = g\left(\sum_{j=0}^{n} W_{j,i}\, a_j\right)$$

  - vi) Notice that we have included a bias weight $W_{0,i}$ connected to a fixed input $a_0 = -1$.

- Activation function :

The activation function g is designed to meet two things. First, we want the unit to be "active" (near + 1) when the "right" inputs are given and "inactive" (near 0) when the "wrong" inputs are given. Second, the activation needs to be nonlinear, otherwise the entire neural network collapse into a simple linear function.



(a) Activation function is a step function or threshold function.

(b) Activation function is a sigmoid function $1/(1 + e^{-x})$.

Fig. 11.2.3 Activation function example

Changing the bias weight $W_{0,i}$ moves the threshold location.

- Implementing logical functions :

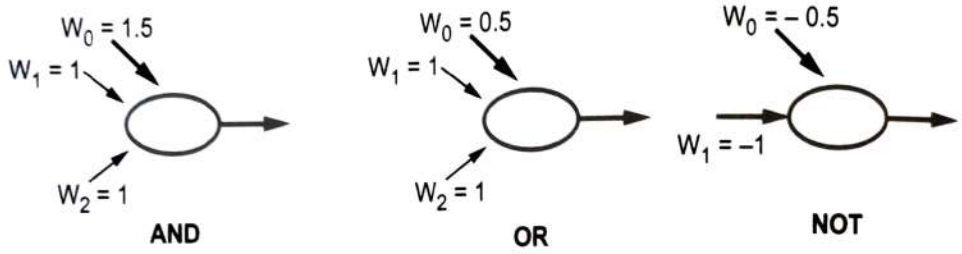


Fig. 11.2.4 Logical functions implementations

McCulloch and Pitts original motivation is that, every Boolean function can be implemented.

- **Network structure :**

1) Feed-forward networks :

These Acyclic networks, called as feed-forward networks implement functions and they have no internal state. There can be, single-layer perceptrons or multi-layer perceptrons feed-forward networks.

2) Recurrent networks :

Recurrent neural nets have directed cycles with delays.

This have internal state (like flip-flops), they can oscillate, reach stability, or even expose chaotic behaviour, etc.

Examples :

i) Hopfield networks have symmetric weights ($W_{i,j} = W_{j,i}$).

ii) $g(x) = \text{sign}(X)$, $a_i = \pm 1$, holographic associative memory.

iii) Boltzmann machines use stochastic activation functions.

- **Feed-forward example :**

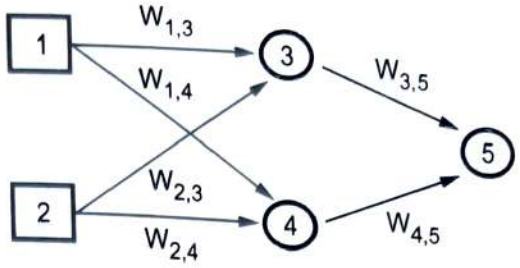


Fig. 11.2.5 Feed forward network [Note : Bias is ignored in this example]

- Feed-forward network = A parameterized family of nonlinear functions : -

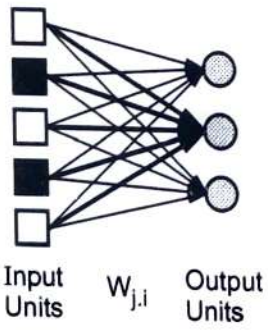
$$a_5 = g(W_{3,5}\, a_3 + W_{4,5}\, a_4)$$

$$= g(W_{3,5}\, g(W_{1,3}\, a_1 + W_{2,3}\, a_2) + W_{4,5}\, g(W_{1,4}\, a_1 + W_{2,4}\, a_2))$$

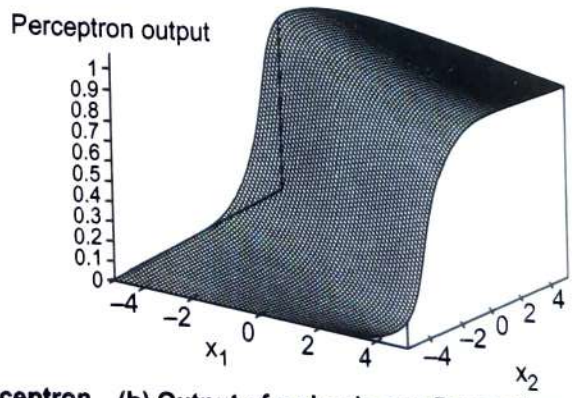
- Adjusting the weights changes the function.
- We can simulate learning by stepwise adjusting the weights.

- **Single-layer perceptrons :**

Feed-forward networks are normally arranged in layers. The simplest one having only one layer, no hidden layers is called as perceptron network.



(a) A perceptron network consisting of three perceptron output units that share five inputs. Note that, in second output unit, weights on its incoming links have no effects on the other output units.



(b) Output of a simple one-Perception unit with two input value

Fig. 11.2.6 Single layer perceptrons

The output units all operate separately and there are no shared weights.

Adjusting weights moves the location, orientation and steepness of cliff.

- **Expressivenes of perceptrons :**

Consider a perceptron with g = step function (Rosenblatt, 1957, 1960). It can represent AND, OR, NOT, majority, etc. but not XOR. It represents a linear separator in input space.

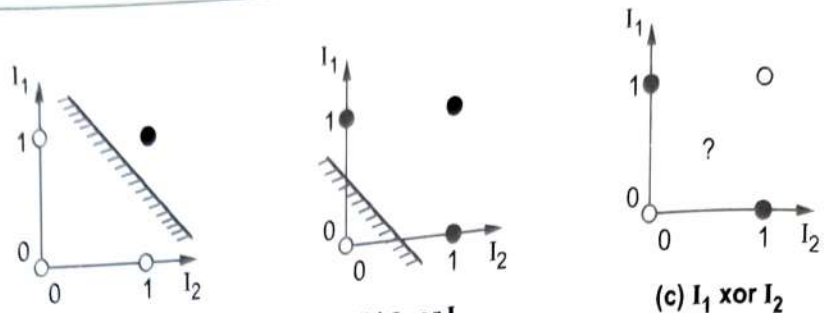


Fig. 11.2.7 Linear separability in threshold perceptrons. Black dots indicate a point in the input space where the value of the function is 1, and white dots indicate a point where the value is 0. The perceptron returns 1 on the region on the non-shaded side of the line. In (c) no such line exists that correctly classifies the inputs

The insufficiency of perceptrons, pointed out by Minsky and Papert (1969).

- **Perceptron learning :**

Learn by adjusting weights to minimize squared error on training set. The squared error for an example with input x and true output y is,

$$E = \frac{1}{2} Err^2 \equiv \frac{1}{2} (y - hw(x))^2$$

Perform optimization search by gradient descent of E (the gradient of a vector gives the direction of the steepest descent).

For us this means, we have to calculate for each weight,

$$\frac{\delta E}{\delta W_j} = Err \times \frac{\delta Err}{\delta W_j}$$

$$= Err \times \frac{\delta}{\delta W_j} g\left(y - \sum_{j=0}^{n} W_j x_j \right)$$

$$= - Err \times g'(in) \times x_j$$

Here, g′ is the derivation of the activation function, example - for sigmoid,

$g' = g(1 - g)$. This leads to a simple weight update rule.

$$W_j \leftarrow W_j + \propto \times Err \times g'(in) \times x_j$$

Example :

Positive error → Increase network

Output → Increase weights on positive inputs, decrease on negative inputs.

- The perceptrons learning algorithm :

Function PERCEPTRONS-LEARNING (examples, network) returns a perceptrons hypothesis

Inputs : Examples, a set of examples, each with input $X = x_1, \ldots, x_n$ and output y.

Network, a perceptrons with weights W_j, $j = 0 \ldots n$ and activation function g

Repeat.

For each e in examples do.

$$in \leftarrow \sum_{j=0}^{n} W_j x_j[e]$$

$$Err \leftarrow y[e] - g(in)$$

$$W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j[e]$$

until some stopping criterion is satisfied.

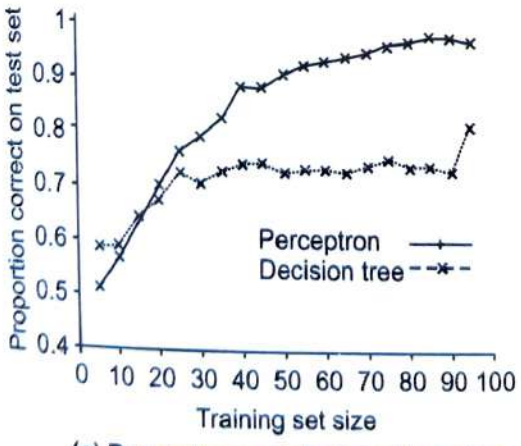
return NEURAL-NET-HYPOTHESIS (Network).

Remarks on perceptrons learning algorithm :-

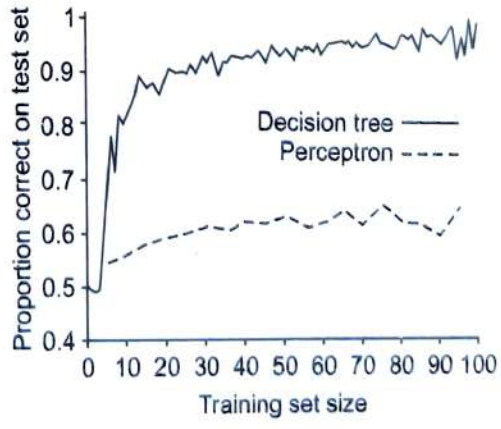
- For threshold functions (where g′ cannot be calculated), g′ is simply committed. Since g′ is the same for all weight, it only changes the magnitude but not the direction of the update for each example.
- Each cycle through the examples is called an epoch.
- The stop criterion is typically that the weight changes become very small.

- **Perceptron learning (diagramatic representation)**

Perceptron learning rule converges to a consistent function for any linearly separable data set.



(a) Perceptrons are better at learning the majority function of 11 inputs



(b) Decision trees are better at learning the will wait predicate in the restaurant example

Fig. 11.2.8 Comparing the performance of perceptrons and decision trees

- Perceptron learns majority function easily, DTL is hopeless.
- DTL learns restaurant function easily, perceptron cannot represent it.

(recall : Only linearly separable functions are learnable).

- **How to overcome the limitation of linearly separable functions ?**

Solution : Multi-layer perceptron networks !

- **Multi-layer perceptrons :**
  - Layers are usually fully connected.
  - Numbers of hidden units typically chosen by hand.

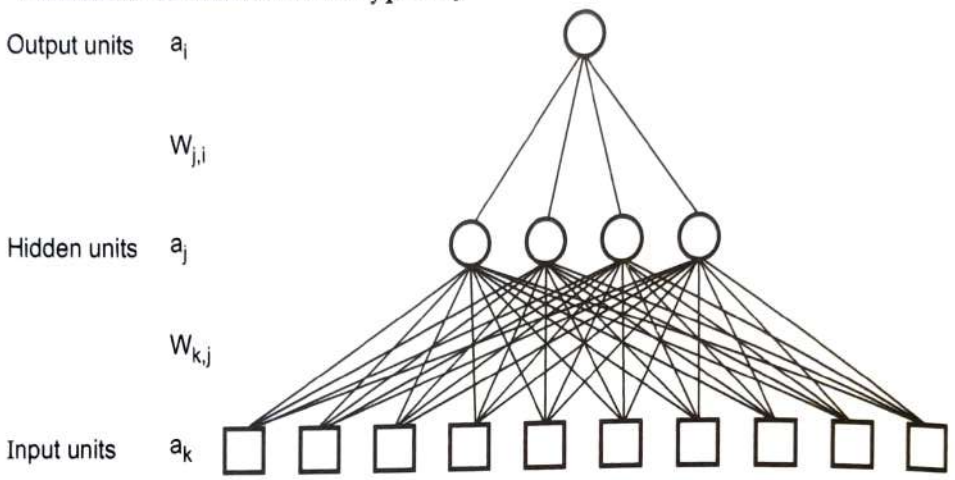
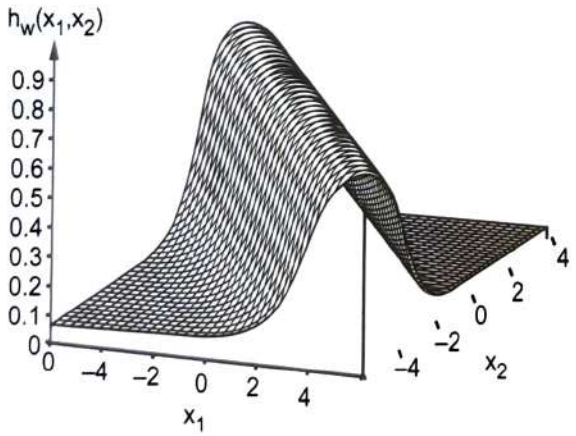


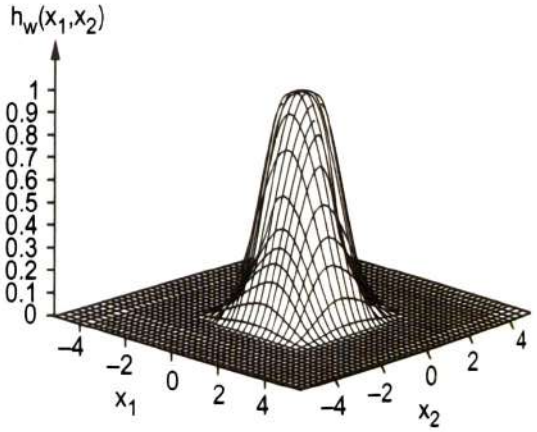
Fig. 11.2.9 Multilayer percetrons network

- Expressiveness of MLPs :

All continuous functions with 2 layers, all functions with 3 layers.



(a) The result of combining two opposite - facing soft threshold functions to produce a ridge.



(b) The result of combining two ridges to produce a pump

Fig. 11.2.10

- Combine two opposite-facing threshold functions to make a ridge.
- Combine two perpendicular ridges to make a bump.
- Add bumps of various sizes and locations to fit any surface.

Problem : Requires exponentially (over the input) many hidden units. In general, Example for all boolean functions : $2^n/n$ units are needed.

- **Back-propagation learning :**

Here output layer is same as for single-layer perceptron.

$$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$$

where, $\Delta_i = Err_i \times g'(in_i)$

Hidden layer : Back-propagate the error from the output layer.

$$\Delta_j = g'(in\, j) \sum_i W_{j,i}\, \Delta_i$$

Update rule for weights in hidden layers.

$$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j$$

(**Remark :** Most neuroscientists deny that back-propagation occurs in the brain).

- Back-propagation learning algorithm :

Function BACK-PROP-LEARNING (examples, network) returns a neural network.

Inputs : examples, a set of examples, each with input vector x and output vector y.

Network, a multilayer network with L layers, weights $W_{j,i}$, activation function g.

Repeat

for each e in examples do

for each node j in the input layer

do $a_j \leftarrow x_j [e]$

for l = 2 to M do

$in_i \leftarrow \sum_j W_{j,i}\, a_j$

$a_i \leftarrow g(in_i)$

for each node i in the output layer do

$\Delta_i \leftarrow g'(in_i) \times (y_i[e] - a_i)$

for l = M – 1 to 1 do

for each node j in layer l do

$\Delta_j \leftarrow g'(in_j) \sum_i W_{j,i}\, \Delta_i$

for each node i in layer l + 1 do

$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$

until some stopping criterion is satisfied

return NEURAL-NET-HYPOTHESIS (network).

- **Back-propagation learning [diagramic representation]**

At each epoch, sum gradient updates for all examples.

Training curve for 100 restaurant examples (with a single hidden-layer network with 4 hidden units) is shown in Fig. 11.2.9. They converge to a perfect fit to the training data.

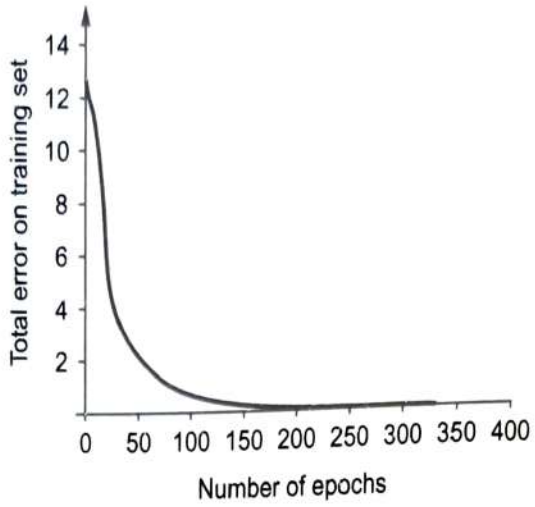


Fig. 11.2.11 (a) Training curve showing gradual reduction in error as weights are modified over several epochs, for a given set of examples in the restaurant domain

Typical problems : Slow convergence, local minima.

Learning curve for MLP with 4 hidden units : -

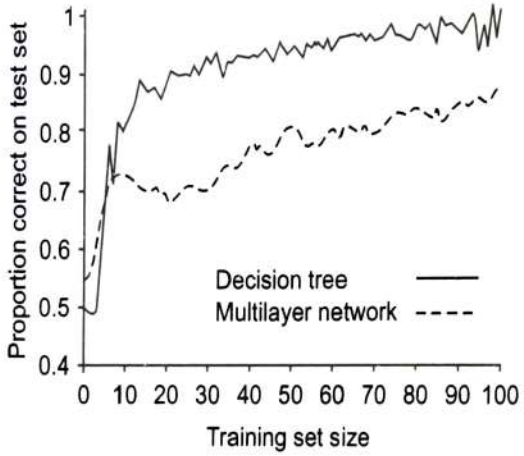


Fig. 11.2.11 (b) Comparative earning curves showing that decision-tree learning does slightly better than back-propagation in a multilayer network

MLPs are quite good for complex pattern recognition tasks, but resulting hypotheses cannot be understood easily.

- **Neural Networks-Summary :**

- Most brains have lots of neurons : Each neuron ≈ linear - threshold unit (?)
- Perceptrons (one-layer networks) is insufficient in expressiveness.
- Multi-layer networks are sufficiently expressive; can be trained by gradient descent, i.e. error back propagation.
- Many applications : Speech recognition, handwriting, fraud detection, economics, etc.
- Engineering, cognitive modelling, and neural system modelling subfields have largely diverged.

11.3 Kernel Machine

It is relatively new family of learning methods that use an efficient training algorithm and can represent complex, non-linear function. Kernel machines are generally called as Support Vector Machines (SVM).

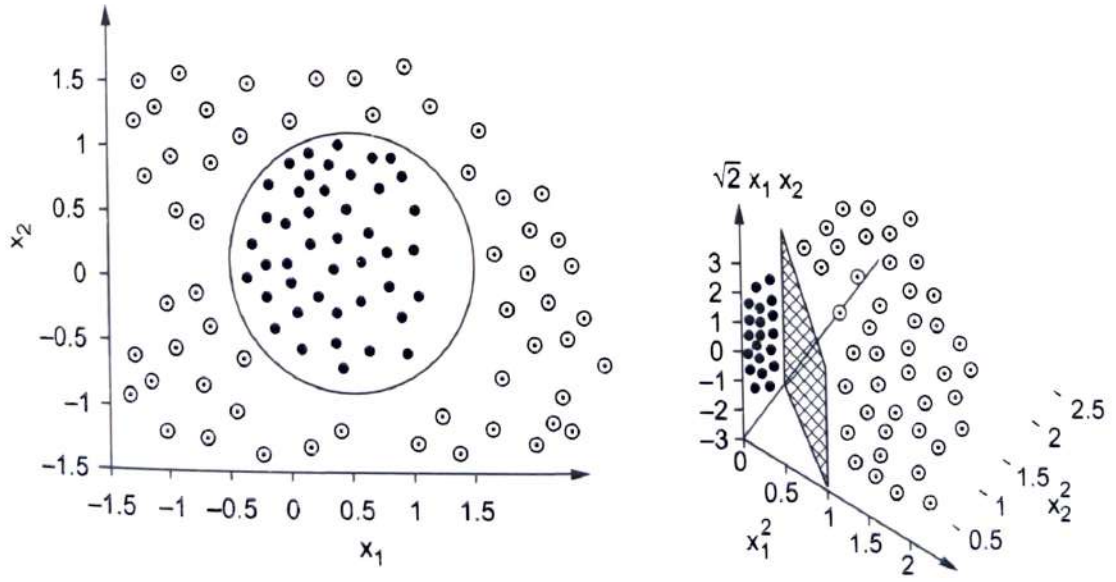
A classifier derived from statistical learning theory by V. Vapnik, et.al.in 1992.

SVM became famous when, using images as input, it gave accuracy comparable to neural-network with hand-designed features in a handwritting recognition task.

Currently, SVM is widely used in object detection and recognition, content-based image retrieval, text recognition, biometrics, speech recognition, etc. They are also used for regression.

- **Characteristics of SVM :**
  - It is supervised learning algorithm (Vapnik 98).
  - It is efficient, it can represent complex, non-linear functions.
  - It is suitable for training data that is not linearly separable (input space), the data is mapped to a higher-dimensional space (feature space).
  - The data are linearly-separable in the new space by maping data into a higher dimension space, they will always be linearly separable (example - N-data points in a space of N–1 dimensions).
- Mapping a bidimensional data to 3-D :

The example :



(a) A two dimensional training with positive examples as black circles and negative examples as white circles. The true decision boundary, $x_1^2 + x_2^2 \leq 1$, is also shown.

(b) The same data after mapping into three dimensional input space x_1^2, x_2^2, $(\sqrt{2}\, x_1, x_2)$. The circular decision boundry in (a) becomes a linear decision boundry in three dimensions.

Fig. 11.3.1

Ideal : We express all inputs through a vector of features :

$$f_1 = x_1^2$$

$$f_2 = x_2^2$$

$$f_3 = \sqrt{2x_1x_2}$$

- In the new 3-dimensional space the data becomes linearly separable !
- The observation in a space with sufficiently many dimensions, makes all data to become linearly separable !

Problem : Danger of overfitting if D ~ N !

- **Optimal linear separator :**
  - If d ≈ N, d - dimension, N - number of examples overfitting.
  - Optimal linear separator : The largest margin between it and the positive examples on one side and the negative examples on the other side.
  - Quadratic programming optimization problem :

Examples x_1 classifications : $y_1 = \pm 1$.

- Maximizing the expression :

$$\sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j, y_i y_j (x_i \cdot x_j)$$

This function has a single global maximum !

The equation of the separator : -

$$h(X) = \text{sign}\left(\sum_i \alpha_i y_i (X \cdot x_i) \right)$$

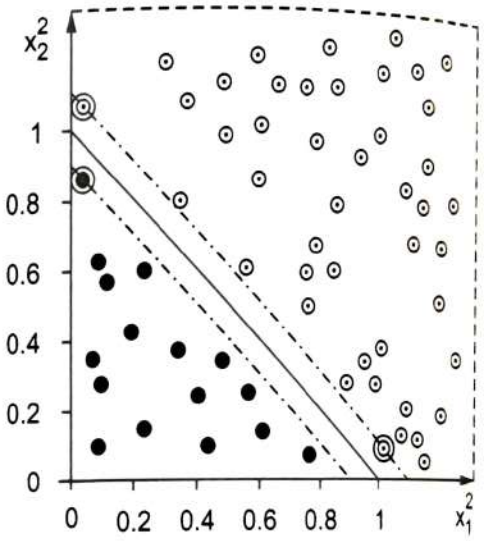


Fig. 11.3.2 A close-up projected onto the first two dimensions, of the optimal seperator shown in Fig. 11.3.1 (b). The separator is shown as a heavy line, with the closest points-the support vectors - marked with the circles. The margin is the seperation between the positive and negative examples

- **Support vectors : -**

Are the points closest to the separator (the only ones for which the weights are not 0).

The optimal number of support vectors is usually much smaller than N!

- Non-linear SVMs :

Datasets that are linearly separable with noise, work out great : -

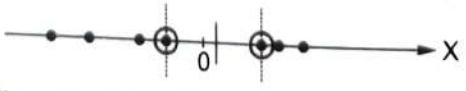


Fig. 11.3.3 (a) Linearly seperable data

But what are we going to do if the dataset is just too hard ?

Fig. 11.3.3 (b) Hard dataset

How about mapping data to a higher-dimensional space as shown in below figure.

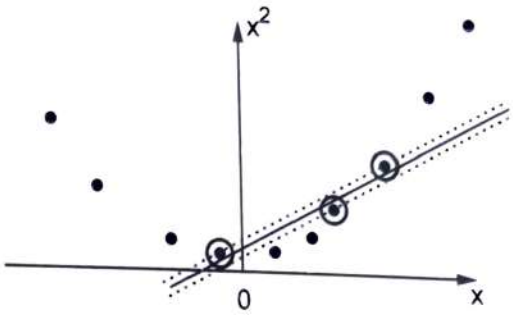


Fig. 11.3.3 (c) Data mapped to higher dimension

- **Non-linear SVMs : Feature space**
  - General idea : The original input space can be mapped to some higher-dimensional feature space where the training set is separable.
- **Non-linear SVM : The Kernel Trick**
  - With this mapping, our discriminant function is now : -

$$g(x) = W^T \phi(x) + b = i \in S V \sum_{i \in sv} \alpha_i \boxed{\phi(x_i)^T \phi(x)} + b$$

  - No need to know this mapping explicitly, because we only use the dot product of feature vectors in both the training and test.
  - A kernel function is defined as a function that corresponds to a dot product of two feature vectors in some expanded feature space.

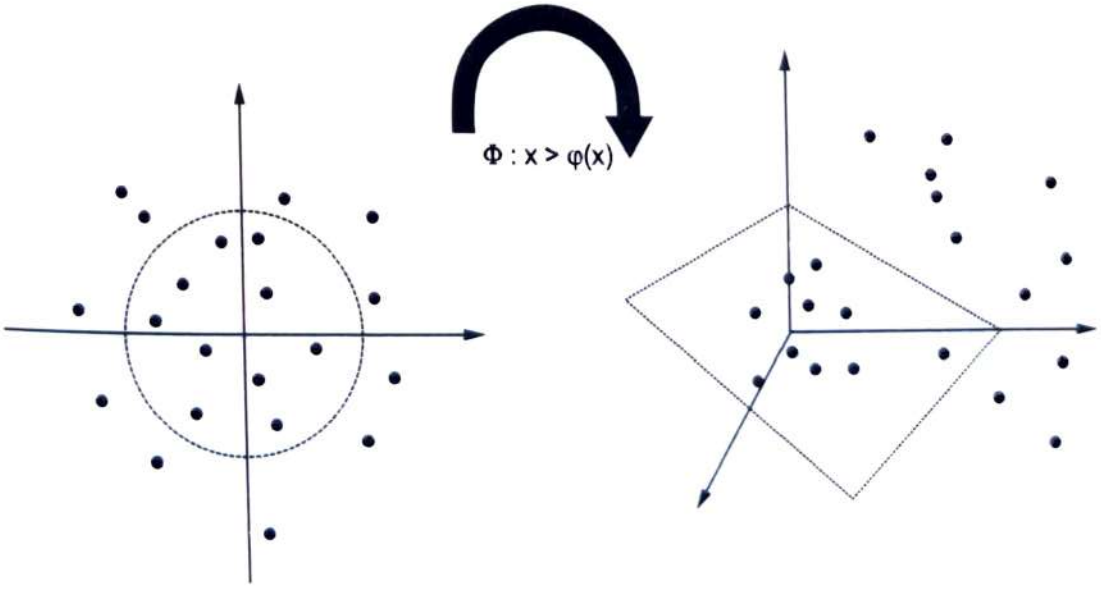


Fig. 11.3.3 (d) Input mapping to higher dimension

$$K(X_i, X_j) \equiv \phi(X_i)^T \phi(X_j)$$

– An example : 2-dimensional vectors $X = [x_1 \ x_2]$;

Let $K(x_i, X_j) = (1 + X_i^T X_j)^2$

Need to show that $K(X_i, X_j) = \phi(X_i)^T \phi(X_j)$:

$$K(X_i, X_j) = (1 + X_i^T X_j)^2$$

$$= 1 + x_{i1}^2 x_{j1}^2 + 2x_{i1} x_{j1} x_{i2} x_{j2} + x_{i2}^2 x_{j2}^2 + 2x_{i1} x_{j1} + 2x_{i2} x_{j2}$$

$$= [1 \ x_{i1}^2 \sqrt{2} x_{i1} x_{i2} \ x_{i2}^2 \sqrt{2} x_{i1} \sqrt{2} x_{i2}]^T \ [1 \ x_{j2} \sqrt{2} x_{j1} x_{j2} \ x_{j2}^2 \sqrt{2} x_{j1} \sqrt{2} x_{j2}]$$

$$= \phi(X_i)^T \phi(X_j), \text{ where } \phi(X) = [1 \ x_1^2 \sqrt{2} x_1 x_2 \ x_2^2 \sqrt{2} x_1 \sqrt{2} x_2]$$

- **Examples of commonly-used kernel functions : -**

i) Linear kernel :

$$K(X_i, X_j) = X_i^T X_j$$

ii) Polynomial kernel :

$$K(X_i, X_j) = (1 + X_i^T X_j)^P$$

iii) Gaussian (Radial-Basis Function (RBF)) kernel :

$$K(X_i, X_j) = \exp\left(- \frac{||X_i - X_j||^2}{2\sigma^2} \right)$$

iv) Sigmoid :

$$K(X_i, X_j) = \tan h (\beta_0 X_i^T X_j + \beta_1)$$

– In general, functions that satisfy Mercer's condition can be kernel functions.

- Support vector machine : Algorithm
  1) Choose a kernel function.
  2) Choose a value for C.
  3) Solve the quadratic programming problem (many software packages are available).
  4) Construct the discriminant function from the support vectors.

- **Some issues of SVM :**

⇒ Choice of kernel :

- Gaussian or polynomial kernel is default.
- If ineffective, more elaborate kernels are needed.
- Domain experts can give assistance in formulating appropriate similarity measures.

⇒ Choice of kernel parameters :

- Example : σ in Gaussian kernel.
- σ is the distance between closest points with different classifications.
- In the absence of reliable criteria, applications rely on the use of a validation set or cross-validation to set such parameters.

⇒ Optimization criterion :

- Hard margin V.S. soft margin
- A lengthy series of experiments in which various parameters are tested.

Answer in Brief

1. *Write a note on statistical learning.* **(Refer section 11.1.1)**
2. *How is learning done with hidden variables ? Explain unsupervised clustering with hidden variables.* **(Refer section 11.1.4)**
3. *Explain EM algorithm.* **(Refer section 11.1.4)**
4. *Explain the concept on learning using decision trees and neural networks approach.* **(Refer section 11.2)**
5. *Distinguish between feed-forward and recurrent neural network strucuture.* **(Refer section 11.2)**
6. *Explain back propagation process with its algorithm.* **(Refer section 11.2)**
7. *How statistical learning method differs from reinforcement learning method ?*

Ans. : Reinforcement learning is concerned with how an agent decides to take actions in an environment so as to maximize some notion of long-term reward. Reinforcement learning algorithms attempt to find a policy that maps states of the world to the actions the agent thought to take in those states.

Statistical learning is inferring a function from supervised training data. The training data consist of a set of training examples. In this learning, each example is a pair consisting of an input object (typically a vector) and a desired output value (also called the supervisory signal). The inferred function should predict the correct output value for any valid input object. This requires the learning algorithm to generalize from the training data to unseen situations in a "reasonable" way.

Reinforcement learning differs from statistical learning because correct input/output pairs are never presented nor sub optimal actions explicitly corrected. In reinforcement

learning there is a focus on on-line performance, which involves finding a balance between exploration (of uncharted territory) and exploitation (of current knowledge).

11.4 University Questions with Answers

Summer - 18

Q.1 *Explain Hopfield Network.* **(Refer section 11.2)** [4]

Q.2 *What is meant by perceptron ? Give one example.* **(Refer section 11.2)** [3]

Winter - 18

Q.3 *List and explain the application of neural network.* **(Refer section 11.2)** [7]

Summer - 19

Q.4 *Define epoch with respect to ANN.* **(Refer section 11.2)** [3]

Q.5 *Discuss perceptron.* **(Refer section 11.2)** [3]

Q.6 *Explain hopfield network.* **(Refer section 11.2)** [4]

Winter - 19

Q.7 *Enlist some applications of neural networks.* **(Refer section 11.2)** [3]

Summer - 20

Q.8 *How do you define artificial neural network ? How does it learn ?* **(Refer section 11.2)** [4]

Q.9 *What do you understand by classification in neural network ? Briefly explain perceptron algorithm and also narrate its limitation.* **(Refer section 11.2)** [7]