Artificial Intelligence

B.Sc. CSIT/BIM

TEJ BAHADUR SHAHI

ARTIFICIAL INTELLIGENCE

Seema Dhami

B.Sc CSIT/BIM

Tej Bahadur Shahi
Lecturer
Central Department of Computer Science
and Information Technology
Tribhuvan University

KEC Publication and Distribution (P.) Ltd.
Kathmandu, Nepal
Phone: 01-4168301, 01-4241777

ARTIFICIAL INTELLIGENCE

Title

ARTIFICIAL INTELLIGENCE

B.Sc CSIT/BIM

ISBN No.

978-9937-724-38-8

Copyright
Authors

Publisher
KEC PUBLICATION AND DISTRIBUTION PVT. LTD.
Kathmandu
01-4168301, 01-4241777

Distributor
KEC PUBLICATION AND DISTRIBUTION PVT. LTD.
Kathmandu
01-4168301, 01-4241777

Edition
First Edition 2076

PriceRs. 450

PREFACE

There are very much good books on artificial intelligence written by eminent writers in the market. However, they don't entirely covers the syllabus of artificial intelligence prescribed by Institute of science and technology, Tribhuvan University for BSc. CSIT course. This is an attempt to fulfil this gap and It is expected that this book might fulfill this requirement and will be helpful for the aspiring student who want to make their carrier on the field of computer science and information technology.

The content provided in this book are divided into six chapters. Chapter one is about the introduction of basic concepts in AI, its history, foundation and applications. Chapter two discuss about the different approaches to design intelligent agents, their environments and configurations. Chapter three explain about one of the important concept in AI- Problem solving by Searching. Chapter four present the details description of knowledge representation techniques and their issues. The machine learning is discussed in chapter five. The last chapter in this book discuss the possible application of artificial intelligence.

The last but not the least, each chapter consist of theoretical exercise as well as practical lab manual which will help the student to implement the algorithms learned in the course into practical framework.

We are grateful to the entire team of KEC Publication for their endless support in designing and publishing the book.

Author

2076

Artificial Intelligence

Course Title: Artificial Intelligence — Full Marks: 60 + 20 + 20
Course No.: CSC 261 — Pass Marks: 24 + 8 + 8
Nature of the Course: Theory + Lab — Credit Hours: 3
Semester: IV

Course Objectives: The course introduces the ideas the techniques underlying the principles and design of artificial intelligent system. The course covers the basics and applications of AI including: design of intelligent agents, problem solving, searching, knowledge representation systems, probabilistic reasoning, neural networks, machine learning and natural language processing.

Course Objectives: The main objective of the course is to introduce fundamental concepts of Artificial Intelligence. The general objectives are to learn about computer systems that exhibit intelligent behaviour, design intelligent agents, identify AI problems and solve the problems, design knowledge representation and expert system, design neural networks for solving problems, identify different machine learning paradigms and identify their practical applications.

Course Contents

Unit I: Introduction **(3Hrs.)**

1.1 Artificial Intelligence (AI), AI Perspectives; acting and thinking humanly,
1.2 History of AI
1.3 Foundations of AI
1.4 Applications ofAI

Unit II: Intelligent Agents **(4 Hrs.)**

2.1 Introduction of agents, structure of Intelligent agent, properties of Intelligent Agents
2.2 Configuration of Agents, PEAS description of Agents
2.3 Types of Agents: Simple Reflexive, Model Based, Goal Based Utility Based.
2.4 Environment Types : Deterministic, Stochastic, Static, Dynamic, Observable, Semi-observable, Single Agent, Multi Agent

Unit III: Problem Solving by Searching **(9 Hrs.)**

3.1 Definition, Problems as a State space search, Problem formulation, Well-defined problems,
3.2 Solving problems by searching, search strategies, Performance evaluation of search techniques
3.3 Uninformed search: Depth First Search, Breadth First search, Depth Limited Search, Iterative Deepening Search, Bidirectional Search.
3.4 Informed Search: Greedy Best first search, A* search, Hill Climbing, Simulated Annealing
3.5 Game playing, Adversarial search techniques, Mini-max Search, Alpha-Beta Pruning.
3.6 Constraints Satisfaction Problems

Unit IV: Knowledge Representation **(14 Hrs.)**

4.1 Definition and importance of Knowledge, Issues in Knowledge Representation, Knowledge Systems, Properties of Knowledge Representation Systems.

4.2 Types of Knowledge Representation Systems: Semantic Nets, Frames, Conceptual Dependencies, Scripts, Rule Based Systems Propositional Logic, Predicate Logic

4.3 Propositional Logic (PL): Syntax, Semantics, Formal logic-connectives, truth tables tautology, validity, well-formed-formula, Inference using Resolution, Backward Chaining and Forward Chaining

4.4 predicate Logic: FOPL, Syntax, Semantics, Quantification, Inference with FOPL, BY converting into PL (Existential and universal instantiation), Unification ald lifting, Inference using resolution

4.5 Handling Uncertain Knowledge, Random Variable, Prior and Posterior probability, Inference using Full Joint Distribution, Baye's Rule and its use, Bayesian Network, Reasoning in Belief networks

4.6 Fuzzy Logic

Unit V: Machine Learning **(9 Hrs.)**

5.1 Introduction to machine Learning, Concepts of Learning, Supervised, Unsupervised and Reinforcement Learning

5.2 Statistical-based Learning: Naive Bayes Model

5.3 Learning by Genetic Algorithm

5.4 Learning with Neural Networks: Introduction, Biological Neural Networks Vs. Artificial neural Networks (ANN), Mathematical Model of ANN, Thypes of ANN: Feed-forwards, Recurrent, Single Layered, Multi-Layered, Application of Artificial Neural Networks, Learning by Training ANN, Supervised vs. Unsupervised Learning, Hebbian Learning, perceptron Learning, Back-propagation Learning

Unit VI: Applications of AI **(6 Hrs.)**

6.1 Expert Systems Development of Expert Systems

6.2 Natural Language Processing: Natural Language Understanding and Natural Language Generation, Steps of Natural Language Processing

6.3 Machine Vision Concepts

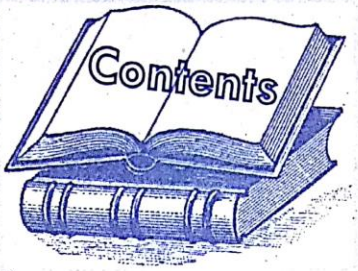
6.4 Robotics

Laboratory Works:

The laboratory work consists of design and implementation of intelligent agents and expert systems, searching techniques, knowledge representation systems and machine learning techniques. Students are also advised to implement Neural networks, Genetic Algorithms for solving practical problems of AI. Students are advised to use LISP, PROLOG, or any other high level language.

Text Books:

1. Stuart Russel and Peter Norvig,. Artificial Intelligence A Modern Approach, Pearson.



1 Unit INTRODUCTION

Introduction to Artificial Intelligence **2**
Background concept 2
Introduction to Artificial Intelligence (AI) 3
Acting Humanly: The Turing Test Approach 5
Thinking Humanly: Cognitive Science Approach 6
Think Rationally: The laws of thought approach 12
Acting Rationally: The rational Agent approach 7
Characteristics of A.I. Programs 8
Foundations of AI **8**
Philosophy 8
Mathematics 9
Psychology 9
Economics 10
Linguistics 10
Neuroscience 10
Control theory 11
Brief Foundation of AI 11
Brief History of AI **12**
The Birth of AI (1956) 12
Great Expectations (1952-1969) 12
Collapse in AI Research (1966 - 1973) 13

AI revival through knowledge-based systems (1969-1970) 13
AI becomes an industry (1980 - present) 13
AI becomes a science (1987 - present) 13
Applications of AI **14**
Game playing 14
Speech recognition 14
Understanding natural language 15
Computer Vision 15
Expert Systems 15
E-Commerce 16
Heuristic Classification 16
❑ Exercise 18

2 Unit INTELLIGENT AGENTS

Introduction **22**
Agent Terminology 22
Rationality 23
Structure of Intelligent Agent 23
Properties of Intelligent Agents 24
PEAS Description of Agents 25
Types of Agents 27
Simple Reflex Agents 28
Model-based Reflex Agents 28
Goal-based Agents 29
Utility-based Agents 30
Environment Types (On the basis of properties) **30**
Complete vs. Incomplete 31
Deterministic vs. Stochastic 31
Static vs. Dynamic 31

Discrete vs. Continuous 32
Competitive Vs. Collaborative 32
Fully Observable vs. Partially Observable 32
Single Agent vs. Multi Agent 32
- **Exercise 34**
- **Lab Manual 35**

3 Unit PROBLEM SOLVING BY SEARCHING

Searching 36
Problem Solving Agent 36
Problem Formulation 39
State Space representation 40
Well-defined Problem 42
Types of Search Algorithms 42
Solving Problems by Searching 43
Search Strategies 43
Uninformed Search 44
Informed or Heuristic search 57
A* Search 64
Hill Climbing Search 70
Simulated Annealing 73
Game Playing 74
Adversarial Search Techniques 74
The Mini-max Search 76
Alpha-Beta Pruning 78
Constraint Satisfaction Problems 82
- **Exercise 87**
- **Lab Manual 89**

4 Unit KNOWLEDGE REPRESENTATION

Definition and Importance of Knowledge **94**
- Issue in Knowledge Representations 94
- Knowledge Representation System and its Properties 95

Types of Knowledge Representation schemes **96**
- Semantic Nets 96
- Components of a Semantic Network 97
- AND/OR tree 97
- Frame based Knowledge Representations 104
- Conceptual Dependencies 106
- Script Structure 113
- Rule Based Knowledge Representation System (RBS) 116
- Logic Based Knowledge Representations 118
- Logical Inference 120
- First-Order Logic 128
- Backward Chaining 136
- Bayes Theorem 138
- probability Distribution 140
- Joint Probability Distribution 141
- Bayesian Network 142
- Inference with Bayesian Network 144

Fuzzy Logic **145**
- Set 145
- Fuzzy Set 147
- Fuzzification 155
- Defuzzification 155

- **Exercise** **158**
- **Lab Manual** **161**

5 Unit MACHINE LEARNING

Introduction **164**
Machine Learning and Artificial Intelligence 164
General Machine Learning agent Framework or architecture 166
Types of Machine Learning 167
Statistical-based Learning: Naive Bayes Model **175**
Model 175
Neural Networks **178**
Components of a Neuron 179
Activation Functions 180
Types of Feed Neural Networks 187
- **Exercise** **199**
- **Lab Manual** **200**

6 Unit APPLICATIONS OF AI

Expert Systems **210**
Block Diagram 210
Stages of Expert System Development 211
Natural Language Processing **215**
Natural Language Generation 215
Natural Language Understanding 217
Levels of knowledge used in Language Understanding 217

Machine Translation **226**
Pre-editing and Post Editing in MT 227
Rule Based Machine Translation (RBMT) 228
Statistical Machine Translation (SMT) 228
Hybrid Machine Translation (HMT) 229
An Introduction to Machine Vision Systems 230
Image Sensor 231
Vision Processor 231
Communication Interface 231
Lens 232
How Machine Vision Systems Work 232
Machine Vision **234**
Computer Vision **236**
Robotics **237**
Arm 238
Drive 238
Controller 238
End-Effector 238
Sensor 238
- **Exercise** **239**
- **Lab manual** **340**
- **Bibliography** **244**

UNIT 1

INTRODUCTION

OUTLINE

- **Introduction to Artificial Intelligence**
  - Background concept
  - Introduction to Artificial Intelligence (AI)
  - Acting Humanly: The Turing Test Approach
  - Thinking Humanly: Cognitive Science Approach
  - Acting Rationally: The rational Agent approach
  - Characteristics of A.I. Programs
- **Foundations of AI**
  - Philosophy
  - Mathematics
  - Psychology
  - Economics
  - Linguistics
  - Neuroscience
  - Control theory
  - Brief Foundation of AI
- **Brief History of AI**
  - The Birth of AI (1956)
  - Great Expectations (1952-1969)
  - Collapse in AI Research (1966 - 1973)
  - AI revival through knowledge-based systems (1969-1970)
  - AI becomes an industry (1980 - present)
  - AI becomes a science (1987 - present)
- **Applications of AI**
  - Game playing
  - Speech recognition
  - Understanding natural language
  - Computer Vision
  - Expert Systems

Introduction to Artificial Intelligence

Background Concept

Artificial Intelligence (AI) is the science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biologically observable. Intelligence is the computational part of the ability to achieve goals in the world. Varying kinds and degrees of intelligence occur in people, many animals and some machines.

The problem is that we cannot yet characterize in general what kinds of computational procedures we want to call intelligent. We understand some of the mechanisms of intelligence and not others. Intelligence involves mechanisms, and AI research has discovered how to make computers carry out some of them and not others. If doing a task requires only mechanisms that are well understood today, computer programs can give very impressive performances on these tasks. Such programs should be considered ``somewhat intelligent".

IQ is based on the rates at which intelligence develops in children. It is the ratio of the age at which a child normally makes a certain score to the child's age. The scale is extended to adults in a suitable way. IQ correlates well with various measures of success or failure in life, but making computers that can score high on IQ tests would be weakly correlated with their usefulness. For example, the ability of a child to repeat back a long sequence of digits correlates well with other intellectual abilities, perhaps because it measures how much information the child can compute with at once. However, ``digit span" is trivial for even extremely limited computers.

After World War II (WWII), a number of people independently started to work on intelligent machines. The English mathematician Alan Turing may have been the first. He gave a lecture on it in 1947. He also may have been the first to decide that AI was best researched by programming computers rather than by building machines. By the late 1950s, there were many researchers on AI, and most of them were basing their work on programming computers.

Introduction to Artificial Intelligence (AI)

Artificial Intelligence is a branch of Science which deals with helping machines to find solutions to complex problems in a more human-like fashion i.e. to create intelligent machine. This generally involves borrowing characteristics from human intelligence, and applying them as algorithms in a computer friendly way. A more or less flexible or efficient approach can be taken depending on the requirements established, which influences how artificial the intelligent behavior appears. Scientists have proposed following two major consensus definitions of intelligence:

1. **From Mainstream Science on Intelligence (1994)**

   A very general mental capability that, among other things, involves the ability to reason, plan, solve problems, think abstractly, comprehend complex ideas, learn quickly and learn from experience. It is not merely book learning, a narrow academic skill, or test-taking smarts. Rather, it reflects a broader and deeper capability for comprehending our surroundings-making sense" of things, or "figuring out" what to do.

2. **From Intelligence: Knowns and Unknowns (1995)**

   Individuals differ from one another in their ability to understand complex ideas, to adapt effectively to the environment, to learn from experience, to engage in various forms of reasoning, [and] to overcome obstacles by taking thought. Although these individual differences can be substantial, they are never entirely consistent: a given person's intellectual performance will vary on different occasions, in different domains, as judged by different criteria. Concepts of "intelligence" are attempts to clarify and organize this complex set of phenomena.

Thus, intelligence is:

- the ability to reason
- the ability to understand
- the ability to create
- the ability to Learn from experience
- the ability to plan and execute complex tasks

What is Artificial Intelligence?

"Giving machines ability to perform tasks normally associated with human intelligence."

AI is intelligence of machines and branch of computer science that aims to create it. AI consists of design of intelligent agents, which is a program that perceives its environment and takes action that maximizes its chance of success. With AI it comes issues like deduction, reasoning, problem solving, knowledge representation, planning, learning, natural language processing, perceptron, etc.

"Artificial Intelligence is the part of computer science concerned with designing intelligence computer systems, that is, systems that exhibit the characteristics we associate with intelligence in human behavior."

Different definitions of AI are given by different books/writers. These definitions can be divided into two by two matrix.

Systems that think like humans	Systems that think rationally
"The exciting new effort to make computers think..... machine with minds, in the full and literal sense." (Haugeland, 1985) "[The automaton of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning....." (Bellman, 1978)	"The study of mental faculties through the use of computational models." (Charniak and McDermott, 1985) "The study of the computations that make it possible to perceive, reason, and act." (Winston, 1992)
Systems that act like humans	**Systems that act rationally**
"The art of creating machines that performs functions that require intelligence when performed by people." (Kurzweil, 1990) "The study of how to make computer do things at which, at the moment, people are better." (Rich and Knight, 1991)	"Computational Intelligence is the study of the design of intelligent agents." (Poole et al., 1998) "AI... is concerned with intelligent behavior in artifacts." (Nilsson, 1998)

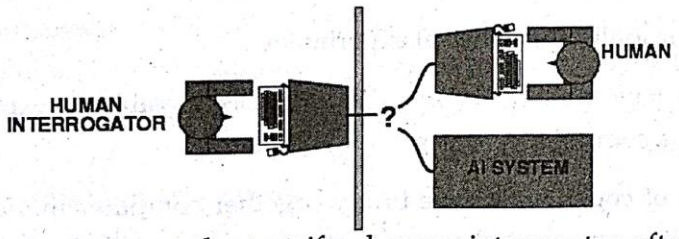
Top dimension is concerned with **thought processes and reasoning,** where as bottom dimension addresses the **behavior.**

The definition on the left measures the success in terms of fidelity of human performance, whereas definitions on the right measure an ideal concept of intelligence, which is called **rationality.**

Human-centered approaches must be an empirical science, involving hypothesis and experimental confirmation. A rationalist approach involves a combination of mathematics and engineering.

Acting Humanly: The Turing Test Approach

The Turing test, proposed by Alan Turing (1950) was designed to convince the people that whether a particular machine can think or not. He suggested a test based on indistinguishability from undeniably intelligent entities- human beings. The test involves an interrogator who interacts with one human and one machine. Within a given time the interrogator has to find out which of the two the human is, and which one the machine.



The computer passes the test if a human interrogator after posing some written questions, cannot tell whether the written response come from human or not.

To pass a Turing test, a computer must have following capabilities:

- **Natural Language Processing:** Must be able to communicate successfully in English
- **Knowledge representation:** To store what it knows and hears.
- Automated reasoning: Answer the Questions based on the stored information.
- **Machine learning:** Must be able to adapt in new circumstances.

Turing test avoid the physical interaction with human interrogator. Physical simulation of human beings is not necessary for testing the intelligence.

The total Turing test includes video signals and manipulation capability so that the interrogator can test the subject's perceptual abilities and object manipulation ability. To pass the total Turing test computer must have following additional capabilities:

- **Computer Vision:** To perceive objects
- **Robotics:** To manipulate objects and move

Thinking Humanly: Cognitive Science Approach

If we are going to say that a given program thinks like a human, we must have some way of determining how humans think. We need to get inside the actual workings of human minds. There are two ways to do this:

- Through introspection: catch our thoughts while they go by
- Through psychological experiments.

Once we have precise theory of mind, it is possible to express the theory as a computer program.

The field of cognitive science brings together computer models from AI and experimental techniques from psychology to try to construct precise and testable theories of the workings of the human mind.

Think Rationally: The Laws of Thought Approach

The Greek philosopher Aristotle was one of the first to attempt to codify ``right thinking," that is, irrefutable reasoning processes. His famous syllogisms provided patterns for argument structures that always gave correct conclusions given correct premises. For example, ``Ram is a man; all men are mortal; therefore Ram is mortal." These laws of thought were supposed to govern the operation of the mind, and initiated the field of logic. He gave Syllogisms that always yielded correct conclusion when correct premises are given.

Problems with this approach

There are two main obstacles to this approach which are listed below:

- First, it is not easy to take informal knowledge and state in the formal terms required by logical notation, particularly when knowledge is not 100% certain.
- Second, solving problem principally is different from doing it in practice. Even problems with certain dozens of fact may exhaust the computational resources of any computer unless it has some guidance as which reasoning step to try first.

Acting Rationally: The Rational Agent Approach

Acting rationally means acting so as to achieve one's goals, given one's beliefs. An agent is just something that perceives and acts. (This may be an unusual use of the word, but you will get used to it.) In this approach, AI is viewed as the study and construction of rational agents.

In the ``laws of thought'' approach to AI, the whole emphasis was on correct inferences. Making correct inferences is sometimes part of being a rational agent, because one way to act rationally is to reason logically to the conclusion that a given action will achieve one's goals, and then to act on that conclusion. On the other hand, correct inference is not all of rationality; because there are often situations where there is no provably correct thing to do, yet something must still be done. There are also ways of acting rationally that cannot be reasonably said to involve inference. For example, pulling one's hand off of a hot stove is a reflex action that is more successful than a slower action taken after careful deliberation. Agent is something that acts. Computer agent is expected to have following attributes:

- Autonomous control
- Perceiving their environment
- Persisting over a prolonged period of time
- Adapting to change
- And capable of taking on another's goal

Rational behavior: doing the right thing.

The right thing: that which is expected to maximizes goal achievement, given the available information.

Rational Agent is one that acts so as to achieve the best outcome or, when there is uncertainty, the best expected outcome.

Advantages

- It is more general than laws of thought approach, because correct inference is just one of several mechanisms for achieving rationality.
- It is more amenable to scientific development than are approaches based on human behavior or human thought because the standard of rationality is clearly defined and completely general.

Characteristics of A.I. Programs

- **Symbolic Reasoning:** reasoning about objects represented by symbols, and their properties and relationships, not just numerical calculations.
- **Knowledge:** General principles are stored in the program and used for reasoning about novel situations.
- **Search:** a ``weak method" for finding a solution to a problem when no direct method exists. Problem: combinatoric explosion of possibilities.
- **Flexible Control:** Direction of processing can be changed by changing facts in the environment.

Foundations of AI

Philosophy

The study of the fundamental nature of knowledge, reality, and existence, especially when considered as an academic discipline is called philosophy. Artificial intelligence has close connections with philosophy because both share several concepts and these include intelligence, action, consciousness, epistemology, and even free will. Furthermore, the technology is concerned with the creation of

artificial animals or artificial people (or, at least, artificial creatures) so the discipline is of considerable interest to philosophers. These factors contributed to the emergence of the philosophy of artificial intelligence. Some scholars argue that the AI community's dismissal of philosophy is detrimental. The philosophy of artificial intelligence attempts to answer such questions as follows:

- Can a machine act intelligently? Can it solve any problem that a person would solve by thinking?
- Are human intelligence and machine intelligence the same? Is the human brain essentially a computer?
- Can a machine have a mind, mental states, and consciousness in the same way that a human being can? Can it feel how things are?

Mathematics

Mathematics is the science that deals with the logic of shape, quantity and arrangement. Math is all around us, in everything we do. It is the building block for everything in our daily lives, including mobile devices, architecture (ancient and modern), art, money, engineering, and even sports.

From math to meaning: Artificial intelligence blends algorithms and applications. Through advances in artificial intelligence and machine learning, Princeton researchers are building technologies in health care, transportation, energy and many other areas. Artificial intelligence is already a part of everyday life. The mathematics of artificial intelligence attempts to answer such questions as follows:

- What are the formal rules to draw the valid conclusions?
- What can be computed?
- How do we reason with uncertain information?

Psychology

The scientific study of the human mind and its functions, especially those affecting behavior in a given context is called psychology. The

psychology in artificial intelligence attempts to answer such questions as follows:

- How humans and animals think and act?

Economics

The branch of knowledge concerned with the production, consumption, and transfer of wealth is called economics. The economics in artificial intelligence attempts to answer such questions as follows:

- How should we make decisions so as to maximize payoff?
- How should we do this when others may not go along?
- How should we do this when the payoff may be far in future?

Linguistics

The scientific study of language and its structure, including the study of grammar, syntax, and phonetics is called linguistics. Specific branches of linguistics include sociolinguistics, dialectology, psycholinguistics, computational linguistics, comparative linguistics, and structural linguistics. It's a confluence of artificial intelligence and computational linguistics which handles interactions between machines and natural languages of humans in which computers are entailed to analyze, understand, alter, or generate natural language. The linguistics in artificial intelligence attempts to answer such questions as follows:

- How does language relate to thought?

Neuroscience

Neuroscience is the scientific study of the nervous system. It is a multidisciplinary branch of biology that combines physiology, anatomy, molecular biology, developmental biology, cytology, mathematical modeling and psychology to understand the fundamental and emergent properties of neurons and neural circuits.

AI researchers trying to imitate its remarkable functionality will benefit by learning more about neuroscience, and the differences between Natural and Artificial Intelligence. A new approach that

bridges AI and neuroscience is described, Embodied Cultured Networks. The Neuroscience in artificial intelligence attempts to answer such questions as follows:

- How do brains process information?

Control theory

Control theory in control systems engineering is a subfield of mathematics that deals with the control of continuously operating dynamical systems in engineered processes and machines.

In an artificial intelligence (AI) planning system the planner generates a sequence of actions to solve a problem. A mathematical theory of AI planning systems that operate in uncertain, dynamic, and time-critical environments is not nearly as well developed as the mathematical theory of systems and control. The Control theory in artificial intelligence attempts to answer such questions as follows:

- How can artifacts operate under their own control?

Brief Foundation of AI

Philosophy	Knowledge representation, logic, foundation of AI.
Mathematics	Search, analysis of search algorithms, logic
Economics	Expert systems, decision theory, principles of rational behavior
Psychology	Behavioristic insights into AI programs
Neutoscience (Brain Science)	Learning, neural networks
Control theory and cybernetics	Information theory and AI, Entropy, Robotics
Computer science and Engineering	Systems for AI

Brief History of AI

- **1943:** Warren Mc Culloch and Walter Pitts: a model of artificial boolean neurons to perform computations.
  - ✓ First steps toward connectionist computation and learning (Hebbian learning).
  - ✓ Marvin Minsky and Dann Edmonds (1951) constructed the first neural network computer
- **1950:** Alan Turing's "Computing Machinery and Intelligence"
  - ✓ First complete vision of AI

The Birth of AI (1956)

- Dartmouth Workshop bringing together top minds on automata theory, neural nets and the study of intelligence.
  - ✓ Allen Newell and Herbert Simon: The logic theorist (first nonnumeric thinking program used for theorem proving)
  - ✓ For the next 20 years the field was dominated by these participants.

Great Expectations (1952-1969)

- Newell and Simon introduced the General Problem Solver.
  - ✓ Imitation of human problem-solving
- Arthur Samuel (1952-) investigated game playing (checkers) with great success.
- John McCarthy(1958-)
  - ✓ Inventor of Lisp (second-oldest high-level language)
  - ✓ Logic oriented, Advice Taker (separation between knowledge and reasoning)
- Marvin Minsky (1958 -)
  - ✓ Introduction of microworlds that appear to require intelligence to solve: e.g. blocks-world.
  - ✓ Anti-logic orientation, society of the mind.

Collapse in AI Research (1966 - 1973)

- Progress was slower than expected.
  - ✓ Unrealistic predictions.
- Some systems lacked scalability.
  - ✓ Combinatorial explosion in search.
- Fundamental limitations on techniques and representations.
  - ✓ Minsky and Papert (1969) Perceptrons

AI Revival Through Knowledge-based Systems (1969-1970)

- General-purpose vs. domain specific
  - ✓ E.g. the DENDRAL project (Buchanan et.al. 1969)
  - ✓ First successful knowledge intensive system.
- Expert systems
  - ✓ MYCIN to diagnose blood infections (Feigenbaum et al.)

    Introduction of uncertainty in reasoning
- Increase in knowledge representation research.

  Logic, frames, semantic nets ...

AI becomes an Industry (1980 - present)

- R1 at DEC (McDermott, 1982)
- Fifth generation project in Japan (1981)
- American response ...

  Puts an end to the AI winter

Connectionist revival (1986 - present): (Return of Neural Network)

Parallel distributed processing (RumelHart and McClelland, 1986); backprop

AI Becomes a Science (1987 - present)

- In speech recognition: hidden markov models

- In neural networks
- In uncertain reasoning and expert systems: Bayesian network formalism

The emergence of intelligent agents (1995 - present)

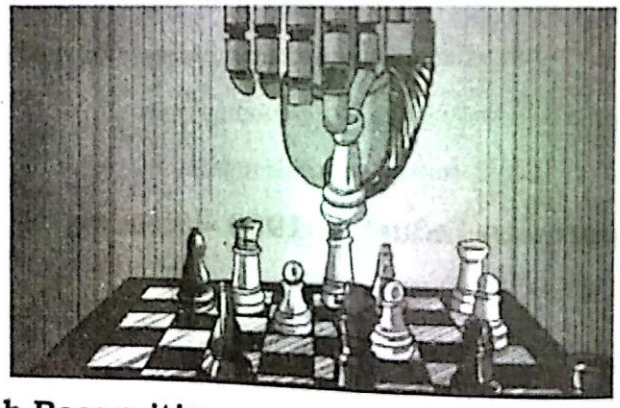
- The whole agent problem:

  "How does an agent act/behave embedded in real environments with continuous sensory inputs"

Applications of AI

Game Playing

We can buy machines that can play master level chess for a few hundred dollars. There is some AI in them, but they play well against people mainly through brute force computation looking at hundreds of thousands of positions. To beat a world champion by brute force and known reliable heuristics requires being able to look at 200 million positions per second.

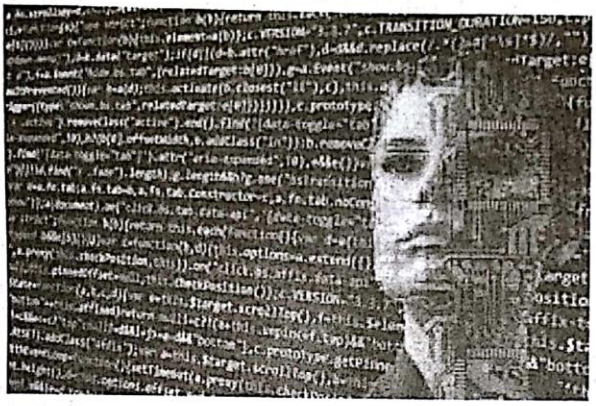


Speech Recognition

In the 1990s, computer speech recognition reached a practical level for limited purposes. Thus United Airlines has replaced its keyboard tree for flight information by a system using speech recognition of flight numbers and city names. It is quite convenient. On the other hand, while it is possible to instruct some computers using speech, most users have gone back to the keyboard and the mouse as still more convenient.

Understanding Natural Language

Just getting a sequence of words into a computer is not enough. Parsing sentences is not enough either. The computer has to be provided with an understanding of the domain the text is about, and this is presently possible only for very limited domains.



Computer Vision

The world is composed of three-dimensional objects, but the inputs to the human eye and computers' TV cameras are two dimensional. Some useful programs can work solely in two dimensions, but full computer vision requires partial three-dimensional information that is not just a set of two-dimensional views. At present there are only limited ways of representing three-dimensional information directly, and they are not as good as what humans evidently use.

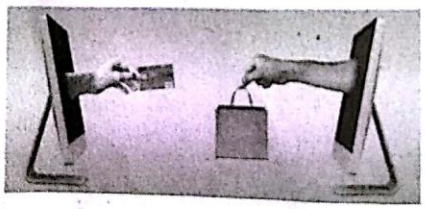
Expert Systems

A ``knowledge engineer'' interviews experts in a certain domain and tries to embody their knowledge in a computer program for carrying out some task. How well this works depends on whether the intellectual mechanisms required for the task are within the present state of AI. When this turned out not to be so, there were many disappointing results. One of the first expert systems was MYCIN in 1974, which diagnosed bacterial infections of the blood and suggested treatments. It did better than medical students or practicing doctors, provided its limitations were observed. Namely, its ontology included bacteria, symptoms, and treatments and did not include patients,

doctors, hospitals, death, recovery, and events occurring in time. Its interactions depended on a single patient being considered. Since the experts consulted by the knowledge engineers knew about patients, doctors, death, recovery, etc., it is clear that the knowledge engineers forced what the experts told them into a predetermined framework. In the present state of AI, this has to be true. The usefulness of current expert systems depends on their users having common sense.

E-Commerce

E-commerce business is a bustling segment of the retail industry, of late. Major e-commerce companies have been using the advanced technologies like AI or machine learning. The impact of the implementation is clearly visible in e-commerce space. Companies are using the AI as chat bots, AI assistants, smart logistics, using algorithms to predict and analyze customers' behaviors. The machine learning based applications are being implemented to augment the marketing strategy. Automation emerged as the key to effective warehouse operations and the technology helped many companies in achieving delivery efficiency. The ultimate aim is to reduce shipping costs. Machine learning helps companies in demand forecasting, product search ranking, product and deals recommendations, merchandising placements, fraud detection, translations and much more.

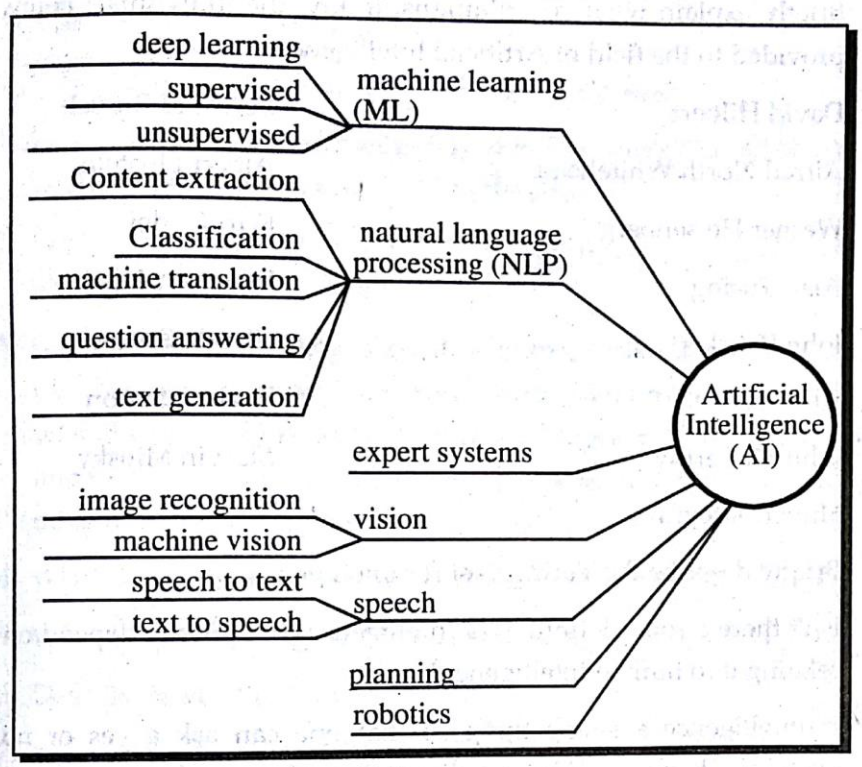


Heuristic Classification

One of the most feasible kinds of expert system given the present knowledge of AI is to put some information in one of a fixed set of categories using several sources of information. An example is advising whether to accept a proposed credit card purchase. Information is available about the owner of the credit card, his record of payment and also about the item he is buying and about the

establishment from which he is buying it (e.g., about whether there have been previous credit card frauds at this establishment).

The application of AI can be shown as follows:



Difference between AI and Omniscience

In Latin, Omnis means "all" and Science means "knowing". Omniscience is the capacity to know everything. Theologians believes to God as being omniscient. A rational agent takes its action according to its percept sequence and instruction from agent function (which may have boundary). Thus, it 'thinks' that it is taking the correct decision

An omniscient agent takes its decision based on an infinite knowledge base. There is noting that the agent doesn't know. Thus it 'knows' that it is taking the correct decision. For example: A rational agent may try to cross a street after perceiving that the road is empty. But while he is in the middle of the road, a metal box can fall from a plane flying overhead. He won't have any idea about this. The omniscient agent would have perceived that too.

Exercis

1. Briefly explain what contributions, if any, the individuals below provided to the field of Artificial Intelligence.

David Hilbert	Bertrand Russell
Alfred North Whitehead	Albert Einstein
Werner Heisenberg	Kurt Gödel
Alan Turing	John W. Mauchly
John P. Eckert	Claude Shannon
Allen Newell	Herbert Simon
John McCarthy	Marvin Minsky
Hubert Dreyfus	

2. Briefly describe the Turing Test for intelligence.
3. Isn't there a solid definition of intelligence that doesn't depend on relating it to human intelligence?
4. Is intelligence a single thing so that one can ask a yes or no question ``Is this machine intelligent or not?
5. Isn't AI about simulating human intelligence?
6. What does the Turing test say about the nature of intelligence?
7. Can you formulate a better test for intelligence?
8. It is not possible to enter into a moral contract with non-human animals; therefore society cannot assign legal rights to non-human animals. The legal world has recognized this issue and many discussions have been held how to protect not only non-human animals, but also land and associated vegetation, etc. Should the scientific community be concerned with animating human intelligence through AI? Do we have moral obligations towards machines?

9. Allen Newell and Herbert A. Simon proposed that intelligent systems require at a minimum a physical symbol system and heuristic search capabilities to perform intelligent actions. What is a heuristic search algorithm?
10. What about IQ? Do computer programs have IQs?
11. Who attended the Dartmouth Summer Conference in 1956? What was the intent (goal) behind the conference?
12. What is the difference, if any, between Computational Intelligence (CI) and Artificial Intelligence (AI).
13. Bill Joy states that a goal should be to force scientists to recognize the heightened risk associated with modern research. The scientific community could require a Hippocratic Oath and a self-imposed system of ethics to regulate research and technology design. State your opinion.
14. What are the applications of AI? Describe with suitable example.
15. What do you mean by AI? Describe different aspects of AI.
16. Describe brief history of AI.
17. Why we need AI? Describe their applications.
18. Describe foundations of AI with suitable examples.
19. Give an explanation on the difference between strong AI and weak AI?
20. Which is not commonly used programming language for AI?
21. What is an artificial intelligence Neural Networks?
22. What about other comparisons between human and computer intelligence?
23. What is agent in artificial intelligence?
24. When did AI research start?
25. Put your comment on Turing's criteria for computer being intelligent.

26. Discuss about potentially negative effects of artificial intelligence on the society i.e. social effect of AI.
27. Define an artificial intelligence (AI). Explain the behaviors of the AI. What do you mean by Turing Test? Explain it.
28. What is Artificial Intelligence (AI)? Describe your own criteria for computer program to be considered intelligent.
29. Give two examples of real world systems that you consider to be examples of Artificial Intelligence. Briefly describe the systems and justify why you believe they fall in this category.
30. Explain about the foundation and historical development of AI.
31. Explain about the recent trend in AI. List out the different activities (from intelligent prospective) that machine can perform completely, those activities machine is partially successful and those activities machine is unable to perform.

✡✡✡

UNIT 2

Intelligent Agents

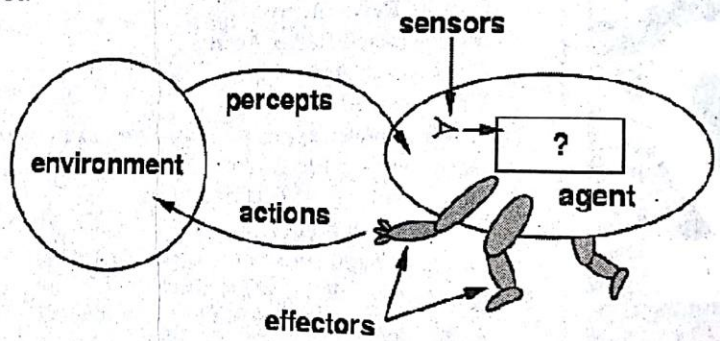
OUTLINE

- ✓ **Introduction**
  - Agent Terminology
  - Rationality
  - Structure of Intelligent Agent
  - Properties of Intelligent Agents
  - PEAS Description of Agents
- ✓ **Types of Agents**
  - Simple Reflex Agents
  - Model-based Reflex Agents
  - Goal-based Agents
  - Utility-based Agents
- ✓ **Environment Types (On the basis of properties)**
  - Complete vs. Incomplete
  - Deterministic vs. Stochastic
  - Static vs. Dynamic
  - Discrete vs. Continuous
  - Competitive Vs. Collaborative
  - Fully Observable vs. Partially Observable

Introduction

In artificial intelligence, an intelligent agent (IA) is an autonomous entity which acts, directing its activity towards achieving goals (i.e. it is an agent), upon an environment using observation through sensors and consequent actuators (i.e. it is intelligent).An Intelligent Agent perceives its environment via sensors and acts rationally upon that environment with its effectors (actuators). Hence, an agent gets percepts one at a time, and maps this percept sequence to actions.

An intelligent agent is a program that can make decisions or perform a service based on its environment, user input and experiences. These programs can be used to autonomously gather information on a regular, programmed schedule or when prompted by the user in real time. Intelligent agents may also be referred to as a bot, which is short for robot.



Agent Terminology

- Performance Measure of Agent – It is the criteria, which determines how successful an agent is.
- Behavior of Agent – It is the action that agent performs after any given sequence of percepts.
- Percept – It is agent's perceptual inputs at a given instance.
- Percept Sequence – It is the history of all that an agent has perceived till date.
- Agent Function – It is a map from the precept sequence to an action.

Rationality

Rationality is nothing but status of being reasonable, sensible, and having good sense of judgment. Rationality is concerned with expected actions and results depending upon what the agent has perceived. Performing actions with the aim of obtaining useful information is an important part of rationality.

What is Ideal Rational Agent?

An ideal rational agent is the one, which is capable of doing expected actions to maximize its performance measure, on the basis of:

- Its percept sequence
- Its built-in knowledge base

Rationality of an agent depends on the following:

- The performance measures, which determine the degree of success.
- Agent's Percept Sequence till now.
- The agent's prior knowledge about the environment.
- The actions that the agent can carry out.

A rational agent always performs right action, where the right action means the action that causes the agent to be most successful in the given percept sequence. The problem the agent solves is characterized by Performance Measure, Environment, Actuators, and Sensors (PEAS).

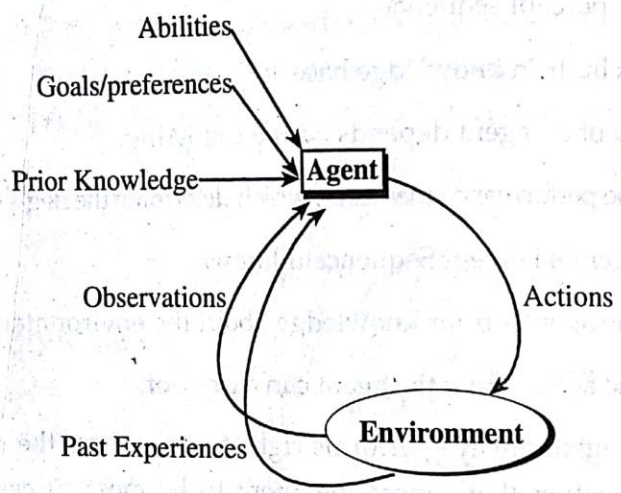
Structure of Intelligent Agent

To understand the structure of Intelligent Agents, we should be familiar with Architecture and Agent Program. Architecture is the machinery that the agent executes on. It is a device with sensors and actuators, for example: a robotic car, a camera, a PC. Agent program is an implementation of an agent function. An agent function is a map from the percept sequence (history of all that an agent has perceived till date) to an action.

Agent = Architecture + Agent Program

Examples of Agent

- **A software agent** has Keystrokes, file contents, received network packages which act as sensors and displays on the screen, files, sent network packets acting as actuators.
- **A Human agent** has eyes, ears, and other organs which act as sensors and hands, legs, mouth, and other body parts acting as actuators.
- **A Robotic agent** has Cameras and infrared range finders which act as sensors and various motors acting as actuators.



Properties of Intelligent Agents

The properties of intelligent agents can be classified into internal and external.

Internal characteristics

Internal characteristics of intelligent agents are listed below,

- **Learning/reasoning**

  An agent has the ability to learn from previous experience and to successively adapt its own behavior to the environment.

- **Reactivity**

  An agent must be capable of reacting appropriately to influences or information from its environment.

- **Autonomy**

  An agent must have both control over its actions and internal states. The degree of the agent's autonomy can be specified. There may need intervention from the user only for important decisions.

- **Goal-oriented**

  An agent has well-defined goals and gradually influence its environment and so achieve its own goals.

External Characteristics

- **Communication**

  An agent often requires an interaction with its environment to fulfill its tasks, such as human, other agents, and arbitrary information sources.

- **Cooperation**

  Cooperation of several agents permits faster and better solutions for complex tasks that exceed the capabilities of a single agent.

- **Mobility**

  An agent may navigate within electronic communication networks.

- **Character**

  Like human, an agent may demonstrate an external behavior with many human characters as possible.

PEAS Description of Agents

PEAS: Performance measure, Environment, Actuators, Sensors.

For Humans

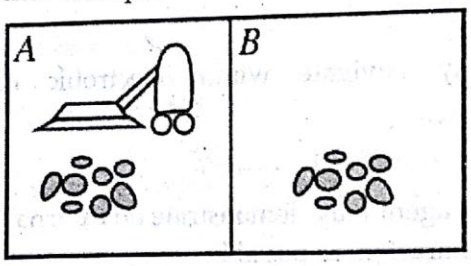
- **Sensors:** Eyes (vision), ears (hearing), skin (touch), tongue (gestation), nose (olfaction), neuromuscular system (proprioception)
- **Percepts:**
  - ✔ At the lowest level – electrical signals from these sensors
  - ✔ After preprocessing – objects in the visual field (location, textures, colors, ...), auditory streams (pitch, loudness, direction), ...

- **Effectors:** limbs, digits, eyes, tongue…
- **Actions:** lift a finger, turn left, walk, run, carry an object …

To set the PEAS, you must first specify the settings for the smart agent design. Consider, for example, the task of designing an automated taxi driver (SELF DRIVING CAR) and setting the Performance Measurement Environment Actuators Sensors. We must first specify the settings for the smart agent design:

- **Measure of performance:** safe, fast, legal, comfortable travel, maximize profits
- **Environment:** Roads, other traffic, pedestrians, customers
- **Actuators:** Steering wheel, accelerator, brake, signal, horn
- **Sensors:** Cameras, sonar, speedometer, GPS, odometer, motor sensors, keyboard, etc …

Let's take another example: **The vacuum-cleaner world**



Environment: square A and B

Percepts: [location and content] E.g. [A, Dirty]

Actions: left, right, suck, and no-op

Percept sequence	Action
[A, Clean]	Right
[A, Dirty]	Suck
[B, Clean]	Left
[B, Dirty]	Suck
…….	……

Example: **Agent = taxi driver**

- **Performance measure:** Safe, fast, legal, comfortable trip, maximize profits
- **Environment:** Roads, other traffic, pedestrians, customers
- **Actuators:** Steering wheel, accelerator, brake, signal, horn
- **Sensors:** Cameras, sonar, speedometer, GPS, odometer, engine sensors, keyboard

Example: **Agent = Medical diagnosis system**

- **Performance measure:** Healthy patient, minimize costs, lawsuits
- **Environment:** Patient, hospital, staff
- **Actuators:** Screen display (questions, tests, diagnoses, treatments, referrals)
- **Sensors:** Keyboard (entry of symptoms, findings, patient's answers)

Example: **Agent = Part-picking robot**

- **Performance measure:** Percentage of parts incorrect bins
- **Environment:** Conveyor belt with parts, bins
- **Actuators:** Jointed arm and hand
- **Sensors:** Camera, joint angle sensors

Types of Agents

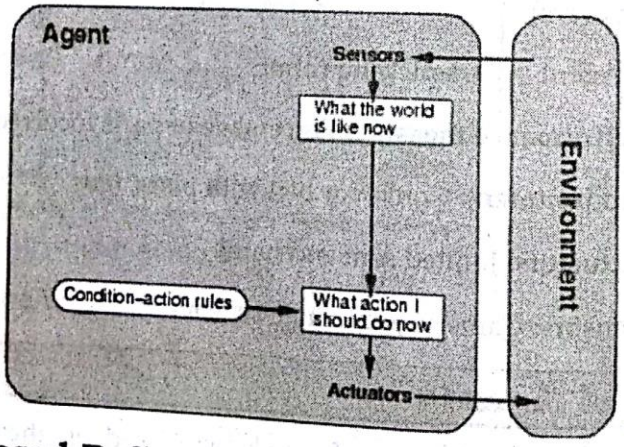
Agents can be grouped into four classes based on their degree of perceived intelligence and capability:

- Simple Reflexive agents
- Model Based agents
- Goal Based agents
- Utility Based agents

Simple Reflex Agents

Simple reflex agents ignore the rest of the percept history and act only on the basis of the current percept. Percept history is the history of all that an agent has perceived till date. The agent function is based on the condition-action rule. A condition-action rule is a rule that maps a state i.e., condition to an action. If the condition is true, then the action is taken, else not. This agent function only succeeds when the environment is fully observable. For simple reflex agents operating in partially observable environments, infinite loops are often unavoidable. It may be possible to escape from infinite loops if the agent can randomize its actions. Problems with Simple reflex agents are:

- Very limited intelligence.
- No knowledge of non-perceptual parts of state.
- Usually too big to generate and store.
- If there occurs any change in the environment, then the collection of rules need to be updated.

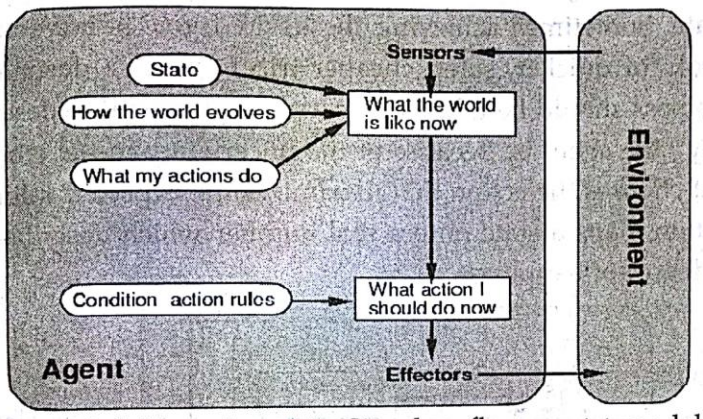


Model-based Reflex Agents

It works by finding a rule whose condition matches the current situation. A model-based agent can handle partially observable environments by use of model about the world. The agent has to keep track of internal state which is adjusted by each percept and that depends on the percept history. The current state is stored inside the agent which maintains some kind of structure describing the part of

the world which cannot be seen. Updating the state requires the information about:

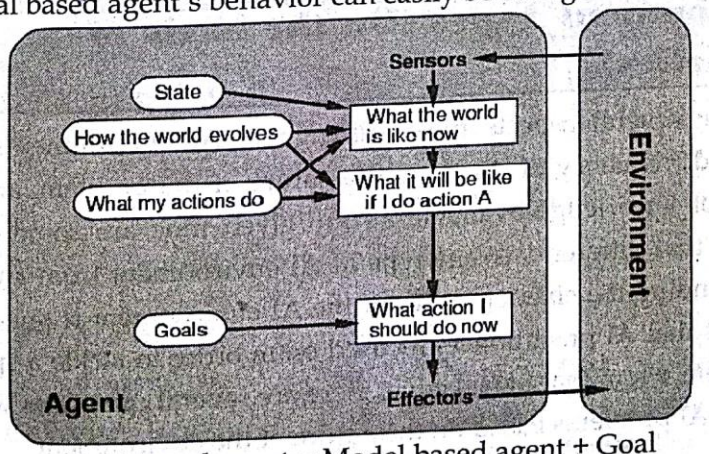
- How the world evolves in-dependently from the agent, and
- How the agent actions affects the world.



Model based agent = Simple reflex agent + model

Goal-based Agents

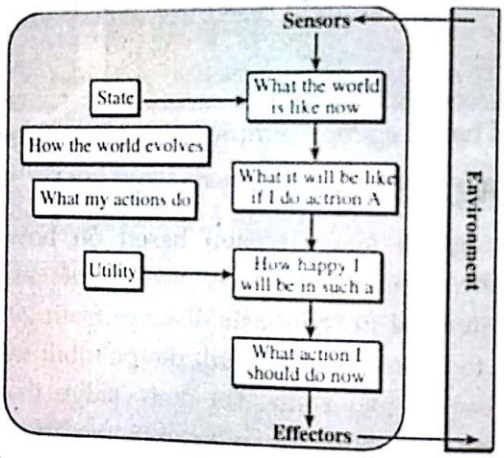
These kinds of agents take decision based on how far they are currently from their goal (description of desirable situations). Their every action is intended to reduce its distance from goal. This allows the agent a way to choose among multiple possibilities, selecting the one which reaches a goal state. The knowledge that supports its decisions is represented explicitly and can be modified, which makes these agents more flexible. They usually require search and planning. The goal based agent's behavior can easily be changed.



Goal based agent = Model based agent + Goal

Utility-based Agents

The agents which are developed having their end uses as building blocks are called utility based agents. When there are multiple possible alternatives, then to decide which one is best, utility based agents are used. They choose actions based on a preference (utility) for each state. Sometimes achieving the desired goal is not enough. We may look for quicker, safer, cheaper trip to reach a destination. Agent happiness should be taken into consideration. Utility describes how "happy" the agent is. Because of the uncertainty in the world, a utility agent chooses the action that maximizes the expected utility. A utility function maps a state onto a real number which describes the associated degree of happiness.



Utility based agent = Goal based agent + utility

Environment Types (On the basis of properties)

There are several aspects those distinguish AI environments. The shape and frequency of the data, the nature of the problem, and the volume of knowledge available at any given time are some of the elements that differentiate one type of AI environment from another. Understanding the characteristics of the AI environment is one of the first tasks that AI practitioners focused on in order to tackle a specific AI problem. From that perspective, there are several categories we use to group AI problems based on the nature of the environment.

- Complete vs. Incomplete
- Deterministic vs. Stochastic
- Static vs. Dynamic
- Discrete vs. Continuous
- Competitive vs. Collaborative
- Fully Observable vs. Partially Observable

Complete vs. Incomplete

Complete AI environments are those on which, at any given time, we have enough information to complete a branch of the problem. Chess is a classic example of a complete AI environment. Poker, on the other hand, is an incomplete environments as AI strategies can't anticipate many moves in advance and, instead, they focus on finding a good 'equilibrium" at any given time.

Deterministic vs. Stochastic

Deterministic AI environments are those on which the outcome can be determined based on a specific state. In other words, deterministic environments ignore uncertainty. Most real world AI environments are not deterministic. Instead, they can be classified as stochastic. Self-driving vehicles are a classic example of stochastic AI processes.

An environment is deterministic if the next state of the environment is solely determined by the current state of the environment and the actions selected by the agents. An inaccessible environment might appear to be non-deterministic since the agent has no way of sensing part of the environment and the result of its actions on it. We have to take into consideration the point of view of the agent when determining whether an environment is deterministic or not since the agent might have limited perception capabilities.

Static vs. Dynamic

Static AI environments rely on data-knowledge sources that don't change frequently over time. Speech analysis is a problem that operates on static AI environments. Contrasting with that model,

dynamic AI environments such as the vision AI systems in drones deal with data sources that change quite frequently.

An environment is dynamic if it changes while an agent is in the process of responding to a percept sequence. It is static if it does not change while the agent is deciding on an action i.e. the agent does not to keep in touch with time. An environment is semi dynamic if it does not change with time but he agent's performance score does.

Discrete vs. Continuous

Discrete AI environments are those on which a finite [although arbitrarily large] set of possibilities can drive the final outcome of the task. Chess is also classified as a discrete AI problem. Continuous AI environments rely on unknown and rapidly changing data sources. Vision systems in drones or self-driving cars operate on continuous AI environments.{p

Competitive Vs. Collaborative

Competitive AI environments face AI agents against each other in order to optimize a specific outcome. Games such as GO or Chess are examples of competitive AI environments. Collaborative AI environments rely on the cooperation between multiple AI agents. Self-driving vehicles or cooperating to avoid collisions or smart home sensors interactions are examples of collaborative AI environments.

Fully Observable vs. Partially Observable

A fully observable AI environment has access to all required information to complete target task. Image recognition operates in fully observable domains. Partially observable environments such as the ones encountered in self-driving vehicle scenarios deal with partial information in order to solve AI problems.

Single Agent vs. Multi Agent

A multi-agent system (MAS or "self-organized system") is a computerized system composed of multiple interacting intelligent

agents. Multi-agent systems can solve problems that are difficult or impossible for an individual agent or a monolithic system to solve. Intelligence may include methodic, functional, procedural approaches, algorithmic search or reinforcement learning. Despite considerable overlap, a multi-agent system is not always the same as an agent-based model (ABM). Multi-agent systems consist of agents and their environment. Typically multi-agent systems research refers to software agents. However, the agents in a multi-agent system could equally well be robots, humans or human teams. A multi-agent system may contain combined human-agent teams.

Single-agent systems should be simpler than multi-agent systems when dealing with a fixed, complex task, the opposite is often the case. Single-agent systems belong at the end of the progression from simple to complex multi-agent systems. The agent in a single-agent system model itself, the environment, and their interactions. In a single-agent system, no other such entities are recognized by the agent.

Exercis

1. What is the difference between Agent function and Agent Program?
2. What are intelligent agents and how are they used in AI?
3. What is machine learning and how does it relate to AI?
4. What is a learning agent and how are they applied?
5. How plausible is human-like artificial intelligence?
6. What is intelligent agent? Describe various types of agents.
7. Differentiate between simple reflexive and model based agents with example.
8. Differentiate between goal based agent and Utility based agent.
9. Describe following terms
   - Simple Reflexive agents
   - Model Based agents
   - Goal Based agents
   - Utility Based agents
10. Describe properties of environment with example.
11. For each of the following agents, determine task environment
    - Medical diagnosis system
    - Satellite image analysis system
    - Part-pricking robot
    - Refinery controller
12. For each of the following agents, identify the six properties of environment
    - Medical diagnosis system
    - Satellite image analysis system
    - Part-pricking robot
    - Soccer player
13. For each of the following agents, determine what type of agent architecture is most appropriate (i.e. simple reflex, Model based, goal-based or utility-based).
    - Medical diagnosis system
    - Satellite image analysis system
    - Part-pricking robot
    - Refinery controller

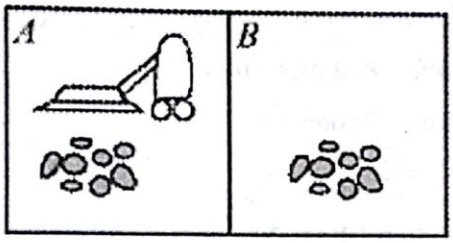
Lab Manual

LAB NO 1:

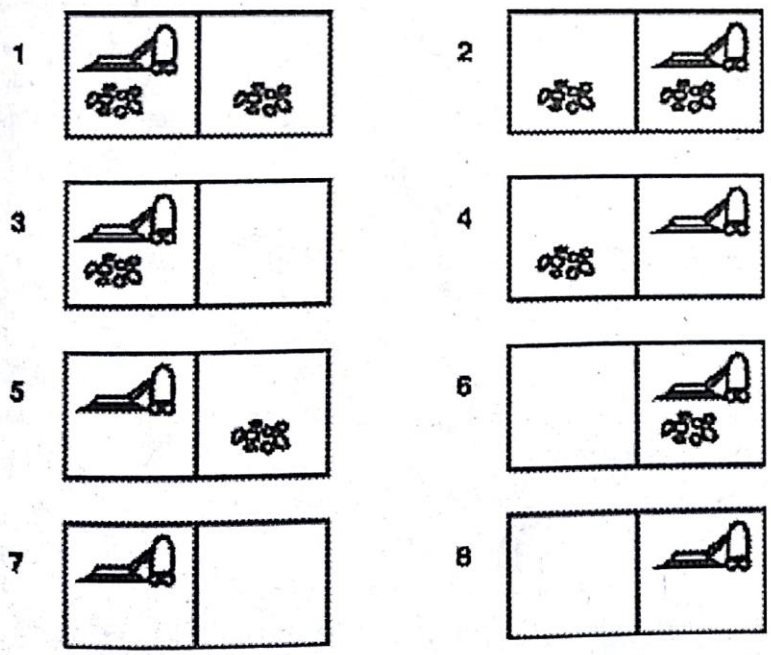
Write programs for implementing simple intelligent agents (Simple reflex agent for Vacuum Cleaner agent)

Problem Descriptions:

Consider a Vacuum cleaner world:



Let us consider that our intelligent agent is a robot vacuum cleaner. Let's suppose that the world has just two rooms. The robot can be in either room and there can be dirt in zero, one, or two rooms as shown below:



Our goal is to reach state 7 or 8. Formally the problem can be defined as PAGE description:

Percepts: [location and content] *E.g. [A, Dirty]*

Actions: left, right, suck, and no-op

Goal: Dirt clean in both room

Environment: square A and B.

The pseudo-code for the above agent is:

```
function Reflex-Vacuum-Agent (location, status) return an action
    if status = Dirty ten return Suck
    else if location = A then return Right
    else if location = B then return Left
```

The python code for the

Please import the code form HTML file

✿✿✿

UNIT 3

Problem Solving By Searching

OUTLINE

- ✓ **Searching**
- ✓ **Problem Solving Agent**
  - Problem Formulation
  - State Space representation
  - Well-defined Problem
- ✓ **Types of Search Algorithms**
  - Solving Problems by Searching
  - Search Strategies
  - Uninformed Search
  - Informed or Heuristic search
  - A* Search
  - Hill Climbing Search
  - Simulated Annealing
  - Game Playing
  - Adversarial Search Techniques
  - The Mini-max Search
  - Alpha-Beta Pruning
- ✓ **Constraint Satisfaction Problems**

Searching

Search is a commonly used method in Artificial Intelligence for solving problems. The search technique explores the possible moves that one can make in a space of 'states', called the search space. The solutions or 'goal states' could sometimes be an object, a goal, a sub-goal or a path to the searched item.

Problem Solving Agent

It decides what to do by finding sequence of actions that leads to goal or desirable situations. This is a kind of goal based agent which always tries to achieve goal. In many situations, "to solve a problem" can be described as to change the current situation, step by step, from an initial state to a final state. If each state is represented by a node, and each possible change is represented by a link, then a "problem" can be represented as a graph (the "state space"), with a "solution" corresponding to a path from the initial state to a final state.

In this way, a solution consists of a sequence of operations, each of which changes one state into another one, and the whole sequence changes the initial state into a final state in multiple steps.

Four general steps in problem solving

1. **Goal formulation**

   A goal is a state that the agent is trying to reach. This step formulates the goal based on current situation and the agent's performance.

2. **Problem formulations**

   This is the process of deciding what actions and states to consider, given goal.

3. **Search Method**

   Determine the possible sequence of actions that lead to the states of known values and then choosing the best sequence.

4. **Execute**

   Once the solution is found, the actions it recommends can be carried out. This is called the execution phase.

Problem Formulation

Before finding the algorithm for evaluating the problem and searching for the solution, we first need to define and formulate the problem. Problem formulation involves deciding what actions and states to consider, given the goal. For example, if the agent were to consider the action to be at the level of "move the left foot by one inch" or "turn the steering wheel by 1 degree left", there would be too many steps for the agent to leave the parking lot, let alone to reach the goal. In general, we need to abstract the state details from the representation. A problem can be defined formally by 4 components:

1. **Initial State**

   It is the state from which our agents start solving the problem {i.e.: in (A)}.

2. **State Description**

   - A description of the possible actions available to the agent, it is common to describe it by means of a successor function, given state x then SUCCESSOR-FN(x) returns a set of ordered pairs <action, successor> where action is a legal action from state x and successor is the state in which we can be by applying action.
   - The initial state and the successor function together defined what is called state space which is the set of all possible states reachable from the initial state {i.e.: in(A), in(B), in(C), in(D), in(E)}.

3. **Goal Test:**

   We should be able to decide whether the current state is a goal state {i.e.: is the current state is in (E)?}.

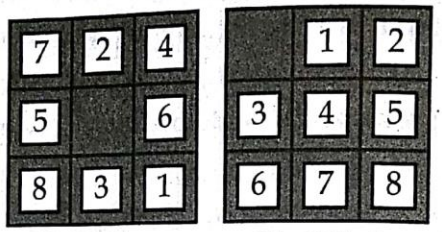
4. **Path cost:**

   A function that assigns a numeric value to each path, each step we take in solving the problem should be somehow weighted, so If I travel from A to E our agent will pass by many cities, the cost to

travel between two consecutive cities should have some cost measure, {i.e.: Traveling from 'A' to 'B' costs 20 km or it can be typed as c(A, 20, B)}.

A solution to a problem is path from the initial state to a goal state, and solution quality is measured by the path cost, and the optimal solution has the lowest path cost among all possible solutions.

Example:



Start State Goal State

8-puzzle

1. **Initial state**

   Our board can be in any state resulting from making it in any configuration.

2. **State description**

   - Successor function generates legal states resulting from applying the three actions {move blank Up, Down, Left, or Right}.
   - State description specifies the location of each of the eight titles and the blank.

3. **Goal test**

   Checks whether the states matches the goal configured in the goal state shown in the picture

4. **Path cost**

   Each step costs 1, so the path cost is the sum of steps in the path.

State Space representation

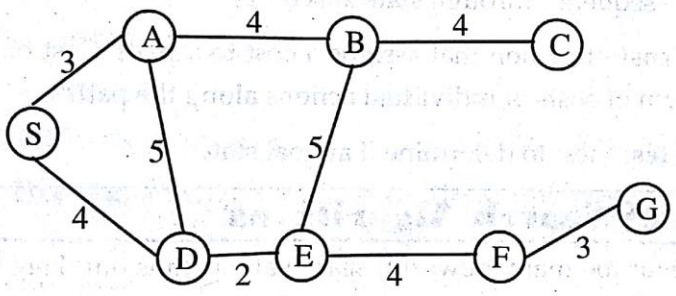
One general formulation of intelligent action is in terms of state space. A state contains all of the information necessary to predict the effects of an action and to determine if it is a goal state. State-space searching assumes that:

- the agent has perfect knowledge of the state space and can observe what state it is in (i.e., there is full observability);
- the agent has a set of actions that have known deterministic effects;
- some states are goal states, the agent wants to reach one of these goal states, and the agent can recognize a goal state; and
- A solution is a sequence of actions that will get the agent from its current state to a goal state.

State space is the set of all states reachable from the initial state. This is defined implicitly by initial state and successor function (Actions) together.

The state space is commonly represented as a directed graph in which each node is a state and each arc represents the application of an operator transforming a state to a successor state. A solution is a path from the initial state to a goal state.

Example of problem formulation: Route Finding Problem



Problem formulation may be as follows:

- **states**
  Locations (e.g.: A, B ...)
- **initial state**
  Starting point (e.g.: S)
- **successor function (operators)**
  Move from one location to another
- **goal test**
  Arrive at a certain location (e.g.: G)
- **path cost**
  Total distance, money, time, travel comfort etc. (here distance in km)

Well-defined Problem

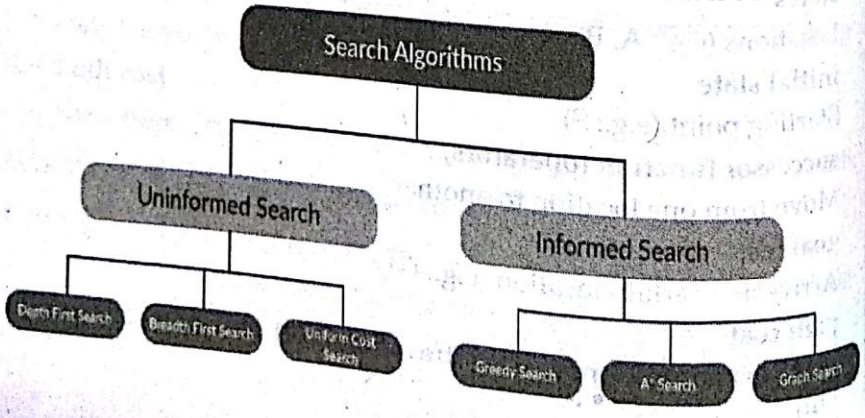
In problem solving, any problem in which the initial state or starting position, the allowable operations, and the goal state are clearly specified, and a unique solution can be shown to exist. Typical examples are the Tower of Hanoi, water-jar problems etc. A problem that lacks one or more of these specified properties is an ill-defined problem, and most problems that are encountered in everyday life fall into this category. Also called a well-defined problem. A well-defined problem can be described by:

Initial state

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

Types of Search Algorithms

There are far too many powerful search algorithms out there to fit in a single article. Instead, this article will discuss six of the fundamental search algorithms, divided into two categories, as shown below.



Solving Problems by Searching

Figure below contains a representation of a map. The nodes represent cities, and the links represent direct road connections between cities. The number associated to a link represents the length of the corresponding road. The search problem is to find a path from a city S to a city G

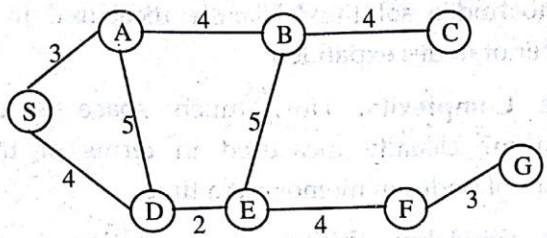


Figure: A graph representation of a map

This problem will be used to illustrate some search methods. Search problems are part of a large number of real world applications:

- VLSI layout
- Web search
- Path planning
- Robot navigation etc.

Search Strategies

Searching is a process to find the solution for a given set of problems. There are two broad classes of search methods:

1. Uninformed (or blind) search methods
2. Heuristically informed search methods

In the case of the uninformed search methods, the order in which potential solution paths are considered is arbitrary, using no domain-specific information to judge where the solution is likely to lie. E.g. Breath first search, Depth first search, Iterative deepening search, Depth limit search, Bidirectional search etc.

In the case of the heuristically informed search methods, one uses domain-dependent (heuristic) information in order to search the space more efficiently. E.g. Greedy best first search, A* search, Hill climbing search, Simulated annealing etc.

Performance evaluation of search techniques

We will evaluate the performance of a search algorithm in four ways

- **Completeness:** An algorithm is said to be complete if it definitely finds solution to the problem, if exist.
- **Time Complexity:** How long (worst or average case) does it take to find a solution? Usually measured in terms of the number of nodes expanded
- **Space Complexity:** How much space is used by the algorithm? Usually measured in terms of the maximum number of nodes in memory at a time
- **Optimality/Admissibility:** If a solution is found, is it guaranteed to be an optimal one? For example, is it the one with minimum cost?

Time and space complexity are measured in terms of

- b -- Maximum branching factor (number of successor of any node) of the search tree
- m -- Depth of the least-cost solution
- d -- Maximum length of any path in the space

Uninformed Search

In the case of the uninformed search methods, the order in which potential solution paths are considered is arbitrary, using no domain-specific information to judge where the solution is likely to lie. E.g: Breath first search, Depth first search, Iterative deepening search, Depth limit search, Bidirectional search etc.

Breadth First Search

Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a 'search key'), and explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level.

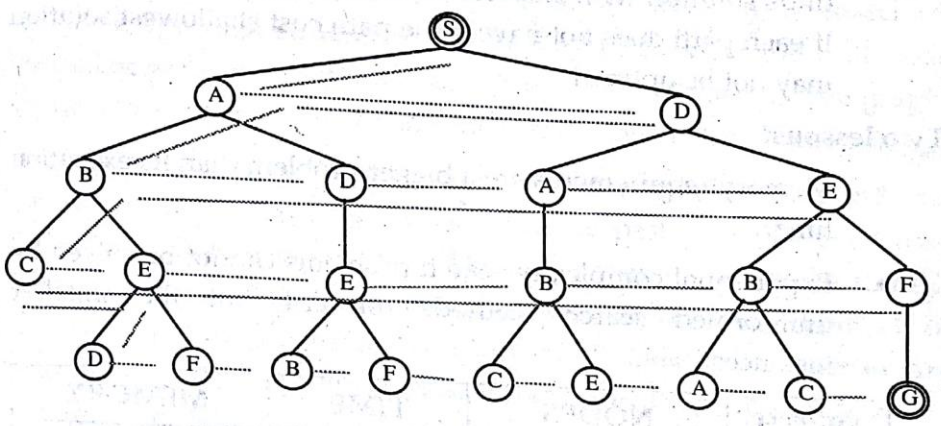
All nodes are expended at a given depth in the search tree before any nodes at the next level are expanded until the goal reached i. e. as the

name BFS suggests; you are required to traverse the graph breadth wise as follows:

- First move horizontally and visit all the nodes of the current layer
- Move to the next layer

Expand shallowest unexpanded node.

Constraint: Do not generate as child node if the node is already parent to avoid more loop.



Performance Measures

Completeness:

- Does it always find a solution if one exists?
- YES
  - ☑ If shallowest goal node is at some finite depth d and If b is finite

Time complexity:

- Assume a state space where every state has b successors.
  - ✓ root has b successors, each node at the next level has again b successors (total b^2), …
  - ✓ Assume solution is at depth d
  - ✓ Worst case; expand all except the last node at depth d
  - ✓ Total no. of nodes generated:

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

Space complexity:

- Each node that is generated must remain in memory
- Total no. of nodes in memory:
- $1 + b + b^2 + b^3 + \ldots\ldots\ldots\ldots\ldots\ldots\ldots b^d + (b^{d+1} - b) = O(b^{d+1})$

Optimal (i.e., admissible):

- if all paths have the same cost. Otherwise, not optimal but finds solution with shortest path length (shallowest solution). If each path does not have same path cost shallowest solution may not be optimal

Two lessons:

- Memory requirements are a bigger problem than its execution time.
- Exponential complexity search problems cannot be solved by uninformed search methods for any but the smallest instances.

DEPTH	NODES	TIME	MEMORY
2	1100	0.11 seconds	1 megabyte
4	111100	11 seconds	106 megabytes
6	107	19 minutes	10 gigabytes
8	109	31 hours	1 terabyte
10	1011	129 days	101 terabytes
12	1013	35 years	10 petabytes
14	1015	3523 years	1 exabyte

Applications

- **Social Networking Websites:**
- In social networks, we can find people within a given distance 'k' from a person using Breadth First Search till 'k' levels.
- **Crawler sin Search Engines:**

  Crawlers build index using Breadth First.

- **GPS Navigation systems:**

  Breadth First Search is used to find all neighboring locations.

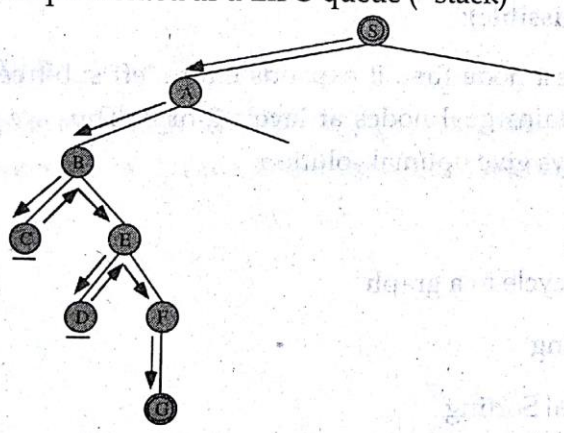
- **Broadcasting in Network:**

  In networks, a broadcasted packet follows Breadth First Search to reach all nodes.

Depth First Search

Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking.

The main strategy of depth-first search is to explore deeper into the graph whenever possible i.e. expand deepest unexpanded node. When all edges have been explored, the search backtracks until it reaches an unexplored neighbor. This process continues until all of the vertices that are reachable from the original source vertex are discovered. DFS uses a stack data structure to keep track of vertices. Fringe is implemented as a LIFO queue (=stack)



Performance Measures

Completeness:

- Does it always find a solution if one exists?
- NO
  - ✓ If search space is infinite and search space contains loops then DFS may not find solution.

Time Complexity

1. Let m is the maximum depth of the search tree. In the worst case Solution may exist at depth m.
2. root has *b* successors, each node at the next level has again b successors (total b^2), ...
3. Worst case; expand all except the last node at depth *m*
4. Total no. of nodes generated:

$$b + b^2 + b^3 + \ldots\ldots\ldots\ldots\ldots\ldots\ldots b^m = O(b^m)$$

Space Complexity:

It needs to store only a single path from the root node to a leaf node, along with remaining unexpanded sibling nodes for each node on the path.

Total no. of nodes in memory:

$1 + b + b + b + \ldots\ldots\ldots\ldots\ldots\ldots$ b m times = **O(bm)**

Optimal (i.e., admissible):

DFS expand deepest node first, if expands entire left sub-tree even if right sub-tree contains goal nodes at levels 2 or 3. Thus we can say DFS may not always give optimal solution.

Applications:

- Detecting cycle in a graph
- Path Finding
- Topological Sorting
- Bipartite graph test
- Solving puzzles with only one solution, such as mazes.

Uniform Cost Search

Uniform-cost search is modified version of BFS to make optimal. Uniform Cost Search is also called the Cheapest First Search. It is

basically a tree search algorithm used for traversing or searching a weighted tree or graph. The search begins at the root node. The search continues by visiting the next node which has the least total cost from the root.

For any step-cost function, uniform cost search expands the node with least path cost. To implement this, the frontier will be stored in a priority queue.. In the queue, each node is associated with its total path cost from the root, where the least-cost paths are given highest priority. The node at the head of the queue is subsequently expanded, adding the next set of connected nodes with the total path cost from the root to the respective node. It does not care about the number of steps, only care about total cost.

Performance Measures

Until the goal node is found. Hence, the space complexity is also O(b^[1+(C*/e)]).

Optimality: Yes. Because, at every step the path with the least cost is chosen

Conclusion

UCS can be used instead of BFS in case that path cost is not equal and is guaranteed to be greater than a small positive value e.

Consider an example:

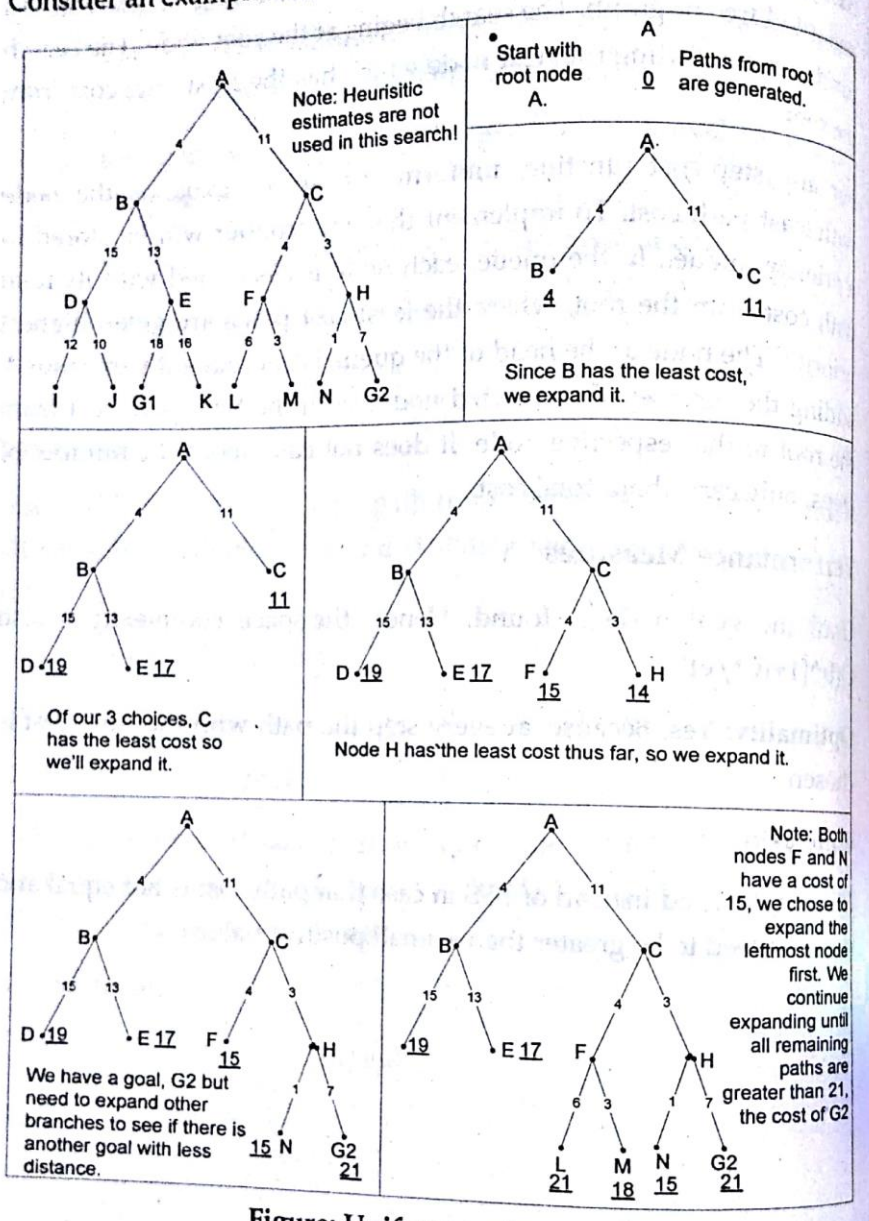


Figure: Uniform cost search

Depth Limited Search

It is a modification of depth-first search .The problem of unbounded trees can be solve by supplying depth-first search with a determined depth limit (nodes at depth are treated as they have no successors) - Depth limited search. Depth-limited search is an algorithm to explore

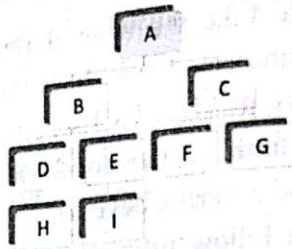
the vertices of a graph. Like the normal depth-first search, depth-limited search is an uninformed search. It works exactly like depth-first search, but avoids its drawbacks regarding completeness by imposing a maximum limit on the depth of the search. Even if the search could still expand a vertex beyond that depth, it will not do so and thereby it will not follow infinitely deep paths or get stuck in cycles. Therefore depth-limited search will find a solution if it is within the depth limit, which guarantees at least completeness on all graphs.

It solves the infinite-path problem of DFS. Yet it introduces another source of problem if we are unable to find good guess of l. Let d is the depth of shallowest solution.

Algorithm

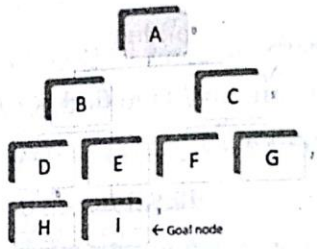
```
depth limit = max depth to search to;
agenda = initial stage:
If initial stage is goal stage then return
solution
else
while agenda not empty do
take node from front of agenda;
If depth (node) < depth limit then
{
new does = apply opertions to node;
add new nodes to front of agenda;
if goal stage in new nodes then return
solution;
}
```

Example: Find DLS of following graph by using A as source and I as destination vertex with using depth limit=2.

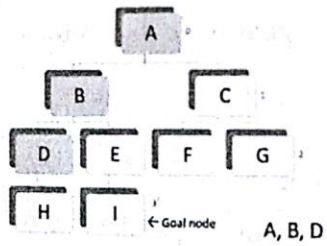


Solution:

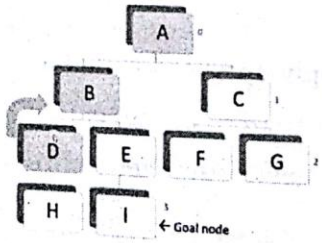
Step 1



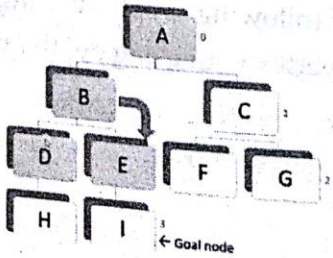
Step 2:



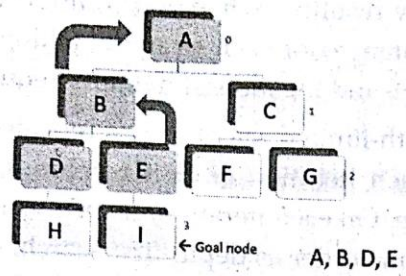
Step 3: Backtracking



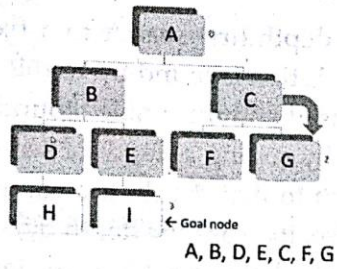
Step 4:



Step 5: Backtracking



Step 6



Since depth limit we taken is 2 but goal node at level 3 so searches unsuccessful.

Performance Measure

Completeness

The limited path introduces another problem which is the case when we choose l < d, in which is our DLS will never reach a goal, in this case we can say that DLS is not complete i.e. If $l < d$ then incompleteness results.

Optimality

- One can view DFS as a special case of the depth DLS, that DFS is DLS with l = infinity.
- DLS is not optimal even if l > d.

Time complexity: $O(b^l)$

Space complexity: $O(b^l)$

Conclusion

DLS can be used when the there is a prior knowledge to the problem, which is always not the case

Iterative Deepening Depth First Search

It is a search strategy resulting when you combine BFS and DFS, thus combining the advantages of each strategy, taking the completeness and optimality of BFS and the modest memory requirements of DFS.

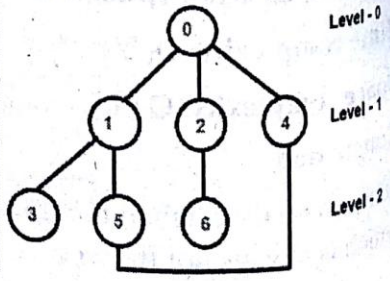
In this strategy, depth-limited search is run repeatedly, increasing the depth limit with each iteration until it reaches *d*, the depth of the shallowest goal state. On each iteration, IDDFS visits the nodes in the search tree in the same order as depth-first search, but the cumulative order in which nodes are first visited, assuming no pruning, is effectively breadth-first.

Iterative deepening is depth-first search to a fixed depth in the tree being searched. If no solution is found up to this depth then the depth to be searched is increased and the whole `bounded' depth-first search begun again. It works by setting a depth of search -say, depth 1- and doing depth-first search to that depth. If a solution is found then the process stops -otherwise, increase the depth by, say, 1 and repeat until a solution is found. Note that every time we start up a new bounded depth search *we start from scratch* - i.e. we throw away any results from the previous search.

Note that depth-first search achieves its efficiency by generating the next node to explore only when this needed. The breadth-first search algorithm has to grow all the search paths available until a solution is found -and this takes up memory. Iterative deepening achieves its memory saving in the same way that depth-first search does -at the expense of redoing some computations again and again (a time cost rather than a memory one). In the search illustrated, we had to visit node d three times in all.

Example 1:

Depth	Iterative Deepening Depth First Search
0	0
1	0 1 2 4
2	0 1 3 5 2 6 4 5
3	0 1 3 5 4 2 6 4 5 1



Performance Measure

Completeness

IDS is like BFS, is complete when the branching factor b is finite.

Optimality

IDS is also like BFS optimal when the steps are of the same cost.

Time Complexity

- Node generation
  - ✓ level d: once
  - ✓ level d-1: 2
  - ✓ level d-2: 3
  - ✓ ...
  - ✓ level 2: d-1
  - ✓ level 1: d
    - o Total no. of nodes generated:

$$d.b + (d-1).\ b^2 + (d-2).\ b^3 + \ldots\ldots\ldots\ldots\ldots\ldots + 1.\ b^d = O(b^d)$$

Example 2: If we consider a realistic numbers, and use b = 10 and d = 5, then number of generated nodes in BFS and IDS will be like

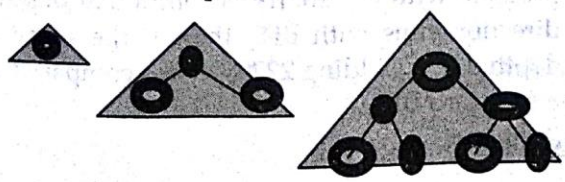
N(IDS) = 50 + 400 + 3000 + 20000 + 100000 = 123450

N(BFS) = 10 + 100 + 1000 + 10000 + 100000 + 999990 = 1111100

BFS generates like 9 time nodes to those generated with IDS.

Space Complexity

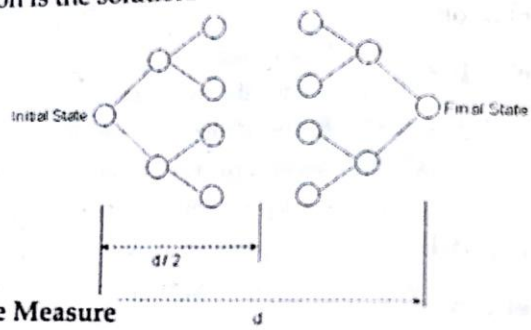
IDS is like DFS in its space complexity, taking $O(bd)$ of memory.



Notice that BFS generates some nodes at depth d+1, whereas IDS does not. The result is that IDS is actually faster than BFS, despite the repeated generation of node.

Bidirectional Search

As the name suggests, bidirectional search suggests to run two simultaneous search graph, one from the initial state and the other from the goal state. It then expands nodes from the start and goal state simultaneously. Check at each stage if the nodes of one have been generated by the other, i.e., they meet in the middle. If so, the path concatenation is the solution.



Performance Measure

- **Completeness**

  YES. Bidirectional search is complete when we use BFS in both searches, the search that starts from the initial state and the other from the goal state.

- **Optimality**
  - ✓ Optimal when BFS is used and paths are of a uniform cost - all steps of the same cost.
  - ✓ Other search strategies can be used like DFS, but this will sacrifice the optimality and completeness, any other combination than BFS may lead to a sacrifice in optimality or completeness or may be both of them.

- **Time complexity**

  Both searches will run the same amount of time meeting in the middle of the graph, thus each search expands $O(b^{d/2})$ node

  For e.g.: If a problem with b = 10, has a solution at depth d = 6, and each direction runs with BFS, then at the worst case they meet at depth d = 3, yielding 22200 nodes compared with 11111100 for a standard BFS.

- **Space complexity:** $O(b^{d/2})$

Conclusion

Bidirectional search seems attractive for its $O(b^{d/2})$ performance, but things are not that easy, especially the implementation part. i. e that is going from the head to the tail is like going from the tail to the head.

Drawbacks of uniformed search

- Criterion to choose next node to expand depends only on a global criterion: level.
- Does not exploit the structure of the problem.
- One may prefer to use a more flexible rule that takes advantage of what is being discovered on the way, and hunches about what can be a good move.
- Very often, we can select which rule to apply by comparing the current state and the desired state

Informed or Heuristic search

Heuristic Search uses domain-dependent (heuristic) information in order to search the space more efficiently. It is also known as informed search. In other words, Heuristic search refers to a search strategy that attempts to optimize a problem by iteratively improving the solution based on a given heuristic function or a cost measure. In this search method, it might not always find the best solution but is guaranteed to find a good solution in reasonable time. By sacrificing completeness it increases efficiency. It is useful in solving tough problems which

- Could not be solved any other way.
- Solutions take an infinite time or very long time to compute.

Ways of using heuristic information

- Deciding which node to expand next, instead of doing the expansion in a strictly breadth-first or depth-first order;

- In the course of expanding a node, deciding which successor or successors to generate, instead of blindly generating all possible successors at one time;
- Deciding that certain nodes should be discarded, or pruned, from the search space.

Heuristic Searches - Why Use?

- It may be too resource intensive (both time and space) to use a blind search
- Even if a blind search will work we may want a more efficient search method

Informed Search uses domain specific information to improve the search pattern

- Define a heuristic function, h(n), that estimates the "goodness" of a node n.
- Specifically, h(n) = estimated cost (or distance) of minimal cost path from n to a goal state.
- The heuristic function is an estimate, based on domain-specific information that is computable from the current state description, of how close we are to a goal.

Best-First Search

BFS and DFS blindly explore paths without considering any cost function. The idea of Best First Search is to use an evaluation function to decide which adjacent is most promising and then explore. This "best first" behavior is implemented with a Priority Queue.

Idea: use an *evaluation function f(n)* that gives an indication of which node to expand next for each node.

- Usually gives an estimate to the goal.
- The node with the lowest value is expanded first.

A key component of *f(n)* is a heuristic function, *h(n)*,which is a additional knowledge of the problem. There is a whole family of best-first search strategies, each with a different evaluation function.

Typically, strategies use estimates of the cost of reaching the goal and try to minimize it.

Special cases: based on the evaluation function.

- Greedy best-first search
- A*search

Greedy Best First Search

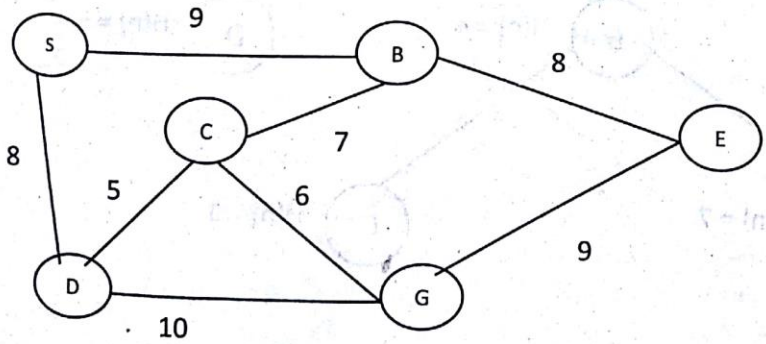
The best-first search part of the name means that it uses an evaluation function to select which node is to be expanded next. The node with the lowest evaluation is selected for expansion because that is the *best* node, since it supposedly has the closest path to the goal (if the heuristic is good). Unlike A* which uses both the link costs and a heuristic of the cost to the goal, greedy best-first search uses only the heuristic, and not any link costs. A disadvantage of this approach is that if the heuristic is not accurate, it can go down paths with high link cost since there might be a low heuristic for the connecting node.

Evaluation function $f(n) = h(n)$ (heuristic) = estimate of cost from n to *goal*.

e.g., $h_{SLD}(n)$ = straight-line distance from n to goal

Greedy best-first search expands the node that appears to be closest to goal. The greedy best-first search algorithm is $O(b^m)$ in terms of space and time complexity. (Where b is the average branching factor (the average number of successors from a state), and m is the maximum depth of the search tree.)

Example: For example consider the following graph



Straight Line distances to node G (goal node) from other nodes is given below:

S→ G = 12

B→ G = 4

E→ G = 7

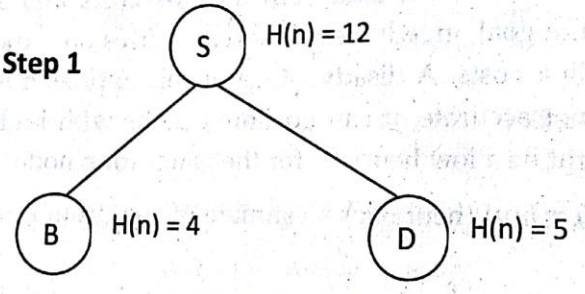
D→ G = 5

C→ G = 3

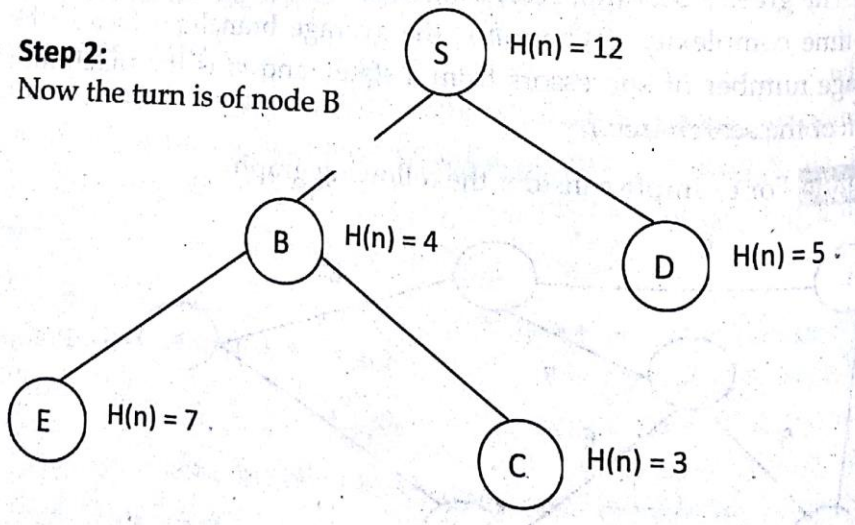
Let H(n)= Straight Line distance

Now Greedy Search operation is done as below:

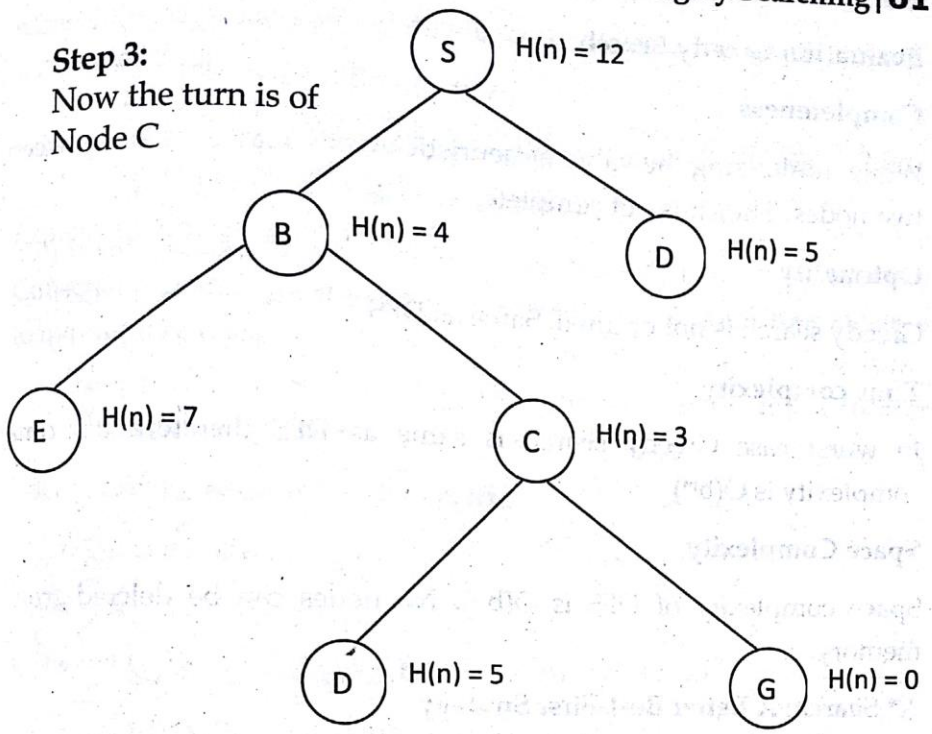
Step 1



Step 2:
Now the turn is of node B

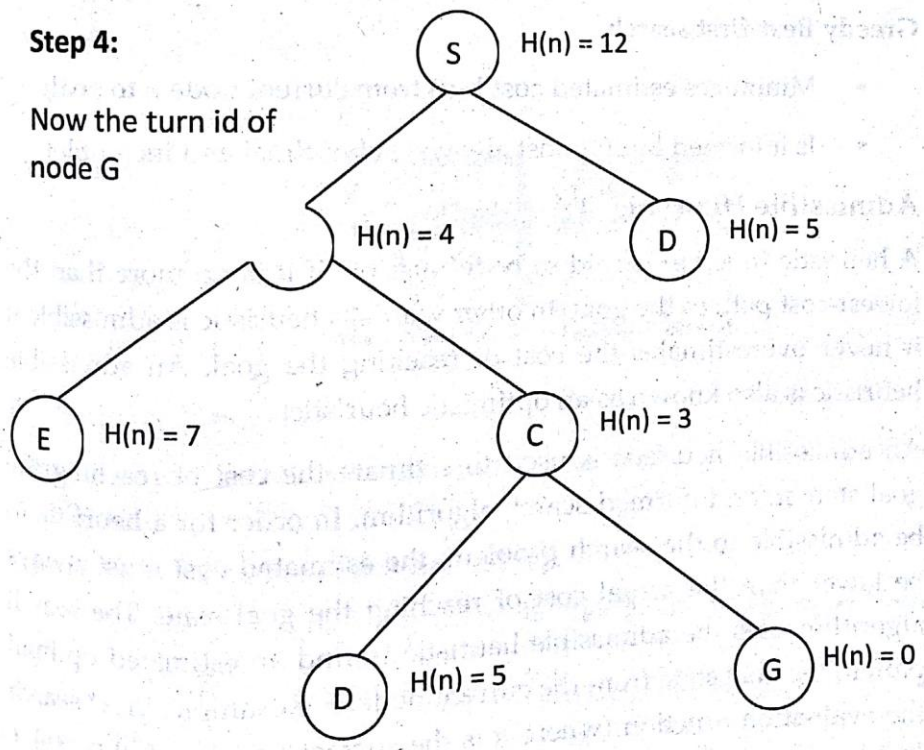


Step 3:
Now the turn is of Node C



Step 4:

Now the turn id of node G



Evaluation Greedy Search

Completeness

While minimizing the value of heuristic greedy may oscillate between two nodes. Thus it is not complete.

Optimality

Greedy search is not optimal. Same as DFS.

Time complexity

In worst case Greedy search is same as DFS therefore it's time complexity is $O(b^m)$.

Space Complexity

Space complexity of DFS is $O(b^m)$. No nodes can be deleted from memory.

A* Search: A Better Best-First Strategy

Greedy Best-first search

- Minimizes estimated cost h(n) from current node n to goal;
- Is informed but (almost always) suboptimal and incomplete.

Admissible Heuristic

A heuristic function is said to be admissible if it is no more than the lowest-cost path to the goal. In other words, a heuristic is admissible if it never overestimates the cost of reaching the goal. An admissible heuristic is also known as an optimistic heuristic.

An admissible heuristic is used to estimate the cost of reaching the goal state in an informed search algorithm. In order for a heuristic to be admissible to the search problem, the estimated cost must always be lower than the actual cost of reaching the goal state. The search algorithm uses the admissible heuristic to find an estimated optimal path to the goal state from the current node. For example, in A* search the evaluation function (where n is the current node) is: $f(n) = g(n) + h(n)$

Where;

$f(n)$ = the evaluation function.

$g(n)$ = the cost from the start node to the current node

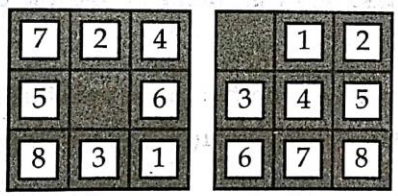
$h(n)$ = estimated cost from current node to goal.

h(n) is calculated using the heuristic function. With a non-admissible heuristic, the A* algorithm would overlook the optimal solution to a search problem due to an overestimation in f(n). It is obvious that the SLD heuristic function is admissible as we can never find a shorter distance between any two towns.

Formulating admissible heuristics

- n is a node
- h is a heuristic
- $h(n)$ is cost indicated by h to reach a goal from n
- $C(n)$ is the actual cost to reach a goal from n
- h is admissible if

$$\forall n, h(n) \leq C(n)$$



Start State Goal State

For Example: 8-puzzle

Figure shows 8-puzzle start state and goal state. The solution is 26 steps long.

$h_1(n)$ = number of misplaced tiles

$h_2(n)$ = sum of the distance of the tiles from their goal position (notdiagonal).

$h_1(S)$ = ? 8

$h_2(S)$ = ? 3+1+2+2+2+3+3+2 = 18

$h_n(S) = \max\{h1(S), h2(S)\} = 18$

Consistency (Monotonicity)

A heuristic is said to be consistent if for any node N and any successor N′ of N , estimated cost to reach to the goal from node N is less than the sum of step cost from N to N′ and estimated cost from node N′ to goal node.

i.e. $h(n) \leq c(n, n') + h(n')$

Where;

$h(n)$ = Estimated cost to reach to the goal node from node n

$c(n, n')$ = actual cost from n to n′

A* Search

A* is a best first, informed graph search algorithm. A* is different from other best first search algorithms in that it uses a heuristic function h(x) as well as the path cost to the node g(x), in computing the cost $f(x) = h(x) + g(x)$ for the node. The $h(x)$ part of the $f(x)$ function must be an admissible heuristic; that is, it must not overestimate the distance to the goal. Thus, for an application like routing, $h(x)$ might represent the straight-line distance to the goal, since that is physically the smallest possible distance between any two points or nodes.

It finds a minimal cost-path joining the start node and a goal node for node n. Evaluation function: $f(n) = g(n) + h(n)$

Where,

$g(n)$ = cost so far to reach n from root

$h(n)$ = estimated cost to goal from n

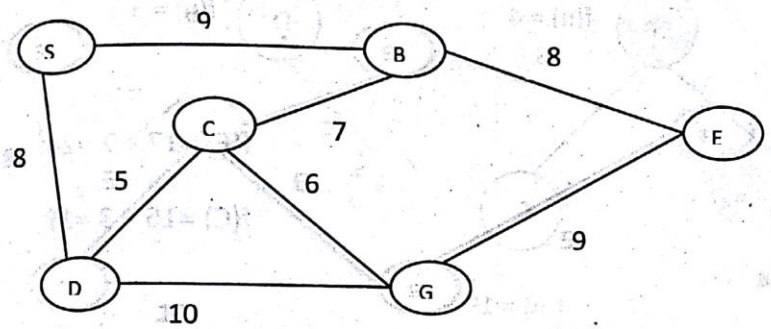
$f(n)$ = estimated total cost of path through n to goal

- combines the two by minimizing $f(n) = g(n) + h(n)$;
- is informed and, under reasonable assumptions, optimal and complete.

As A* traverses the graph, it follows a path of the lowest *known* path, keeping a sorted priority queue of alternate path segments along the way. If, at any point, a segment of the path being traversed has a

higher cost than another encountered path segment, it abandons the higher-cost path segment and traverses the lower-cost path segment instead. This process continues until the goal is reached.

A* Search Example: For example consider the following graph



Straight Line distances to node G (goal node) from other nodes is given below:

S→G = 12

B→G = 4

E→G = 7

D→G = 6

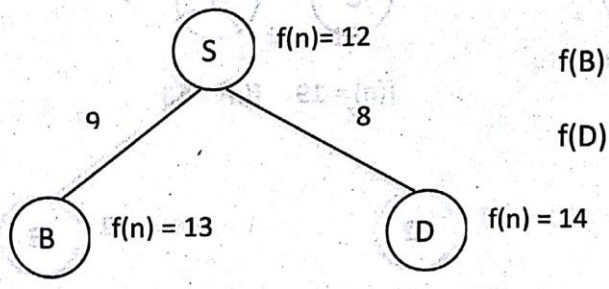
C→G = 3

Labels in the graph shows actual distance.

Let H(n)= Straight Line distance

Now A* Search operation is done as below:

Step 1

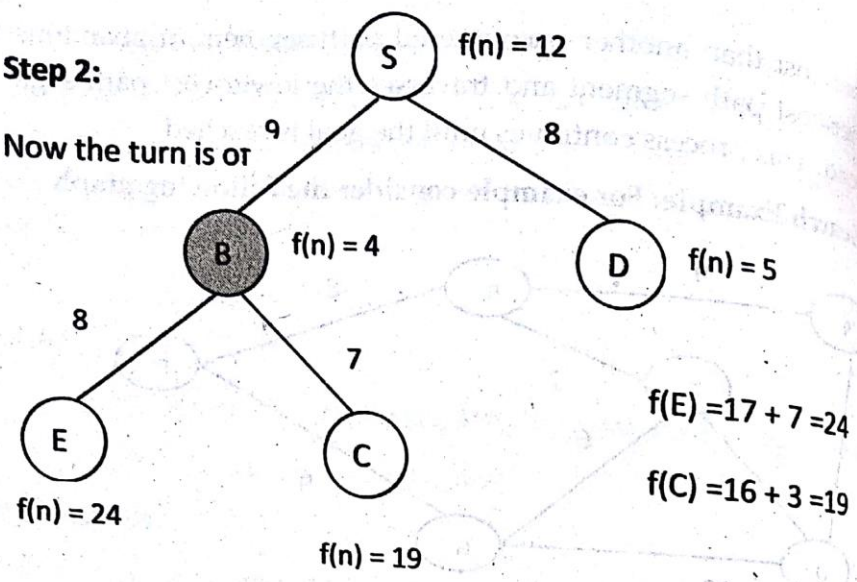


f(B) = 9 + 4 =13

f(D) = 8 + 6 =14

Step 2:

Now the turn is of

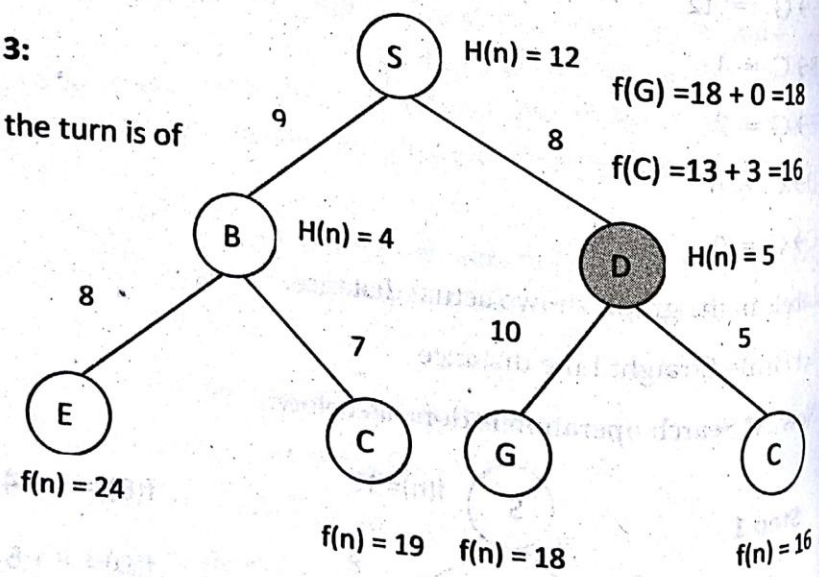


f(E) =17 + 7 =24

f(C) =16 + 3 =19

Step 3:

Now the turn is of

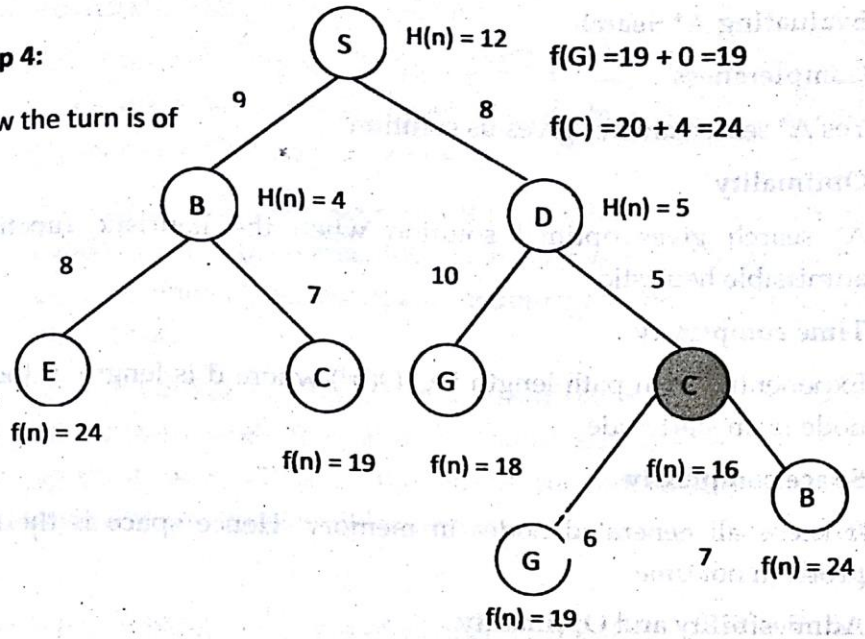


f(G) =18 + 0 =18

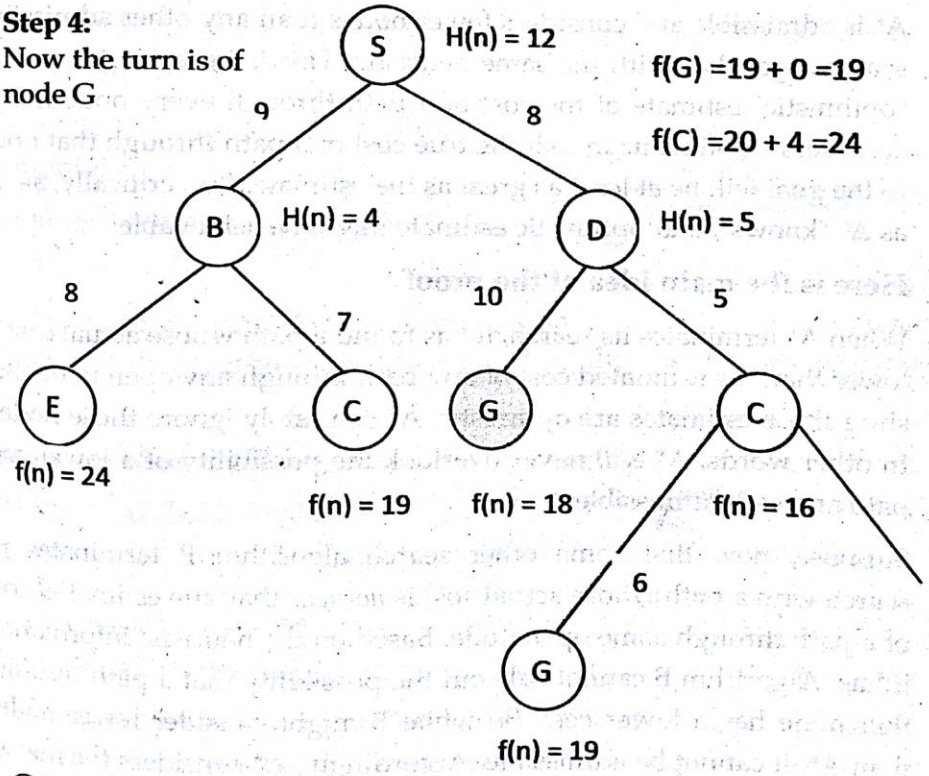
f(C) =13 + 3 =16

Step 4:

Now the turn is of



Step 4:
Now the turn is of node G



G is goal node, hence we are done

Evaluating A* Search

Completeness

Yes A* search always gives us solution

Optimality

A* search gives optimal solution when the heuristic function is admissible heuristic.

Time complexity

Exponential with path length i.e. $O(b^d)$ where d is length of the goal node from start node.

Space complexity

It keeps all generated nodes in memory. Hence space is the major problem not time

Admissibility and Optimality

A* is admissible and considers fewer nodes than any other admissible search algorithm with the same heuristic. This is because A* uses an "optimistic" estimate of the cost of a path through every node that it considers—optimistic in that the true cost of a path through that node to the goal will be at least as great as the estimate. But, critically, as far as A* "knows", that optimistic estimate might be achievable.

Here is the main idea of the proof

When A* terminates its search, it has found a path whose actual cost is lower than the estimated cost of any path through any open node. But since those estimates are optimistic, A* can safely ignore those nodes. In other words, A* will never overlook the possibility of a lower-cost path and so is admissible.

Suppose, now that some other search algorithm B terminates its search with a path whose actual cost is *not* less than the estimated cost of a path through some open node. Based on the heuristic information it has, Algorithm B cannot rule out the possibility that a path through that node has a lower cost. So while B might consider fewer nodes than A*, it cannot be admissible. Accordingly, A* considers the fewest nodes of any admissible search algorithm.

This is only true if both:

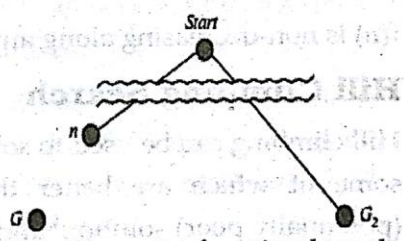
- A* uses an admissible heuristic. Otherwise, A* is not guaranteed to expand fewer nodes than another search algorithm with the same heuristic.
- A* solves only one search problem rather than a series of similar search problems. Otherwise, A* is not guaranteed to expand fewer nodes than incremental heuristic search algorithms

Thus, if estimated distance h(n) never exceed the true distance h*(n) between the current node to goal node, the A* algorithm will always find a shortest path -This is known as the admissibility of A* algorithm and h(n) is a admissible heuristic.

IF 0 =< h (n) =< h*(n), and costs of all arcs are positive

THEN A* is guaranteed to find a solution path of minimal cost if any solution path exists.

Theorem: A* is optimal if h(n) is admissible



Suppose suboptimal goal G2 in the queue.

Let n be an unexpanded node on a shortest path to optimal goal G and C* be the cost of optimal goal node.

f(G2) = h (G2) + g(G2)

f(G2) = g(G2), since h(G2)=0

f(G2) > C* .....(1)

Again, since h(n) is admissible, It does not overestimates the cost of completing the solution path.

f(n) = g(n) + h(n) ≤C* .....(2)

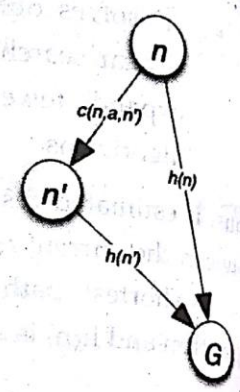
Now from (1) and (2) f(n) ≤ C* < f(G2)

Since f(G2) > f(n), A* will never select G2 for expansion. Thus A* gives us optimal solution when heuristic function is admissible.

Theorem: If h(n) is consistent , then the values of f(n) along the path are non-decreasing.

Suppose n′ is successor of n, then

$$g(n') = g(n) + C(n, a, n')$$



We know that,

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

$$f(n') = g(n) + C(n, a, n') + h(n') \ldots (1)$$

A heuristic is consistent if

$$h(n) \leq C(n, a, n') + h(n') \ldots (2)$$

Now from (1) and (2)

$$f(n') = g(n) + C(n, a, n') + h(n') \geq g(n) + h(n) = f(n)$$

$$f(n') \geq f(n)$$

f(n) is non-decreasing along any path.

Hill Climbing Search

Hill climbing can be used to solve problems that have many solutions, some of which are better than others. It starts with a random (potentially poor) solution, and iteratively makes small changes to the solution, each time improving it a little. When the algorithm cannot see any improvement anymore, it terminates. Ideally, at that point the current solution is close to optimal, but it is not guaranteed that hill climbing will ever come close to the optimal solution.

For example, hill climbing can be applied to the traveling salesman problem. It is easy to find a solution that visits all the cities but will be very poor compared to the optimal solution. The algorithm starts with such a solution and makes small improvements to it, such as switching the order in which two cities are visited. Eventually, a much better route is obtained. In hill climbing the basic idea is to always head towards a state which is better than the current one. So, if you

are at town A and you can get to town B and town C (and your target is town D) then you should make a move IF town B or C appear nearer to town D than town A does.

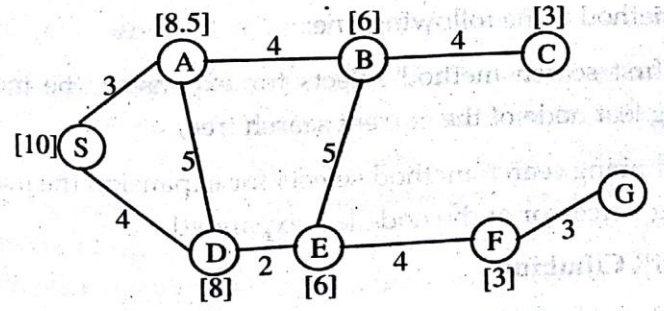
The hill climbing can be described as follows:

1. Evaluate the initial state, if it is goal state then Terminate. Otherwise, current state is initial state.
2. Select a new operator for this state and generate a new state.
3. Evaluate the new state
   - if it is closer to goal, make it current state.
   - if it is no better, ignore.
4. If the current state is goal state or no new operators available, Terminate. Otherwise repeat from 2.

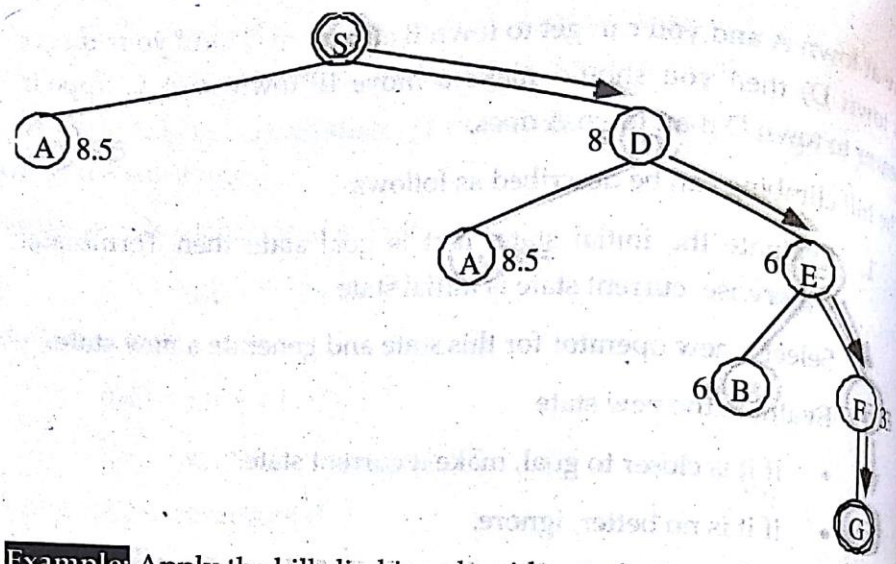
Hill climbing terminates when there are no successors of the current state which are better than the current state itself.

Hill climbing is depth-first search with a heuristic measurement that orders choices as nodes are expanded. It always selects the most promising successor of the node last expanded.

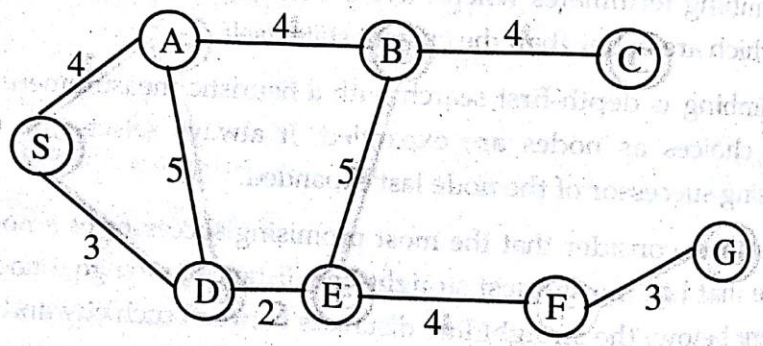
For instance, consider that the most promising successor of a node is the one that has the shortest straight-line distance to the goal node G. In figure below, the straight line distances between each city and goal G is indicated in square brackets, i.e. the heuristic.



The hill climbing search from S to G proceeds as follows:



Example: Apply the hill climbing algorithm to find a path from S to G, considering that the most promising successor of a node is its closest neighbor.



Note: The difference between the hill climbing search method and the best first search method is the following one:

- the best first search method selects for expansion the most promising leaf node of the current search tree;
- the hill climbing search method selects for expansion the most promising successor of the node last expanded.

Problems with Hill Climbing

- Gets stuck at local maxima when we reach a position where there are no better neighbors, it is not a guarantee that we have found the best solution.

- Ridge is a sequence of local maxima.
- Another type of problem we may find with hill climbing searches is finding a plateau. This is an area where the search space is flat so that all neighbors return the same evaluation

Simulated Annealing

It is motivated by the physical annealing process in which material is heated and slowly cooled into a uniform structure. Compared to hill climbing the main difference is that SA allows downwards steps. Simulated annealing also differs from hill climbing in that a move is selected at random and then decides whether to accept it. If the move is better than its current position then simulated annealing will always take it. If the move is worse (i.e. lesser quality) then it will be accepted based on some probability. The probability of accepting a worse state is given by the equation

P = exponential (-*c* / *t*) > r

Where

c	=	the change in the evaluation function
t	=	the current value
r	=	a random number between 0 and 1

The probability of accepting a worse state is a function of both the current value and the change in the cost function. The most common way of implementing an SA algorithm is to implement hill climbing with an accept function and modify it for SA

By analogy with this physical process, each step of the SA algorithm replaces the current solution by a random "nearby" solution, chosen with a probability that depends on the difference between the corresponding function values and on a global parameter T (called the temperature), that is gradually decreased during the process. The dependency is such that the current solution changes almost randomly when T is large, but increasingly "downhill" as T goes to zero. The allowance for "uphill" moves saves the method from becoming stuck at local optima—which are the bane of greedier methods. In short

- Permits "bad" moves to states with lower value, thus escape the local optima
- Gradually decreases the frequency of such moves and theirsize (parameter controlling it – temperature)

Game Playing

Game Playing is an important domain of artificial intelligence. Games don't require much knowledge; the only knowledge we need to provide is the rules, legal moves and the conditions of winning or losing the game. Both players try to win the game. So, both of them try to make the best move possible at each turn. Searching techniques like BFS (Breadth First Search) are not accurate for this as the branching factor is very high, so searching will take a lot of time. So, we need another search procedures that improve s–

- **Generate procedure** so that only good moves are generated.
- **Test procedure** so that the best move can be explored first.

Games are a form of multi-agent environment

- What do other agents do and how do they affect our success?
- Cooperative vs. competitive multi-agent environments.
- Competitive multi-agent environments give rise to adversarial search often known as games.

Adversarial Search Techniques

Adversarial search, also known as Minimax search is known for its usefulness in calculating the best move in two player games where all the information is available, such as chess or tic tac toe. It consists of navigating through a tree which captures all the possible moves in the game, where each move is represented in terms of loss and gain for one of the players. It follows that this can only be used to make decisions in zero-sum games, where one player's loss is the other player's gain. In these types of games there is always a set of strategies which leads to both players gaining the same value and that seeing as this is the best possible value one can expect to gain.

Games-adversary

- Solution is strategy (strategy specifies move for every possible opponent reply).
- Time limits force an approximate solution
- Evaluation function: evaluate "goodness" of game position
- Examples: chess, checkers, Othello, backgammon

Difference between the search space of a game and the search space of a problem: In the first case it represents the moves of two (or more) players, whereas in the latter case it represents the "moves" of a single problem-solving agent.

An exemplary game: Tic-tac-toe

There are two players denoted by X and O. They are alternatively writing their letter in one of the 9 cells of a 3 by 3 board. The winner is the one who succeeds in writing three letters in line. The game begins with an empty board. It ends in a win for one player and a loss for the other, or possibly in a draw.

A complete tree is a representation of all the possible plays of the game. The root node is the initial state, in which it is the first player's turn to move (the player X). The successors of the initial state are the states the player can reach in one move, their successors are the states resulting from the other player's possible replies, and so on. Terminal states are those representing a win for X, loss for X, or a draw.

Each path from the root node to a terminal node gives a different complete play of the game. Figure given below shows the initial search space of Tic-Tac-Toe.

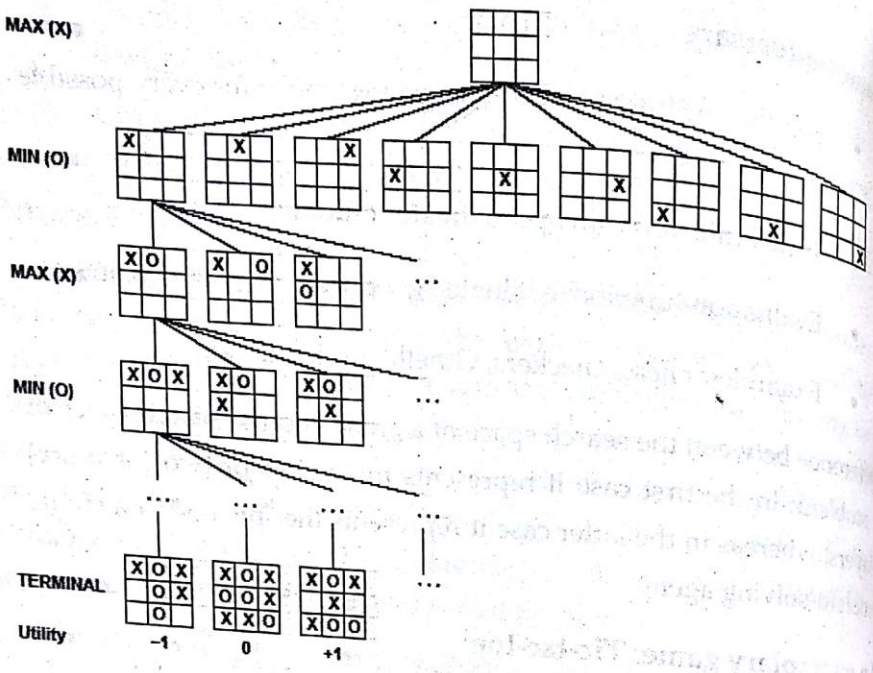


Figure: Partial game tree for Tic-Tac-Toe

A game can be formally defined as a kind of search problem as below:

- Initial state: It includes the board position and identifies the players to move.
- Successor function: It gives a list of (move, state) pairs each indicating a legal move and resulting state.
- Terminal test: This determines when the game is over. States where the game is ended are called terminal states.
- Utility function: It gives numerical value of terminal states. E.g. win (+1), loose (-1) and draw (0). Some games have a wider variety of possible outcomes eg. ranging from +92 to -192.

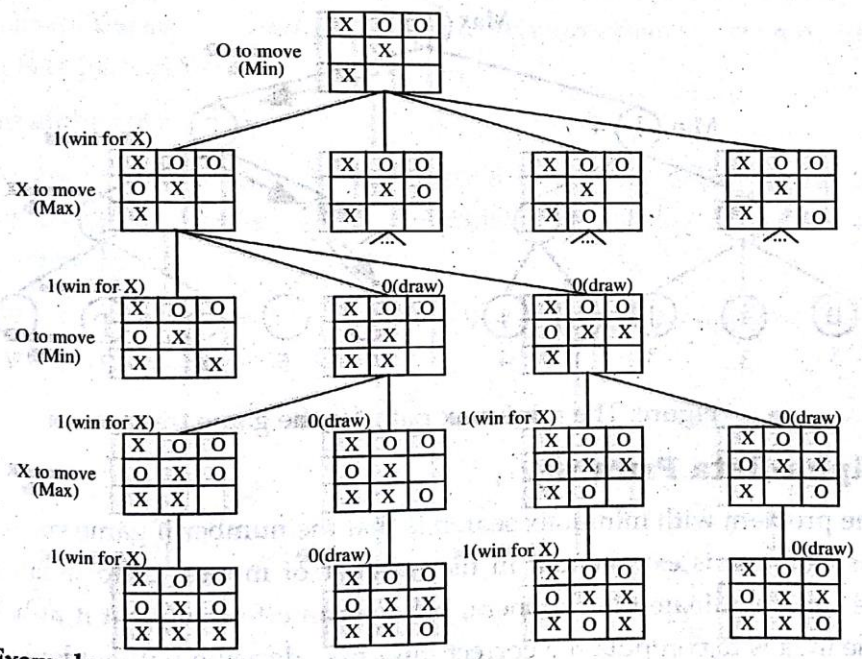
The Mini-max Search

Let us assign the following values for the game: 1 for win by X, 0 for draw, -1 for loss by X. Given the values of the terminal nodes (win for X (1), loss for X (-1), or draw (0)), the values of the non-terminal nodes are computed as follows:

- The value of a node where it is the turn of player X to move is the maximum of the values of its successors (because X tries to maximize its outcome);
- The value of a node where it is the turn of player O to move is the minimum of the values of its successors (because O tries to minimize the outcome of X).

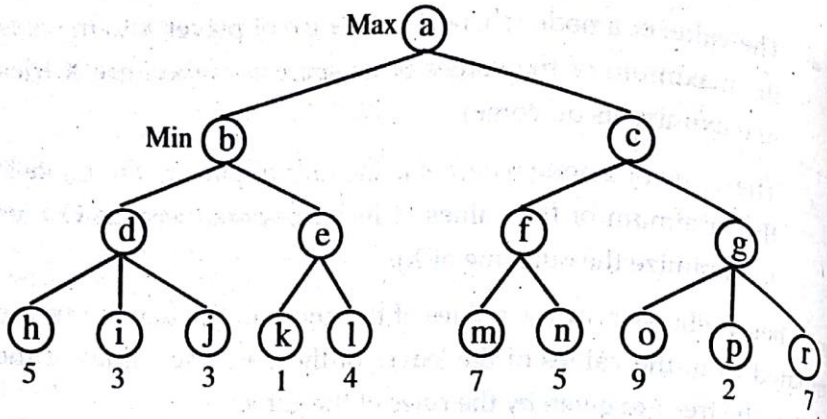
Figure below shows how the values of the nodes of the search tree are computed from the values of the leaves of the tree. The values of the leaves of the tree are given by the rules of the game:

- 1 if there are three X in a row, column or diagonal;
- -1 if there are three O in a row, column or diagonal;
- 0 otherwise



Example

Consider the following game tree (drawn from the point of view of the Maximizing player):



Show what moves should be chosen by the two players, assuming that both are using the mini-max procedure.

Solution:

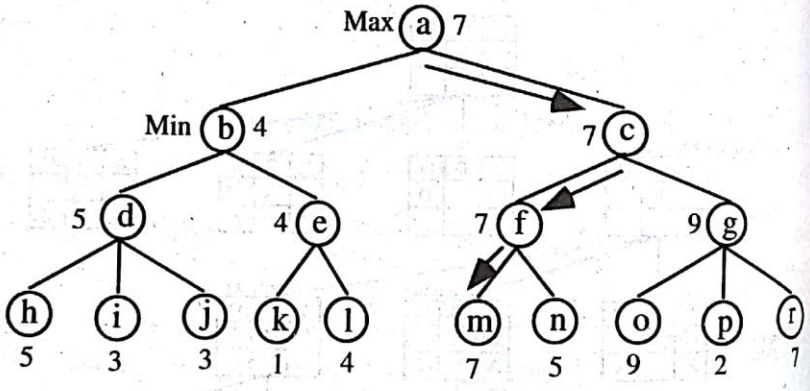


Figure: The mini-max path for the game tree

Alpha-Beta Pruning

The problem with mini-max search is that the number if game states it has examine is exponential in the number of moves. Unfortunately, we can't eliminate the exponent, but we can effectively cut it in half. The idea is to compute the correct mini-max decision without looking at every node in the game tree, which is the concept behind pruning. Here idea is to eliminate large parts of the tree from consideration. The particular technique for pruning that we will discuss here is "Alpha-Beta Pruning". When this approach is applied to a standard minimax tree, it returns the same move as minimax would, but

prunes away branches that cannot possibly influence the final decision. Alpha-beta pruning can be applied to trees of any depth, and it is often possible to prune entire sub-trees rather than just leaves.

Alpha-beta pruning is a technique for evaluating nodes of a game tree that eliminates unnecessary evaluations. It uses two parameters, alpha and beta.

Alpha: is the value of the best (i.e. highest value) choice we have found so far at any choice point along the path for MAX.

Beta: is the value of the best (i.e. lowest-value) choice we have found so far at any choice point along the path for MIN.

Alpha-beta search updates the values of alpha and beta as it goes along and prunes the remaining branches at a node as soon as the value of the current node is known to be worse than the current alpha or beta for MAX or MIN respectively.

An alpha cutoff

To apply this technique, one uses a parameter called alpha that represents a lower bound for the achievement of the Max player at a given node.

Let us consider that the current board situation corresponds to the node A in the following figure.

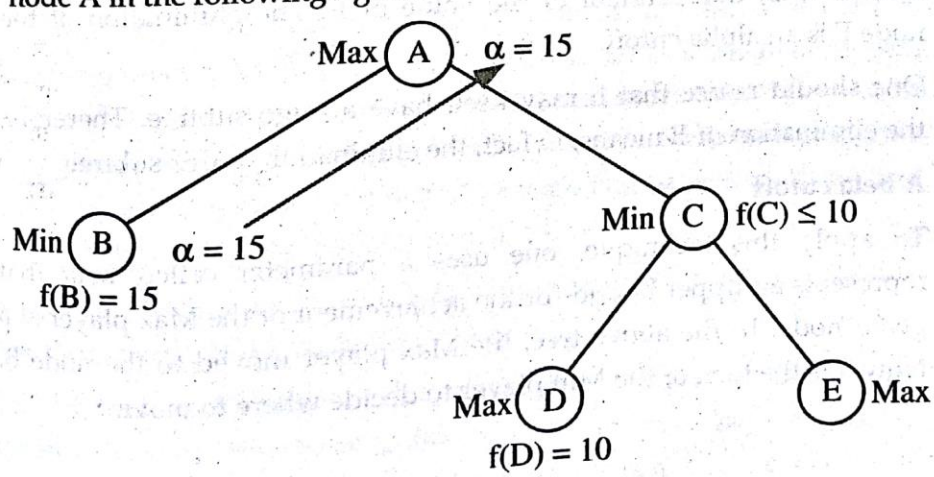


Figure: Illustration of the alpha cut-off.

The mini-max method uses a depth-first search strategy in evaluating the descendants of a node. It will therefore estimate first the value of the node B. Let us suppose that this value has been evaluated to 15, either by using a static evaluation function, or by backing up from descendants omitted in the figure. If Max will move to B then it is guaranteed to achieve 15. Therefore 15 is a lower bound for the achievement of the Max player (it may still be possible to achieve more, depending on the values of the other descendants of A). Therefore, the value of α at node B is 15. This value is transmitted upward to the node A and will be used for evaluating the other possible moves from A.

To evaluate the node C, its left-most child D has to be evaluated first. Let us assume that the value of D is 10 (this value has been obtained either by applying a static evaluation function directly to D, or by backing up values from descendants omitted in the figure). Because this value is less than the value of α, the best move for Max is to node B, independent of the value of node E that need not be evaluated. Indeed, if the value of E is greater than 10, Min will move to D which has the value 10 for Max. Otherwise, if the value of E is less than 10, Min will move to E which has a value less than 10. So, if Max moves to C, the best it can get is 10; which is less than the value α= 15 that would be gotten if Max would move to B. Therefore, the best move for Max is to B, independent of the value of E. The elimination of the node E is an alpha cutoff.

One should notice that E may itself have a huge subtree. Therefore, the elimination of E means, in fact, the elimination of this subtree.

A beta cutoff

To apply this technique, one uses a parameter called beta that represents an upper bound for the achievement of the Max player at a given node. In the above tree, the Max player moved to the node B. Now it is the turn of the Min player to decide where to move:

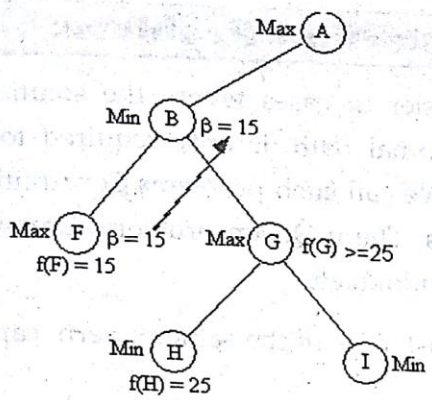


Figure: Illustration of the beta cut-off.

The Min player also evaluates its descendants in a depth-first order.

Let us assume that the value of F has been evaluated to 15. From the point of view of Min, this is an upper bound for the achievement of Min (it may still be possible to make Min achieve less, depending of the values of the other descendants of B). Therefore the value of β at the node F is 15. This value is transmitted upward to the node B and will be used for evaluating the other possible moves from B.

To evaluate the node G, its left-most child H is evaluated first. Let us assume that the value of H is 25 (this value has been obtained either by applying a static evaluation function directly to H, or by backing up values from descendants omitted in the figure). Because this value is greater than the value of β, the best move for Min is to node F, independent of the value of node I that need not be evaluated. Indeed, if the value of I is $v \geq 25$, then Max (in G) will move to I. Otherwise, if the value of I is less than 25, Max will move to H. So in both cases, the value obtained by Max is at least 25 which is greater than β(the best value obtained by Max if Min moves to F).

Therefore, the best move for Min is at F, independent of the value of I. The elimination of the node I is a beta cutoff.

One should notice that by applying alpha and beta cut-off, one obtains the same results as in the case of mini-max, but (in general) with less effort. This means that, in a given amount of time, one could search deeper in the game tree than in the case of mini-max.

Constraint Satisfaction Problems

Search can be made easier in cases where the solution instead of corresponding to an optimal path, is only required to satisfy local consistency conditions. We call such problems Constraint Satisfaction (CS) Problems. Examples: The n-Queen problem, Crossword puzzle, Map coloring problem, Sudoku etc.

All these examples are instances of the same pattern, captured by the following definition:

A Constraint Satisfaction Problem is characterized by:

- a set of variables {x1, x2, .., xn},
- for each variable xi a domain Di with the possible values for that variable, and a set of constraints, i.e. relations, that are assumed to hold between the values of the variables. [These relations can be given intentionally, i.e. as a formula, or extensionally, i.e. as a set, or procedurally, i.e. with an appropriate generating or recognizing function.]We will only consider constraints involving one or two variables.

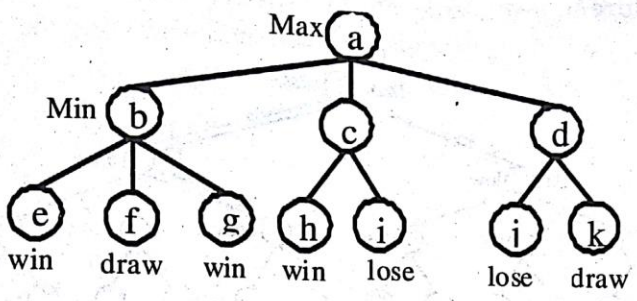
The constraint satisfaction problem is to find, for each i from 1 to n, a value in Di for xi so that all constraints are satisfied.

A CS problem can easily be stated as a sentence in first order logic, of the form:

(exist x1)..(exist xn) (D1(x1) & .. Dn(xn) =>(C1..Cm)

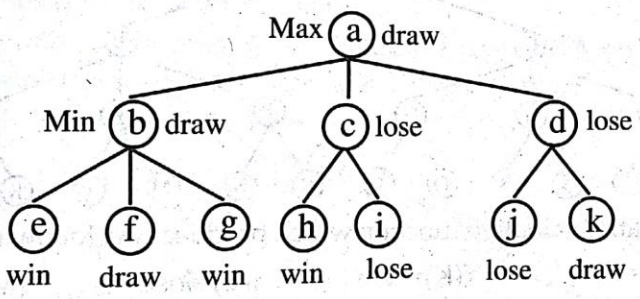
A CS problem is usually represented as an undirected graph, called Constraint Graph where the nodes are the variables and the edges are the binary constraints. Unary constraints can be disposed of by just redefining the domains to contain only the values that satisfy all the unary constraints. Higher order constraints are represented by hyper arcs. In the following we restrict our attention to the case of unary and binary constraints.

Example 1: Consider the following game tree (drawn from the point of view of the Maximizing player):



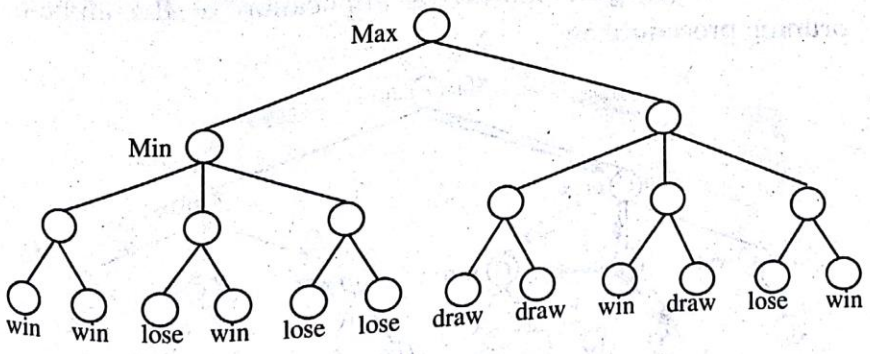
What move should be chosen by the Max player, and what should be the response of the Min player, assuming that both are using the mini-max procedure?

Solution:



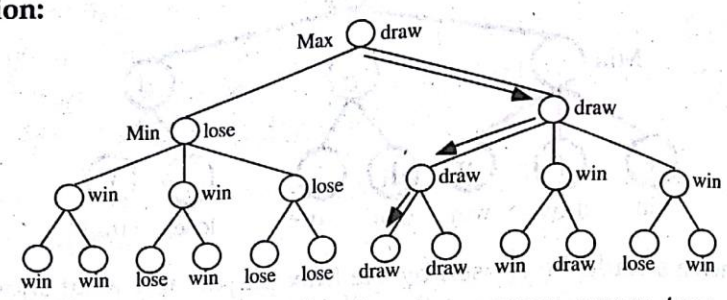
Max will move to b and min will respond by moving to f.

Example 2: Consider the following game tree (drown from the point of view of the Maximizing player):

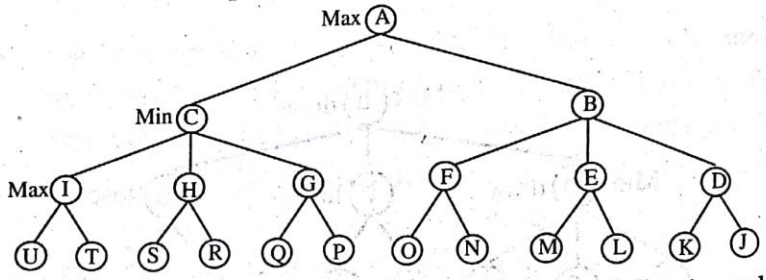


What move should be chosen by the Max player, and what should be the response of the Min player, assuming that both are using the mini-max procedure?

Solution:



Example 3: Consider the following two-person game tree, drown from the point of view of the Maximizing player:

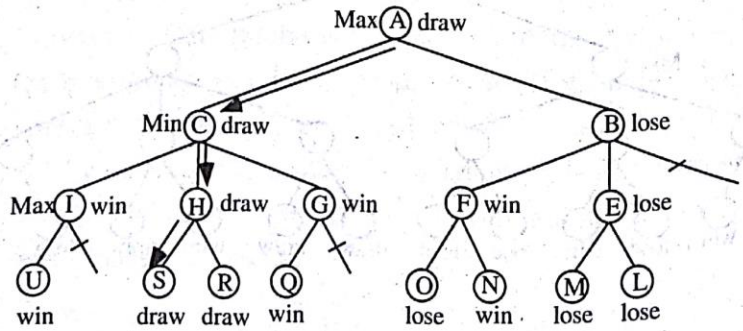


Assume a static evaluation function which produces the following values:

f(J)=win	f(K)=win	f(L)=lose	f(M)=lose
f(N)=win	f(O)=lose	f(P)=lose	f(Q)=win
f(R)=draw	f(S)=draw	f(T)=draw	f(U)=win

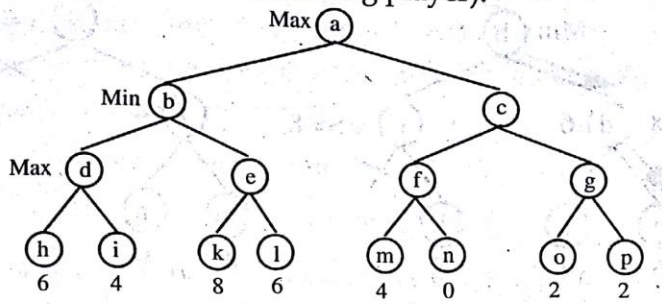
Use the alpha-beta pruning procedure to determine the moves which should be chosen by the two players, assuming a depth-first search (generation) of the tree. Which nodes need not to be examined (generated)?

a) The following figure shows the application of the alpha-beta pruning procedure on:



Max moves to C and Min moves to H and Max moves to either S or R. The game ends in a draw.

Example 4: Consider the following game tree (drown from the point of view of the Maximizing player):

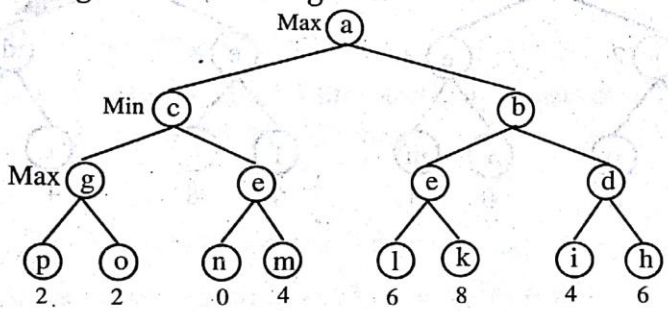


a) Use the mini-max procedure and show what moves should be chosen by the two players.

b) Use the alpha-beta pruning procedure and show what nodes would not need to be examined.

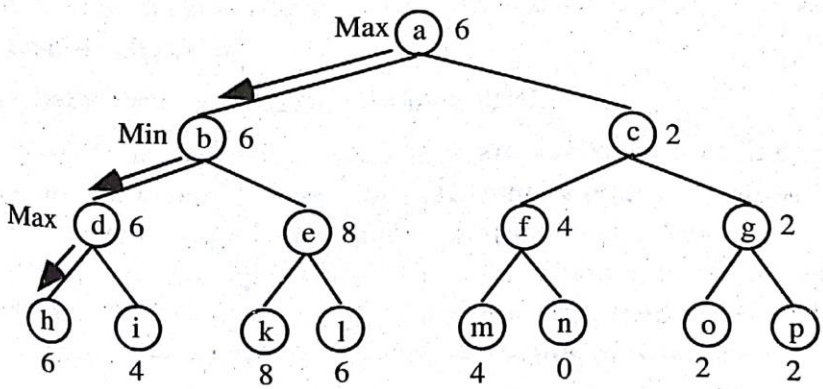
c) Consider the mirror image of the above tree and apply again the alpha-beta pruning procedure. What do you notice?

The mirror image is the following tree:

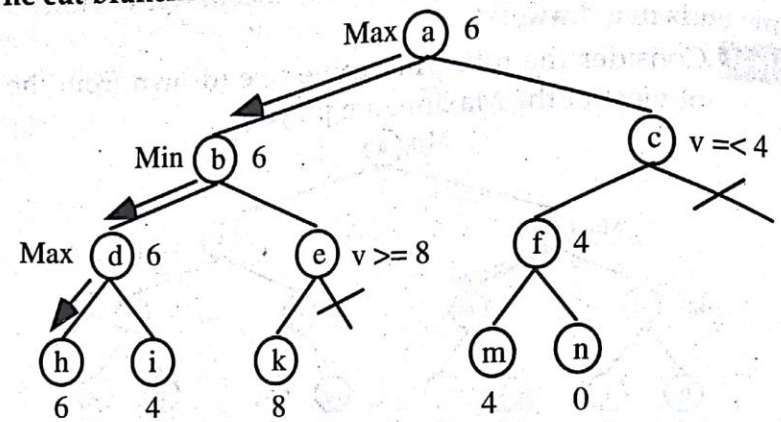


Solution:

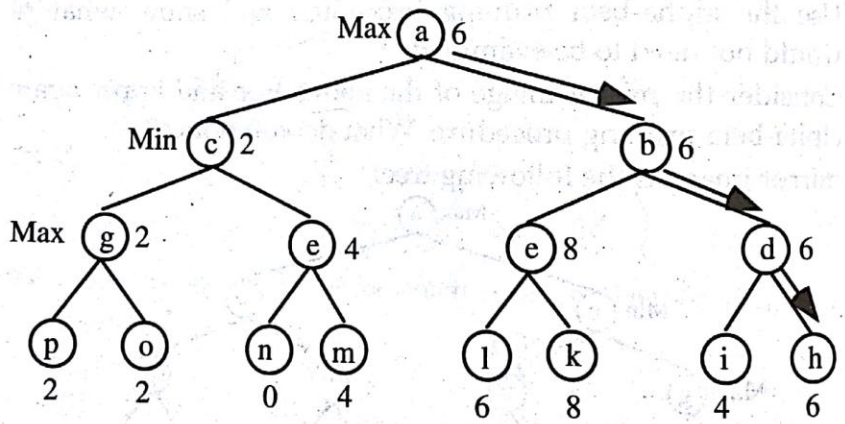
a) The mini-max procedure produce the following tree



b) **The cut branches need not be examined:**



c) Using alpha-beta pruning on the mirror image of the tree does not produce any savings:



Exercis

1. A farmer has to cross a river with his fox, goose and grain. Each trip, his boat can only carry himself and one of his possessions. How can he cross the river if an unguarded fox eats the goose and an unguarded goose the grain?
   - Find a good representation.
   - Perform Depth-first search (queues)
   - Perform Breadth-first search (search tree)
2. Which methods other than breadth-first can be used in bidirectional search?
3. Is it possible to replace breadth-first for either or both of the forward and backward direction?
4. Does the method still work if the check for the shared state is replaced by a check for identical end nodes?
5. Solve the water jugs problem
   - Given two jugs of 4 liter and 3 liter respectively, fill the 4 liter jug with 2 liter of water.
   - Find a good heuristic.
   - Perform Hill-climbing II Search.
6. What is meant by search algorithm completeness?
7. What is meant by search algorithm optimality?
8. What are the advantages of breadth-first search (BFS) over depth-first search (DFS)?
9. What is the advantage of DFS over BFS?
10. A 3-litre and a 5-litre water jugs are available, together with a source of water. The two jugs are initially empty. Your task is to fill one of the jugs with exactly one litre of water, leaving the other empty, by drawing the lowest possible amount of water from the source. To this aim you are only allowed to add water from the source to one of the jugs until the jug is full, or to empty one of the

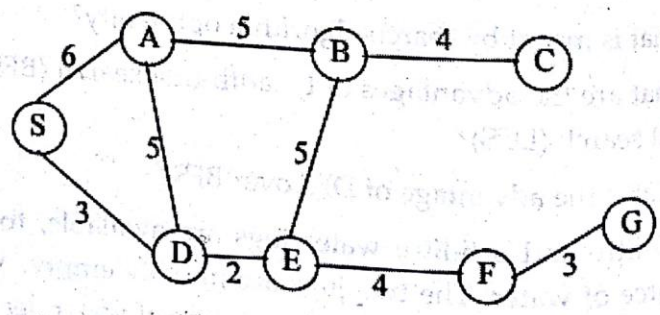
jugs by throwing away the water it contains, or to pour the water of one of the jugs into the other until the former is empty or the latter is full (whatever happens first).

a) Formulate the above task as a search problem, by precisely defining the state space, the initial state, the goal test, the set of possible actions and the path cost.

b) Solve the above problem using the uniform cost search strategy, avoiding repeated states: build the search tree until a solution is found, by clearly showing the order of expansion of each node, the actions associated to the edges of the tree and their cost, the state and the path cost of each node.

c) Discuss the possibility of using informed search strategies for this problem.

11. Discuss the tic-tac-toe problem in detail and explain how it can be solved using AI techniques?
12. Explain A* algorithm with an example. What are the limitations of A* algorithm?
13. Define Heuristic search? What are the advantages of Heuristic search?
14. Describe the heuristic search technique applied to a hill-climbing problem with an example?
15. What do you mean by heuristic search? Given following state space representation, show how greedy best first and A* search is used to find the goal state.



S is the start state and G is the goal state. The heuristics of the states are h(S) = 12, h(A)= 8, h(D)= 9, h(B)= 7, h(D)= 6, h(E)= 4, h(C)= 5 , h(F)= 2, h(G)= 0.

Lab Manual

LAB No 1:

Depth First Search (DFS) Program in C [Adjacency Matrix]

```
#include<stdio.h>
void DFS(int);
int G[10][10],visited[10],n;    //n is no of vertices and graph is sorted
in array G[10][10]
void main()
{
  int i,j;
  printf("Enter number of vertices:");
  scanf("%d",&n);
   //read the adjecency matrix
  printf("\nEnter adjecency matrix of the graph:");
 for(i=0;i<n;i++)
        for(j=0;j<n;j++)
                scanf("%d",&G[i][j]);
  //visited is initialized to zero
 for(i=0;i<n;i++)
    visited[i]=0;
  DFS(0);
}
void DFS(int i)
{
  int j;
  printf("\n%d",i);
  visited[i]=1;
  for(j=0;j<n;j++)
    if(!visited[j]&&G[i][j]==1)
       DFS(j);
}
```

LAB No 2:

Breadth First Search (BFS) Program in C

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 100
#define initial 1
#define waiting 2
#define visited 3
int n;
int adj[MAX][MAX];
int state[MAX];
void create_graph();
void BF_Traversal();
void BFS(int v);
int queue[MAX], front = -1,rear = -1;
void insert_queue(int vertex);
int delete_queue();
int isEmpty_queue();
int main()
{
        create_graph();
        BF_Traversal();
        return 0;
}
void BF_Traversal()
{
        int v;
        for(v=0; v<n; v++)
                state[v] = initial;
        printf("Enter Start Vertex for BFS: \n");
        scanf("%d", &v);
        BFS(v);
}
void BFS(int v)
{
```

```
        int i;
        insert_queue(v);
        state[v] = waiting;
        while(!isEmpty_queue())
        {
                v = delete_queue( );
                printf("%d ",v);
                state[v] = visited;
                for(i=0; i<n; i++)
                {
                        if(adj[v][i] == 1 && state[i] == initial)
                        {
                                insert_queue(i);
                                state[i] = waiting;
                        }
                }
        }
        printf("\n");
}
void insert_queue(int vertex)
{
        if(rear == MAX-1)
                printf("Queue Overflow\n");
        else
        {
                if(front == -1)
                        front = 0;
                rear = rear+1;
                queue[rear] = vertex ;
        }
}
int isEmpty_queue()
{
        if(front == -1 || front > rear)
                return 1;
        else
```

```
                return 0;
}
int delete_queue()
{
        int delete_item;
        if(front == -1 || front > rear)
        {
                printf("Queue Underflow\n");
                exit(1);
        }
        delete_item = queue[front];
        front = front+1;
        return delete_item;
}
```

LAB No 3:

Best First Search Algorithm

```
Best-First-Search (Grah g, Node start)
  1) Create an empty PriorityQueue
    PriorityQueue pq;
  2) Insert "start" in pq.
    pq.insert(start)
  3) Until PriorityQueue is empty
      u = PriorityQueue.DeleteMin
      If u is the goal
        Exit
      Else
        Foreach neighbor v of u
          If v "Unvisited"
            Mark v "Visited"
            pq.insert(v)
        Mark u "Examined"
End procedure
```

✿✿✿

UNIT 4

Knowledge Representation

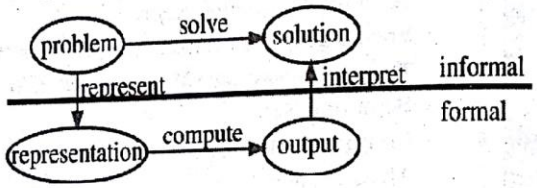
OUTLINE

- ✓ **Definition and Importance of Knowledge**
  - Issue in Knowledge Representations
  - Knowledge Representation System and its Properties
- ✓ **Types of Knowledge Representation schemes**
  - Semantic Nets
  - Components of a Semantic Network
  - AND/OR tree
  - Frame based Knowledge Representations
  - Conceptual Dependencies
  - Script Structure
  - Rule Based Knowledge Representation System (RBS)
  - Logic Based Knowledge Representations
  - Logical Inference
  - First-Order Logic
  - Bayes Theorem
- ✓ **Fuzzy Logic**
  - Fuzzification
  - Defuzzification

Definition and Importance of Knowledge

Knowledge is what we know as facts. The facts are declarative sentences that are either true or false. In case of problem solving by computer, Knowledge is the information about a particular domain that can be used to solve problems in that domain.

Knowledge is important in problem solving by computer because to solve a problem by a computer, it first must be represented in computer understandable form. The general framework for solving problems by computer can be given as shown in figure below:



To solve problems, it requires full of knowledge about the problem domain, and this knowledge must be represented in the computer. As part of designing a program to solve problems, we must define how the knowledge will be represented in computer understandable form.

The process of converting informal knowledge into computer understandable form is known as knowledge representation scheme. A representation scheme specifies the form of the knowledge. A knowledge base is the representation of all of the knowledge that is stored by an agent.

Issue in Knowledge Representations

Some of the issue regarding knowledge representation can be formulated as questions as follows:

- How can the problem be represented?
- What distinctions in the world are needed to solve the problem?
- What specific knowledge about the world is required?
- How can an agent acquire the knowledge from experts or from experience?
- How can the knowledge be debugged, maintained, and improved?

In summary, the main issue in knowledge representations are:

Selection of Important Attributes

How to find the important attribute form the bank of information about an object or world.

Relationships

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

Granularity

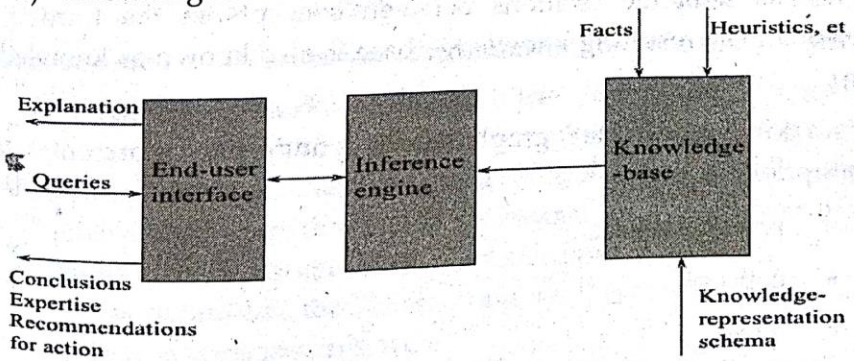
At what level should the knowledge be represented and what are the primitives. Choosing the Granularity of Representation Primitives are fundamental concepts such as holding, seeing, playing and as English is a very rich language with over half a million words it is clear we will find difficulty in deciding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

Knowledge Representation System and its Properties

Knowledge representation system is a formalized structure and sets of operations that embodies the descriptions, relationships, and procedures provided in an artificial Intelligence system. The process considers how to represent the knowledge in the system such that it can be used to solve problems.

It basically consists of two components:

1) Knowledge base 2) Inference Methods



A good knowledge representation system should consist of following properties.

Representational adequacy

The scheme is rich enough to express the knowledge needed to solve the problem. In other words, the ability to represent all of the kinds of knowledge that are needed in a certain domain.

Inferential adequacy

The ability to represent all of the kinds of inferential procedures (procedures that manipulate the representational structures in such a way as to derive new structures corresponding to new knowledge inferred from old).

Inferential efficiency

The ability to represent efficient inference procedures (for instance, by incorporating into the knowledge structure additional information that can be used to focus the attention of the inference mechanisms in the most promising directions).

Acquisitional efficiency

The ability to acquire new knowledge using automatic methods wherever possible rather than reliance on human intervention.

Types of Knowledge Representation schemes

Semantic Nets

It is a directed or undirected graph consisting of vertices, which represent concepts, and edges, which represent semantic relations between concepts. It is a method of knowledge representation that represents semantic relations between concepts in the form of a network. The resulting knowledge base is also known as knowledge map.

It is a kind of directed graph where a node may represent a fact description such as

- physical object
- concept
- event

And an arc (or link) represents relationships between nodes. There are some 'standard' relationship types

- 'Is-a' (Class/instance relationship): represent class/instance relationships
- 'Has-a' (part-subpart relationship):identify property relationships

Components of a Semantic Network

The fundamental components of semantic networks are straightforward to identify:

Lexical part

- nodes – denoting objects
- links – denoting relations between objects
- abels – denoting particular objects and relations

Structural part

- the links and nodes form directed graphs the labels are placed on the links and nodes

Semantic part

- meanings are associated with the link and node labels (the details will depend on the application domain)

Procedural part

- constructors allow creation of new links and nodes destructors allow the deletion of links and nodes writers allow the creation and alteration of labels
- readers can extract answers to questions

Types of Semantic Network

Following are six of the most common kinds of semantic networks:

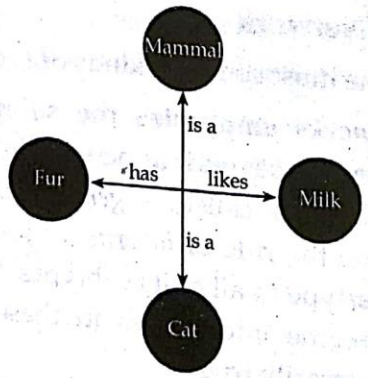
- ***Definitional networks*** **emphasize the** ***subtype*** or *is-a* relation between a concept type and a newly defined subtype. The resulting network, also called a *generalization* or *subsumption* hierarchy, supports the rule of *inheritance* for copying properties defined for a supertype to all of its subtypes. Since definitions are true by definition, the information in these networks is often assumed to be necessarily true.
- ***Assertional networks*** are designed to assert propositions. Unlike definitional networks, the information in an assertional network is

assumed to be contingently true, unless it is explicitly marked with a modal operator. Some assertionalnetworks have been proposed as models of the *conceptual structures* underlying natural language semantics.

- ***Implicational networks*** use implication as the primary relation for connecting nodes. They may be used to represent patterns of beliefs, causality, or inferences.
- ***Executable networks*** include some mechanism, such as marker passing or attached procedures, which can perform inferences, pass messages, or search for patterns and associations.
- ***Learning networks*** build or extend their representations by acquiring knowledge from examples. The new knowledge may change the old network by adding and deleting nodes and arcs or by modifying numerical values, called *weights*, associated with the nodes and arcs.
- ***Hybrid networks*** combine two or more of the previous techniques, either in a single network or in separate, but closely interacting networks.

Examples of Semantic Network: The following statement/ information are represented in the figure below:

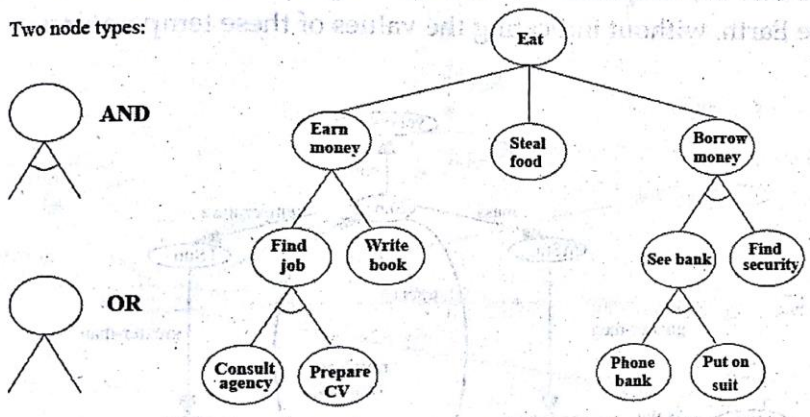
- cat is a mammal
- cat likes milk
- cat has fur
- tommy is a cat



Other examples of Semantic network can be consider as AND/OR tree, Is_a Hierarchy, Is_Part Hierarchy etc.

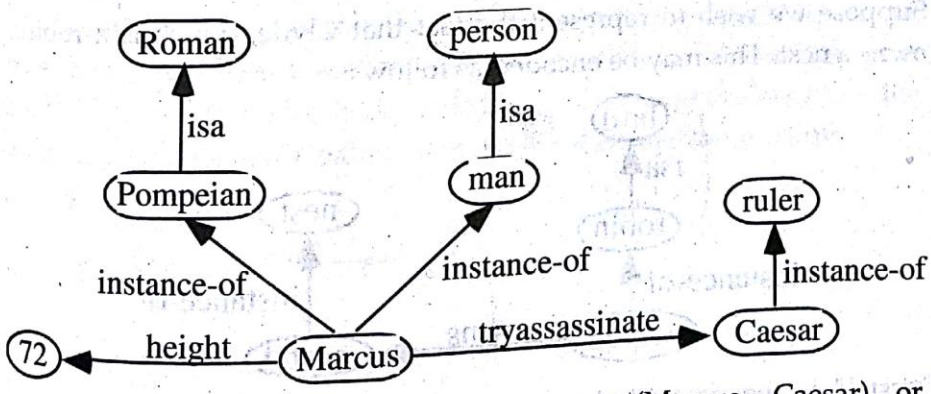
AND/OR tree

- An AND/OR tree is a tree (or, just as frequently, a DAG) whose internal nodes are labelled either "AND" or "OR". A *valuation* of an AND/OR tree is an assignment of "TRUE" or "FALSE" to each of the leaves.
- Given a tree T and a valuation over the leaves of T, the values of the internal nodes and of T are defined recursively in the obvious way:
  - ✓ An OR node is TRUE if at least one of its children is TRUE.
  - ✓ An AND node is TRUE if all of its children are TRUE



Semantic networks with binary relations

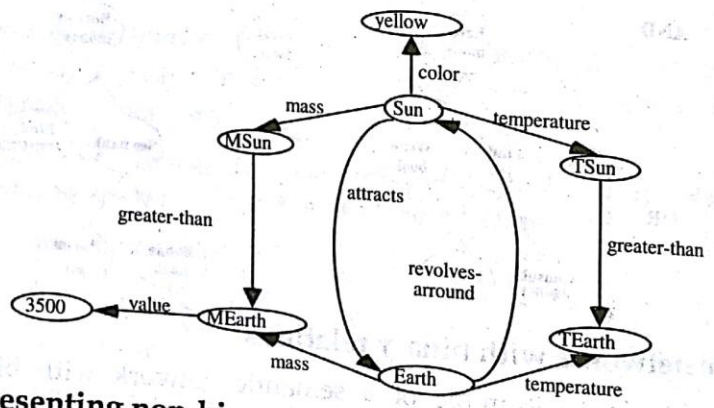
The following is an example of a semantic network with binary relations:



Some nodes represent individual objects (Marcus, Caesar) or constants (72).Other nodes represent classes of individuals (Pompeian,

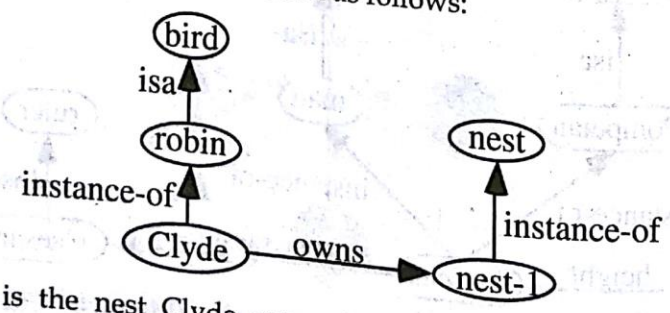
Roman, man, person, ruler). The relationship "instance-of" asserts that an individual is an instance (element) of a class. The relationship "isa" asserts that a class, (e.g. "man"), is a subclass of another class ("person"). "Man isa person" means that any element of man is also an element of person. That is, if someone is a man then he is a person. The relationship "tryassassinate", from Marcus to Caesar, states that Marcus tried to assassinate Caesar. The relationship "height", from Marcus to 72, states that the height of Marcus is 72. Marcus and Caesar are called instances because they represent individual objects.

The following is another example of a semantic network representing knowledge about our solar system. One may notice that this network states that the temperature of the Sun is greater than the temperature of the Earth, without indicating the values of these temperatures:



Representing non-binary predicates

Suppose we wish to represent the fact that Clyde, which is a robin, owns a nest. This may be encoded as follows:



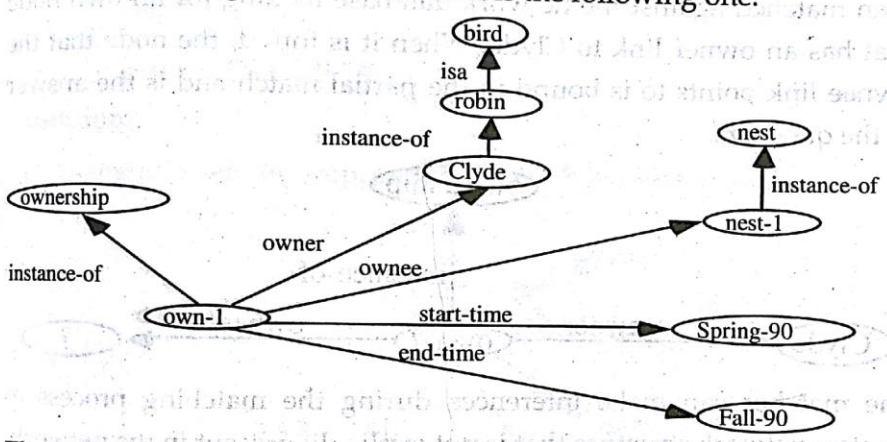
"nest-1" is the nest Clyde owns. It is an instance of "nest" which represents the class of all nests. Suppose one wants to encode the

additional information that Clyde owned nest-1 from Spring 90 to Fall 90. This is impossible to do in the above network because the ownership situation is encoded as a link, and links, by their nature, can encode only binary relations.

The solution is to represent this ownership situation as a node characterized by four properties: the owner (Clyde), the object owned (nest-1), the time when the ownership started (Spring 90), and the time when the ownership ended (Fall 90). In this way one may represent the equivalent of a four-place predicate:

ownership(owner, ownee, start-time, end-time)

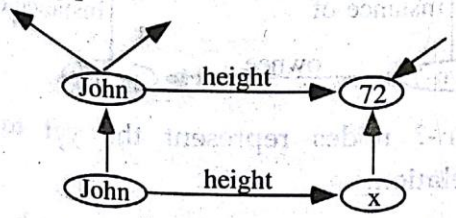
The corresponding semantic network is the following one:



Therefore, the semantic networks are not limited to representing only binary relationships

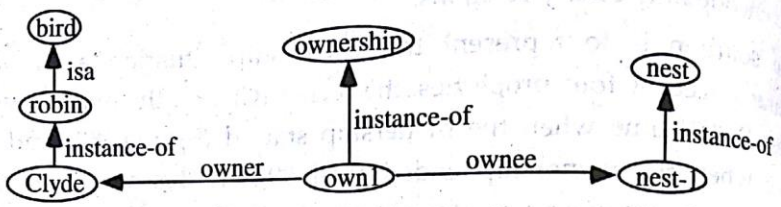
Network Match in Graph

A network fragment is constructed, representing a sought-for object or a query, and then matched against the network database to see if such an object exists. Variable nodes in the fragment are bound in the matching process to the values with which they match perfectly.

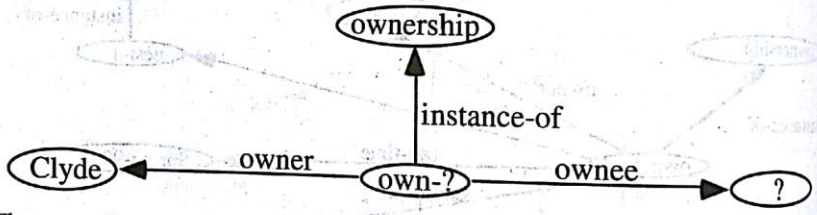


Question:
What is the height of John ?
Answer:
The height of John is 72.

As another example, let us suppose that we use the network given below as a database and suppose we wish to answer the question "What does Clyde own?".



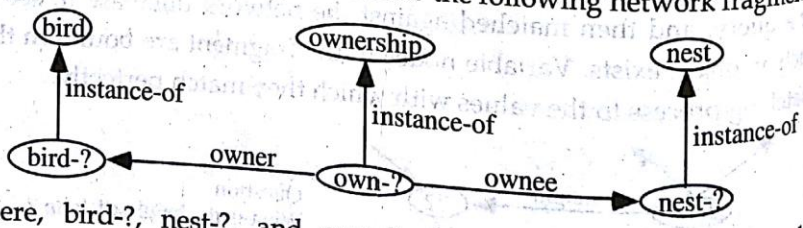
We might construct the network in above figure which represents an instance of ownership in which Clyde is the owner. This fragment is then matched against the network database looking for an own node that has an owner link to Clyde. When it is found, the node that the ownee link points to is bound in the partial match and is the answer to the question.



The matcher can make inferences during the matching process to create a network structure that is not explicitly present in the network. For example, suppose we wish to answer the question

"Is there a bird who owns a nest?"

We could translate that question into the following network fragment:



Here, bird-?, nest-?, and own-? nodes represent the yet to be determined bird-owning-nest relation.

Notice that the query network fragment does not match the knowledge base exactly. The deduction procedure would have to

construct an "instance-of" link from Clyde to bird to make the match possible. The matcher would bind bird-? to the node Clyde, own-? to own-1, and nest-? to nest-1, and the answer to the question would be "Yes, Clyde".

Solved Examples

1. Represent the following sentences into a semantic network.

   Birds are animals.

   Birds have feathers, fly and lay eggs.

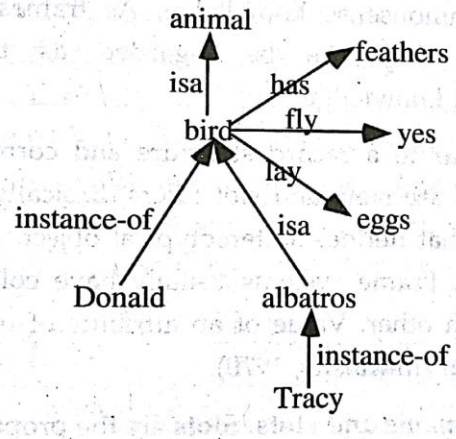
   Albatros is a bird.

   Donald is a bird.

   Tracy is an albatros.

Solution:

The Semantic network representation of above sentences is:



2. Represent the following sentences into a semantic network:

   Puss is a calico.

   Herb is a tuna.

   Charlie is a tuna.

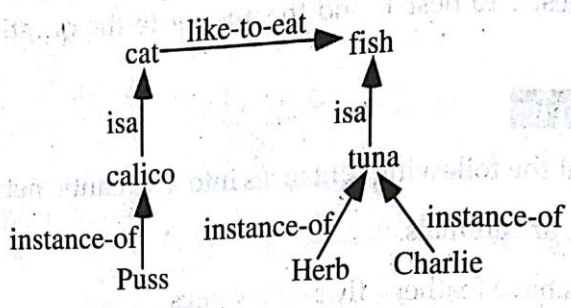
   All tunas are fishes.

   All calicos are cats.

   All cats like to eat all kinds of fishes.

Solution:

The Semantic network representation of above sentences is:



Frame based Knowledge Representations

Frame is a type of schema used in many AI applications including vision and natural language processing. Frames provide a convenient structure for representing objects that are typical to a stereotypical situations. The situations to represent may be visual scenes, structure of complex physical objects, etc. Frames are also useful for representing commonsense knowledge. As frames allow nodes to have structures they can be regarded as three-dimensional representations of knowledge.

A frame is similar to a record structure and corresponding to the fields and values are slots and slot fillers. Basically it is a group of slots and fillers that defines a stereotypical object. A single frame is not much useful. Frame systems usually have collection of frames connected to each other. Value of an attribute of one frame may be another frame(MarvinMinsky, 1970).

Each frame has **a name and slots. Slots** are the properties of the entity that has the name, and they have **values** or pointer to other frames (a table like data structure). A particular value may be:

- a default value
- an inherited value from a higher frame
- a procedure, called a daemon, to find a value
- a specific value, which might represent an exception.

When the slots of a frame are all filled, the frame is said to be **instantiated**, it now represents a specific entity of the type defined by the unfilled frame. Empty frames are sometimes called object prototypes

The idea of frame hierarchies is very similar to the idea of class hierarchies found in object-orientated programming. Frame are an application of the object-oriented approach to knowledge-based systems.

Examples:

A frame for a book is given below:

Slots	Fillers
publisher	Thomson
title	Expert Systems
author	Giarratano
edition	Third
year	1998
pages	600

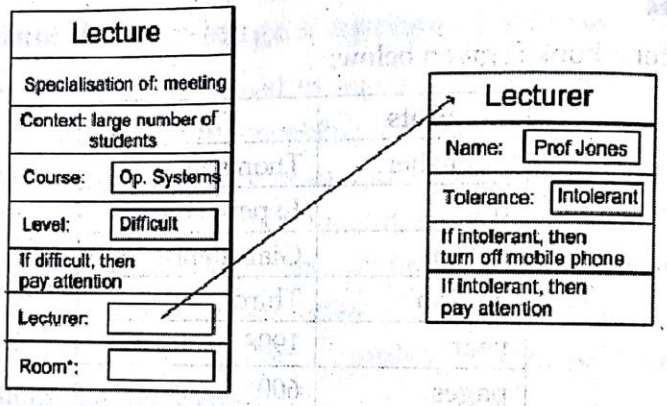
The above example is simple one but most of the frames are complex. Moreover with filler slots and inheritance provided by frames powerful knowledge representation systems can be built. Frames can represent either generic or frame. Following is the example for generic frame.

Slot	Fillers
name	computer
specialization_of	a_kind_of machine
types	(desktop, laptop, mainframe, super) if-added: Procedure ADD_COMPUTER
speed	default: faster if-needed: Procedure FIND_SPEED
location	(home, office, mobile)
under_warranty	(yes, no)

The fillers may values such as computer in the name slot or a range of values as in types slot. The procedures attached to the slots are called

procedural attachments. There are mainly three types of procedural attachments: if-needed, default and if-added. As the name implies if-needed types of procedures will be executed when a filler value is needed. Default value is taken if no other value exists. Defaults are used to represent *commonsense knowledge*. Commonsense is generally used when no more situation specific knowledge is available.

The following figure show the two frame: Lecture and Lecturer, connected with a link.



Conceptual Dependencies

CD theory was developed by Schank in 1973 to 1975 to represent the meaning of Natural Language sentences.CD representation of a sentence is not built using words in the sentence rather built using conceptual primitives which give the intended meanings of words. Since it represent meaning of sentences using primitives but not the exact words in a sentences:

- It helps in drawing inferences
- It is independent of the language

CD provides structures and specific set of primitives from which representation can be built.

CD primitives: There are following standard CD primitives:

- ATRANS Transfer of an abstract relationship (i.e. give)
- PTRANS Transfer of the physical location of an object (e.g., go)

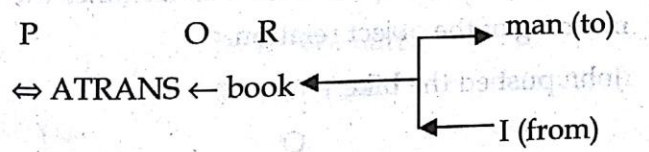
- PROPEL Application of physical force to an object (e.g. push)
- MOVE Movement of a body part by its owner (e.g. kick)
- GRASP Grasping of an object by an action (e.g. throw)
- INGEST Ingesting of an object by an animal (e.g. eat)
- EXPEL Expulsion of something from the body of an animal (e.g. cry)
- MTRANS Transfer of mental information (e.g. tell)
- MBUILD Building new information out of old (e.g decide)
- SPEAK Producing of sounds (e.g. say)
- ATTEND Focusing of a sense organ toward a stimulus (e.g. listen)

There are four conceptual categories

- ACT Actions {one of the CD primitives}
- PP Objects {picture producers}
- AA Modifiers of actions {action aiders}
- PA Modifiers of PP's {picture aiders}

Example

I gave a book to the man. CD representation is as follows:



It should be noted that this representation is same for different saying with same meaning. For example

- I gave the man a book,
- The man got book from me,
- The book was given to man by me etc.

C.) Notations

- Arrows indicate directions of dependency

- Double arrow indicates two way link between actor and action.

  O – for the object case relation

  R – for the recipient case relation

  P – for past tense

  D – destination

Rule 1: PP ⇔ ACT

It describes the relationship between an actor and the event he or she causes.

- This is a two-way dependency, since neither actor nor event can be considered primary.
- The letter P in the dependency link indicates past tense.

Example: John ran

CD Rep: John ⇔ PTRANS (P)

Rule 2: ACT ← PP

It describes the relationship between a ACT and a PP (object) of ACT.

- The direction of the arrow is toward the ACT since the context of the specific ACT determines the meaning of the object relation.

Example: John pushed the bike

CD Rep: John ⇔ PROPEL ← bike (O)

Rule 3: PP ↔ PP

It describes the relationship between two PP's, one of which belongs to the set defined by the other.

Example: John is doctor

CD Rep: John ↔ doctor

Rule 4: PP ← PP

It describes the relationship between two PP's, one of which provides a particular kind of information about the other.

- The three most common types of information to be provided in this way are possession (shown as POSS-BY), location (shown as LOC), and physical containment (shown as CONT).
- The direction of the arrow is again toward the concept being described.

Example: John's dog

CD Rep dog $\xleftarrow{\text{poss-by}}$ John

Rule 5: PP ⇔ PA

It describes the relationship between a PP and a PA that is asserted to describe it.

- PA represents states of PP such as height, health etc.

Example: John is fat

CD Rep John ⇔ weight (> 80)

Rule 6: PP ← PA

It describes the relationship between a PP and an attribute that already has been predicated of it.

- Direction is towards PP being described.
- Example: Smart John

CD Rep John ← smart

Rule 7: ACT ←

R → PP (to)

← PP (from)

It describes the relationship between an ACT and the source and the recipient of the ACT

Example: John took the book from Mary

CD Rep: John ⇔ ATRANS ← R → John

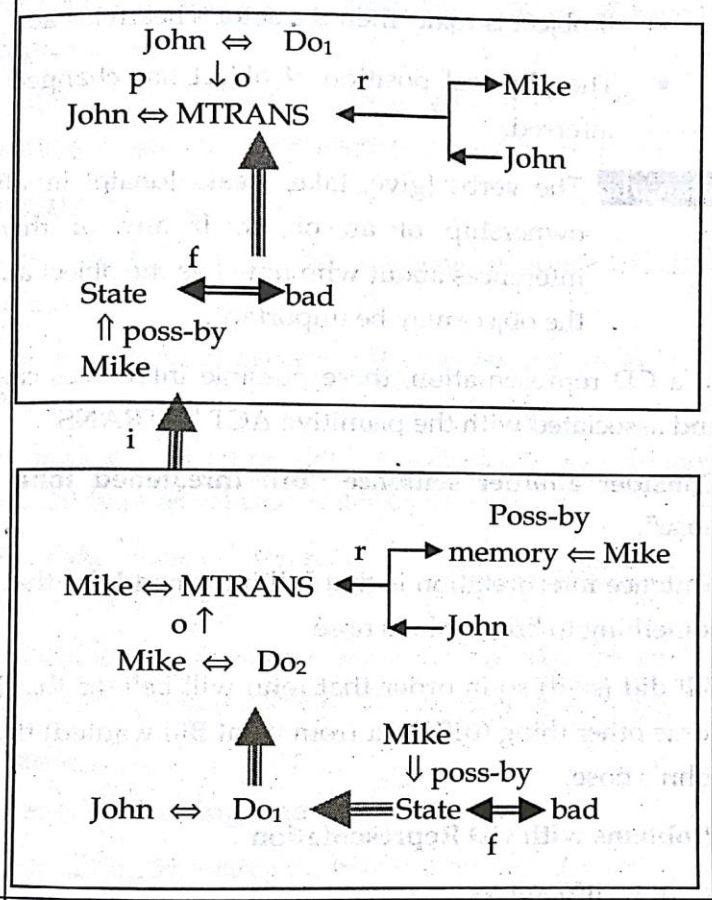
O ↑← Mary

book

Generation of CD representations: Examples

Sentences	CD Representations
Jenny cried	Jenny ⇔ EXPEL (p) ← tears (o); EXPEL → ? (d); tears ← eyes; eyes poss-by ⇑ Jenny
Mike went to India	Mike ⇔ PTRANS (p); PTRANS → India (d); PTRANS ← ? (source is unknown)
Mary read a novel	Mary ⇔ MTRANS (p) ← info (o); MTRANS → CP(Mary) (d); info ← novel; ↑ i (instrument) Mary ⇔ ATTEND (p) ← eyes (o); ATTEND → novel (d); eyes ← ?

Sentence	CD Representation
Since drugs can kill, I stopped.	One ⇔ INGEST (o) ← durgs (o); INGEST → One (r); durgs ← Mouth; c ⇑ One → health = -10; One → health > -10; c ⇑ I ⇔ INGEST (t_{fp}) ← durgs (o); INGEST → I (r); durgs ← mouth

Sentence	CD Representation
John warned Mike with dire consequence.	

Inferences Associated with Primitive Act

General inferences are stored with each primitive Act thus reducing the number of inferences that need to be stored explicitly with each concept.

For example, from a sentence "John killed Mike", we can infer that "Mike is dead".

Let us take another example of primitive Act INGEST.

The following inferences can be associated with it.

- The object ingested is no longer available in its original form.
- If object is eatable, then the actor has less hunger.
- If object is toxic, then the actor's heath is bad.
- The physical position of object has changed. So PTRANS is inferred.

Example: The verbs {give, take, steal, donate} involve a transfer of ownership of an object. If any of them occurs, then inferences about who now has the object and who once had the object may be important.

In a CD representation, these possible inferences can be stated once and associated with the primitive ACT "ATRANS".

Consider another sentence "Bill threatened John with a broken nose"

Sentence interpretation is that Bill informed John that he (Bill) will do something to break john's nose.

Bill did (said) so in order that John will believe that if he (john) does some other thing (different from what Bill wanted) then Bill will break John's nose.

Problems with CD Representation

1. It is difficult to
   - construct original sentence from its corresponding CD representation.
   - CD representation can be used as a general model for knowledge representation, because this theory is based on representation of events as well as all the information related to events.
2. Rules are to be carefully designed for each primitive action in order to obtain semantically correct interpretation.

3. Many verbs may fall under different primitive ACTs, and it becomes difficult to find correct primitive in the given context.
4. The CD representation becomes complex requiring lot of storage for many simple actions.
5. For example, the sentence "John bet Mike that Indian cricket team will win incoming world cup" will require huge CD structure.

Script Structure

Scripts were introduced by Schank and Abelson introduced in 1977 that used CD framework.

The scripts are useful in describing certain stereotyped situations such as going to theater

It consists of set of slots containing default values along with some information about the type of values similar to frames.

It differs from FS as the values of the slots in scripts must be ordered and have more specialized roles.

In real world situations, we see that event tends to occur in known patterns because of clausal relationship to the occurrence of events

Script Components

Each script contains the following main components.

- ***Entry Conditions*:** Must be satisfied before events in the script can occur.
- ***Results:*** Conditions that will be true after events in script occur.
- ***Props:*** Slots representing objects involved in the events.
- ***Roles:*** Persons involved in the events.
- ***Track:*** Specific variation on more general pattern in the script. Different tracks may share many components of the same script but not all.
- ***Scenes:*** The sequence of *events* that occur. Events are represented in conceptual dependency form.

Examples

Script : Play in theater	Various Scenes
Track: Play in Theater **Props:** • Tickets • Seat • Play **Roles:** • Person (who wants to see a play) – P • Ticket distributor – TD • Ticket checker – TC **Entry Conditions:** • P wants to see a play • P has a money **Results:** • P saw a play • P has less money • P is happy (optional if he liked the play)	***Scene 1: Going to theater*** • P PTRANS P into theater • P ATTEND eyes to ticket counter ***Scene 2: Buying ticket*** • P PTRANS P to ticket counter • P MTRANS (need a ticket) to TD • TD ATRANS ticket to P ***Scene 3: Going inside hall of theater and sitting on a seat*** • P PTRANS P into Hall of theater • TC ATTEND eyes on ticket POSS by P • TC MTRANS (showed seat) to P • P PTRANS P to seat • P MOVES P to sitting position ***Scene 4: Watching a play*** • P ATTEND eyes on play • P MBUILD (good moments) from play ***Scene5: Exiting*** • P PTRANS P out of Hall and theater

Script Invocation

It must be activated based on its significance.

If the topic is important, then the script should be opened.

If a topic is just mentioned, then a pointer to that script could be held.

For example, given "John enjoyed the play in theater", a script "Play in Theater" suggested above is invoked.

All implicit questions can be answered correctly.

Here the significance of this script is high.

- Did john go to theater?
- Did he buy ticket?
- Did he have money?

If we have a sentence like "John went to theater to pick his daughter", then invoking this script will lead to many wrong answers.

- Here significance of the script theater is less.

Getting significance from the story is not straightforward. However, some heuristics can be applied to get the value.

Advantages/Disadvantages

- Advantages
  - ✓ Capable of predicting implicit events
  - ✓ Single coherent interpretation may be build up from a collection of observations.
- Disadvantage
  - ✓ More specific (inflexible) and less general than frames.
  - ✓ Not suitable to represent all kinds of knowledge.
- To deal with inflexibility, smaller modules called memory organization packets (MOP) can be combined in a way that is appropriate for the situation.

Rule Based Knowledge Representation System (RBS)

A rule-base system (or production system) is a Knowledge Based System (KBS) in which the knowledge is stored as rules; an expert system is a RBSs in which the rules come from human experts in a particular domain.

The underlying idea of production systems is to represent knowledge in the form of condition-action pairs called production rules.

- If the condition C is satisfied then the action A is appropriate.

Examples:

- If it is raining then open the umbrella.
- If Cesar is a man then Cesar is a person

Note: Production system is also called ruled-based system.

RBS can be used for problem-solving, or for control (eg of animate motion). In an RBS, the knowledge is separated from the AI reasoning processes, which means that new RBSs are easy to create. An RBS can be fast, flexible, and easy to expand

The syntax of rules is

IF <premise> THEN <action>

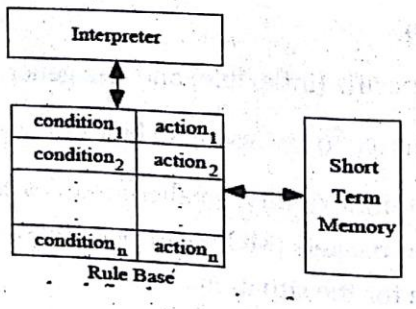
Components of RBS

Working memory – a small allocation of memory into which only appropriate rules are copied.

Rule base – the rules themselves, possibly stored specially to facilitate efficient access to the antecedent.

Interpreter – The processing engine which carries out reasoning on the rules and derives an answer.

A **user interface** or other connection to the outside world through which input and output signals are received and sent.



The Interpreter Cycle

The execution of a production system can be defined as a series of recognize-act cycles:

- Match–memory contain matched against condition of production rules, this produces a subset of production called conflict set.
- Conflict resolution –one of the production in the conflict set is then selected.
- Apply the rule.

This is also called Retrieval, Refinement and Execution cycle of Interpreter.

Example: Production system for string sorting

Problem: Sorting a string composed of letters a, b & c.
Short Term Memory: cbaca
Production Set:

1. ba → ab
2. ca → ac.
3. cb → bc

Interpreter: Choose one rule according to some strategy.

Iteration #	Memory	Conflict Set	Rule fired
0	cbaca	1, 2, 3	1
1	cabca	2	2
2	acbca	2, 3	2
3	acbac	1, 3	1
4	acabc	2	2
5	aacbc	3	3
6	aabcc	ø	halt

Conflict Resolution Strategies

The interpreter must choose one path to examine at a time, and so needs a method of conflict resolution in case of multiple rules.

Methods include:

- **Recency**
  - ✓ When two or more rules could be chosen, favor the one that matches the most recently added facts, as these are most likely to describe the current situation.

- **Not previously used**
  - ✓ If a rule's conditions are satisfied, but previously the same rule has been satisfied by the same facts, ignore the rule.
  - ✓ This helps to prevent the system from entering infinite loops.
- **FCFS:** choose the first rule that applies.
- **Order**
  - ✓ Pick the first applicable rule in order of presentation.
  - ✓ This is the strategy that Prolog interpreters use by default.
- priorities (by expert): choose the highest rank - most specific rule is chosen: eg, apply the rule with most elements in antecedent
- Arbitrary choice –
  - ✓ Pick a rule at random. This has the merit of being simple to compute.

Logic Based Knowledge Representations

The primary difference between PL and FOPL is their ontological commitment:

Ontological Commitment: What exists in the world — TRUTH

- PL: facts hold or do not hold.
- FL : objects with relations between them that hold or do not hold

Another difference is:

Epistemological Commitment: What an agent believes about facts — BELIEF

Language	Ontological Commitment	Epistemological Commitment
Propositional logic	facts	true/false/unknown
First-order logic	facts, objects, relations	true/false/unknown
Temporal logic	facts, objects, relations, times facts	true/false/unknown
Probability theory	degree of truth ∈ [0, 1]	degree of belief ∈ [0, 1]
Fuzzy logic		known interval value

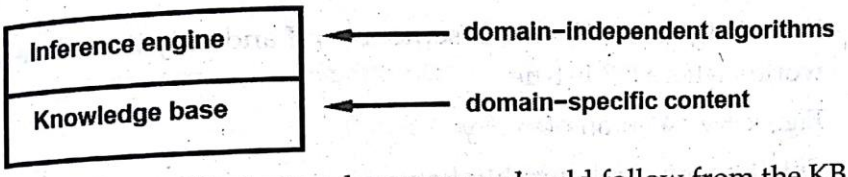
Propositional Logic

Logic (Knowledge-Based) agents combine general knowledge with current percepts to infer hidden aspects of current state prior to selecting actions

- Crucial in partially observable environments

Knowledge Base

Knowledge Base :set of sentences represented in a knowledge representation language and represents assertions about the world.



Then it can ask itself what to do-answers should follow from the KB

Agents can be viewed at the knowledge level

i.e., what they know, regardless of how implemented

Or at the implementation level

i.e., data structures in KB and algorithms that manipulate them

Abilities KB agent

Agent must be able to:

- Represent states and actions,
- Incorporate new percepts
- Update internal representation of the world
- Deduce hidden properties of the world
- Deduce appropriate actions

Propositional Logic

Logics are formal languages for representing information such that conclusions can be drawn: It has syntax and semantics:

- Syntax defines the sentences in the language
- Semantics define the meaning of sentences. that defines truth of a sentence in a world

E.g., the language of arithmetic

$x + 2 \geq y$ is a sentence; x2 + y > is not a sentence

$x + 2 \geq y$ is true iff the number $x + 2$ is no less than the number y

$x + 2 \geq y$ is true in a world where x=7; y =1

$x + 2 \geq y$ is false in a world where x=0; y =6

Entailment

Entailment means that one thing follows from another:

KB |= α

Knowledge base KB entails sentence α if and only if α is true in all worlds where KB is true

E.g., x + y =4 entails 4=x + y

Entailment is a relationship between sentences (i.e., syntax) that is based on semantics

Models

Logicians typically think in terms of models, which are formally structured worlds with respect to which truth can be evaluated.

m is a model of a sentence α if α is true in m.

M(α) is the set of all models of α

Logical Inference

If an inference algorithm i can derive α from KB, we write KB |-α which is pronounced as " i derives α from KB"

If an algorithm only derives entailed sentences it is called *sound* or *truth preserving*.

- Otherwise it just makes things up.

i is sound if whenever KB |-iα it is also true that KB |= α

Completeness : the algorithm can derive any sentence that is entailed.

i is complete if whenever KB |= α it is also true that KB |-iα

Propositional Logic: Syntax

Propositional logic is the simplest logic. We use the symbols like P1, P2 to represent sentences. Propositional logic is defined as:

If S is a sentence, ¬S is a sentence (negation)

If S1 and S2 are sentences, S1 ^ S2 is a sentence (conjunction)

If S1 and S2 are sentences, S1 ∨ S2 is a sentence (disjunction)

If S1 and S2 are sentences, S1 ⇒ S2 is a sentence (implication)

If S1 and S2 are sentences, S1 ⇔ S2 is a sentence (biconditional)

Formal grammar for propositional logic can be given as below:

Sentennce →AutomicSentence | ComplexSedntence

AutomicSentence → True | False | Symbol

Symbol → P | Q | R .....

ComplexSentence →¬Sentence

| (Sentence ^ Sentence)

| (Sentence∨ Sentence)

| (Sentence⇒ Sentence)

| (Sentence⇔ Sentence)

Propositional Logic: Semantics

Each model specifes true/false for each proposition symbol

Rules for evaluating truth with respect to a model:

¬S is true if, S is false

S1 ^ S2 is true if, S1 is true and S2 is true

S1 ∨S2 is true if, S1 is true or S2 is true

S1 ⇒ S2 is true if, S1 is false or S2 is true

S1 ⇔ S2 is true if, S1 ⇒ S2 is true and S2 ⇒ S1 is true

Truth Table showing the evaluation of semantics of complex sentences:

P	Q	¬P	P∧Q	P∨Q	P⇒Q	P⇔Q
false	false	true	false	false	true	true
false	true	true	false	true	true	false
true	false	false	false	true	false	false
true	true	false	true	true	true	true

Logical equivalence

Two sentences$\alpha$and ß are *logically equivalent*($\alpha \equiv$ ß) iff true they are true inn same set of models

or Two sentences$\alpha$and ß are *logically equivalent*($\alpha \equiv$ ß) iffα |= ß and ß |= α.

$$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha) \quad \text{commutativity of } \wedge$$
$$(\alpha \vee \beta) \equiv (\beta \vee \alpha) \quad \text{commutativity of } \vee$$
$$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma)) \quad \text{associativity of } \wedge$$
$$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma)) \quad \text{associativity of } \vee$$
$$\neg(\neg\alpha) \equiv \alpha \quad \text{double-negation elimination}$$
$$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha) \quad \text{contraposition}$$
$$(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta) \quad \text{implication elimination}$$
$$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) \quad \text{biconditional elimination}$$
$$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta) \quad \text{de Morgan}$$
$$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta) \quad \text{de Morgan}$$
$$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) \quad \text{distributivity of } \wedge \text{ over } \vee$$
$$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) \quad \text{distributivity of } \vee \text{ over } \wedge$$

Validity

A sentence is *valid* if it is true in all models,

e.g., *True*, A∨¬A, A ⇒A, (A ∧ (A ⇒ B)) ⇒ B

Valid sentences are also known as tautologies

Every valid sentence is logically equivalent to True

Satisfiability

A sentence is *satisfiable* if it is true in *some* model

- e.g., A ∨ B, C

A sentence is *unsatisfiable* if it is true in *no* models

- e.g., A¬∧A

Validity and satisfiablity are related concepts

- α is valid iff¬α is unsatisfiable
- α is satisfiableiff¬α is not valid

Satisfiability is connected to inference via the following:

- *KB* |= α if and only if (*KB*∧¬α) is unsatisfiable

Inference rules in PL

Modus Ponens

$$\frac{\alpha \Rightarrow \beta,\ \alpha}{\beta}$$

And-elimination

$$\frac{\alpha \wedge \beta}{\wedge\alpha}$$

Monotonicity: the set of entailed sentences can only increase as information is added to the knowledge base.

For any sentence α and β if KB |= α then KB ∧β |= α.

Resolution

Unit resolution rule:

Unit resolution rule takes a clause – a disjunction of literals – and a literal and produces a new clause. Single literal is also called unit clause.

$$\frac{\ell_1 \vee \ldots \vee \ell_k, \quad m}{\ell_1 \vee \ldots \vee \ell_{2-1} \vee \ell_{i+1} \vee \ldots \ell_k}$$

Where ℓ_i and m are complementary literals

Generalized resolution rule:

Generalized resolution rule takes two clauses of any length and produces a new clause as below.

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \quad m_1 \vee \cdots \vee m_n}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n}$$

For example,

$$\frac{\ell_1 \vee \ell_2, \quad \neg\ell_2 \vee \ell_3}{\ell_1 \vee \ell_3}$$

Resolution Uses CNF (Conjunctive normal form)

- Conjunction of disjunctions of literals (clauses)

  The resolution rule is sound:

- Only entailed sentences are derived

Resolution is complete in the sense that it can always be used to either confirm or refute a sentence (it cannot be used to enumerate true sentences.)

Conversion to CNF

A sentence that is expressed as a conjunction of disjunctions of literals is said to be in conjunctive normal form (CNF).

A sentence in CNF that contains only k literals per clause is said to be in k-CNF.

Let's illustrate the conversion to CNF by using an example.

$B \Leftrightarrow (A \vee C)$

- Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow ß$ with $(\alpha \Rightarrow ß) \wedge (ß \Rightarrow \alpha)$.
  - ✓ $(B \Rightarrow (A \vee C)) \wedge ((A \vee C) \Rightarrow B)$
- Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow ß$ with $\neg\alpha \vee ß$.
  - ✓ $(\neg B \vee A \vee C) \wedge (\neg(A \vee C) \vee B)$
- Move $\neg$ inwards using de Morgan's rules and double-negation:
  - ✓ $(\neg B \vee A \vee C) \wedge ((\neg A \wedge \neg C) \vee B)$
- Apply distributivity law ($\wedge$ over $\vee$) and flatten:
  - ✓ $(\neg B \vee A \vee C) \wedge (\neg A \vee B) \wedge (\neg C \vee B)$

Resolution algorithm

Convert KB into CNF

Add negation of sentence to be entailed into KB i.e. $(KB \wedge \neg\alpha)$

Then apply resolution rule to resulting clauses.

The process continues until:

- There are no new clauses that can be added
- Hence *KB* **does not** entail α
- Two clauses resolve to entail the empty clause.
- Hence *KB* **does** entail α

Resolution example

Consider the knowledge base given as below:

$KB = (B \Leftrightarrow (A \vee C)) \wedge \neg B$

Prove that $\neg A$ can be inferred from above KB by using resolution.

Convert KB into CNF

$B \Rightarrow (A \vee C)) \wedge ((A \vee C) \Rightarrow B) \wedge \neg B$

$(\neg B \vee A \vee C) \wedge (\neg(A \vee C) \vee B) \wedge \neg B$

$(\neg B \vee A \vee C) \wedge ((\neg A \wedge \neg C) \vee B) \wedge \neg B$

$(\neg B \vee A \vee C) \wedge (\neg A \vee B) \wedge (\neg C \vee B) \wedge \neg B$

Add negation of sentence to be inferred from KB into KB

Now KB contains following sentences all in CNF

$(\neg B \vee A \vee C)$

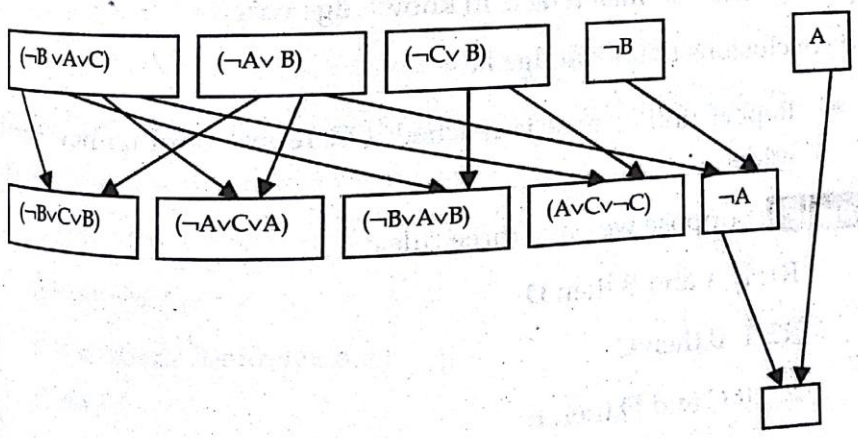
$(\neg A \vee B)$

$(\neg C \vee B)$

$\neg B$

A

Now use Resolution algorithm



Forward and Backward Chaining

The completeness of resolution makes it a very important inference model. But in many practical situations full power of resolution is not

needed. Real-world knowledge bases often contain only clauses of restricted kind called **Horn Clause.**

A Horn clauses is disjunction of literals with at most one positive literal

Three important properties of Horn clause are:

- Can be written as an implication
- Inference through forward chaining and backward chaining.
- Deciding entailment can be done in a time linear size of the knowledge base.

Forward chaining

Forward chaining is the logical process of inferring unknown truths from known data and moving forward using determined conditions and rules to find a solution

It is also know as data driven reasoning.

Steps:

- Given knowledge base of true facts:

Apply all rules that match facts in knowledge base

Add conclusions to knowledge base.

- Repeat until a goal is reached, OR repeat until no new facts added

Example 1: Suppose we have three rules:

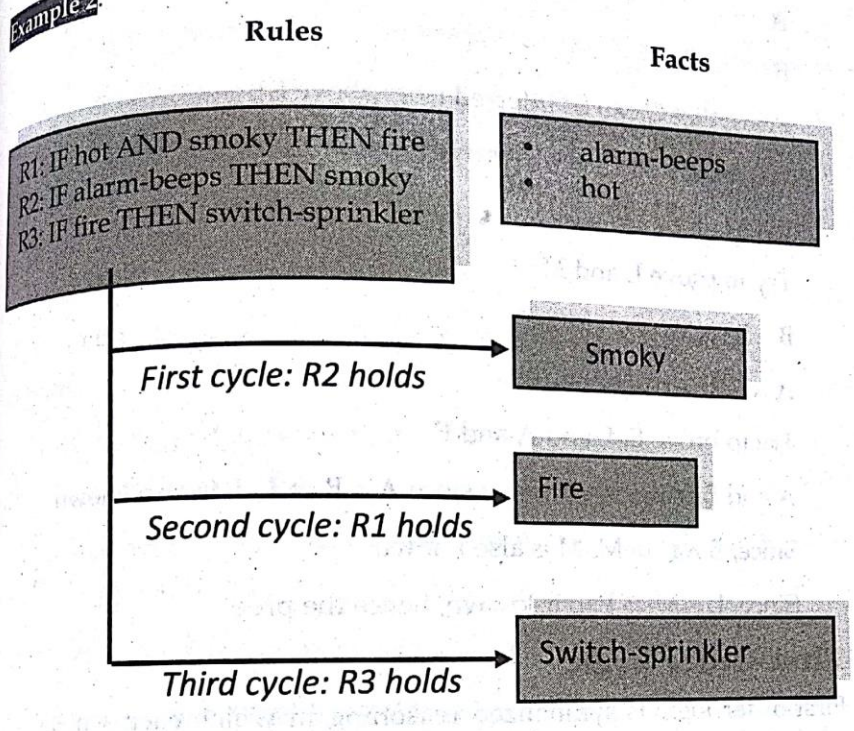
R1: If A and B then D

R2: If B then C

R3: If C and D then E

If facts A and B are present, we infer D from R1 and infer C from R2. With D and C inferred, we now infer E from R3.

Example 2:



Backward Chaining

Idea: work backwards from the query q: to prove q by BC,

Check if q is known already, or

Prove by BC all premises of some rule concluding q

Avoid loops: check if new subgoal is already on the goal stack

Avoid repeated work: check if new subgoal

1. has already been proved true, or
2. has already failed

Foe example for above KB

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

Prove that Q can be inferred from above KB

We know $P \Rightarrow Q$, try to prove P

$L \wedge M \Rightarrow P$

Try to prove L and M

$B \wedge L \Rightarrow M$

$A \wedge P \Rightarrow L$

Try to prove B, L and A and P

A and B is already known, since $A \wedge B \Rightarrow L$, L is also known

Since, $B \wedge L \Rightarrow M$, M is also known

Since, $L \wedge M \Rightarrow P$, p is known, hence the proof

First-Order Logic

First-order logic is symbolized reasoning in which each sentence, or statement, is broken down into a subject and a predicate. The predicate modifies or defines the properties of the subject. In first-order logic, a predicate can only refer to a single subject. First-order logic is also known as first-order predicate calculus or first-order functional calculus.

A sentence in first-order logic is written in the form Px or P(x), where P is the predicate and x is the subject, represented as a variable.

Atomic Sentence:

A atomic sentence is formed from a predicate symbol followed by a parenthesized list of terms.

E.g., Brother (Bill, Mary),

Father (Jack, John), Mother (Jill, John),

Sister (Jane, John)

Married (Jack, Jill)

BrotherOf(Bill) = Bob

Complex Sentence:

Complex sentences are made from atomic sentences using connectives.

E.g., Sibling (Tom, Bill) ⇒Sibling (Bill, Tom).

Father (Jack, John) ∧Mother (Jill, John) ∧Sister (Jane, John).

¬Sister (John, Jane).

Older (Father-Of (John), 30) ∨Older (Mother-Of (John), 20).

The first-order logic (like natural language) assumes the world contains

- Objects: people, houses, numbers, colors, baseball games, wars,
- Relations: red, round, prime, brother of, bigger than, part of, comes between,
- Functions: father of, best friend, one more than, plus,

Representing knowledge in first-order logic

The objects from the real world are represented by constant symbols (a,b,c,...).

For instance, the symbol "Tom" may represent a certain individual called Tom.

Properties of objects may be represented by predicates applied to those objects (P(a), ...): e.g "male(Tom)" represents that Tom is a male.

Relationships between objects are represented by predicates with more arguments: "father(Tom, Bob)" represents the fact that Tom is the father of Bob.

The value of a predicate is one of the boolean constants T (i.e. true) or F (i.e. false)."father(Tom, Bob) = T" means that the sentence "Tom is the father of Bob" is true. "father(Tom, Bob) = F" means that the sentence "Tom is the father of Bob" is false.

Besides constants, the arguments of the predicates may be functions (f,g,...) or variables (x,y,...).

Function symbols denote mappings from elements of a domain (or tuples of elements of domains) to elements of a domain. For instance, weight is a function that maps objects to their weight: weight (Tom) = 150.Therefore the predicate greater-than (weight (Bob), 100) means that the weight of Bob is greater than 100. The arguments of a function may themselves be functions.

Variable symbols represent potentially any element of a domain and allow the formulation of general statements about the elements of the domain.

The quantifier's ∀ and ∃ are used to build new formulas from old ones.

"∃x P(x)" expresses that there is at least one element of the domain that makes P(x) true.

"∃x mother(x, Bob)" means that there is x such that x is mother of Bob or, otherwise stated, Bob has a mother.

"∀x P(x)" expresses that for all elements of the domain P(x) is true.

Quantifiers

Allows us to express properties of collections of objects instead of enumerating objects by name. Two quantifiers are:

Universal: "for all" ∀

Existential: "there exists" ∃

Universal quantification:

∀<*Variables*><*sentence*>

Everyone at VUB is smart:

∀x At(x,VUB) ⇒ Smart(x)

∀x*P* is true in a model *miff P* is true for all *x* in the model

Roughly speaking, equivalent to the conjunction of instantiations of *P*

At(KingJohn,VUB) ⇒ Smart(KingJohn) ∧ At(Richard,VUB) ⇒ Smart(Richard)

∧At(VUB,VUB) ⇒ Smart(VUB)∧ ...

Typically, ⇒ is the main connective with ∀

- A universally quantifier is also equivalent to a set of implications over all objects

Common mistake: using ∧ as the main connective with ∀:

∀x At(x, VUB) ∧ Smart(x)

means "Everyone is at VUB and everyone is smart"

Existential quantification

∃<*variables*><*sentence*>

Someone at VUB is smart:

∃x At(x, VUB) ∧ Smart(x)

∃x*P* is true in a model *m*iff*P* is true for at least one *x* in the model

Roughly speaking, equivalent to the disjunction of instantiations of *P*

At(KingJohn,VUB) ∧ Smart(KingJohn)∨At(Richard,VUB) ∧ Smart(Richard)

∨At(VUB, VUB) ∧ Smart(VUB) ∨ ...

Typically, ∧ is the main connective with ∃

Common mistake: using ⇒ as the main connective with ∃:

∃x At(x, VUB) ⇒ Smart(x)

is true even if there is anyone who is not at VUB!

Properties of quantifiers

∀x∀y is the same as ∀y ∀x

∃x∃y is the same as ∃y ∃x

∃x∀y is not the same as ∀y ∃x

∃x∀y Loves(x,y)

- "There is a person who loves everyone in the world"

  ∀y ∃x Loves(x,y)

- "Everyone in the world is loved by at least one person"

Quantifier duality: each can be expressed using the other

$\forall x$ Likes(x, IceCream) $\equiv \exists \neg x$ $\neg$Likes(x, IceCream)

$\exists x$ Likes(x, Broccoli) $\equiv \neg \forall x$ $\neg$Likes(x, Broccoli)

Equality

term1 = *term2* is true under a given interpretation if and only if *term1* and *term2* refer to the same object

e.g.

Father (John) = Henary

Here Objects referred by father of john and Henary is same.

Inference in first-order logic

First Order Logic to Predicate Logic

First order inference can be done by converting the knowledge base to PL and using propositional inference.

- How to convert universal quantifiers?
  - Replace variable by ground term.
- How to convert existential quantifiers?
  - Skolemization.

Universal instantiation (UI)

Substitute ground term (term without variables) for the variables.

For example consider the following KB

$\forall x$ King (x) $\wedge$ Greedy (x) $\Rightarrow$ Evil(x)

King (John)

Greedy (John)

Brother (Richard, John)

It's UI is:

King (John) $\wedge$ Greedy (John) $\Rightarrow$Evil(John)

King (Richard) $\wedge$ Greedy (Richard) $\Rightarrow$Evil(Richard)

King (John)

Greedy (John)

Brother (Richard, John)

Note: Remove universally quantified sentences after universal instantiation.

Existential instantiation (EI)

For any sentence α and variable v in that, introduce a constant that is not in the KB (called skolem constant) and substitute that constant for v.

e.g.

Consider the sentence.

∃x crown(x) ∧OnHead(x, John)

After EI

crown(C1) ∧OnHead(C1, John) where C1 is Skolem Constant.

EI versus UI

UI can be applied several times to *add* new sentences; the new KB is logically equivalent to the old.

EI can be applied once to replace the existential sentence; the new KB is not equivalent to the old but is satisfiable if the old KB was satisfiable.

Note: After propositionalization, we can use any algorithm fool Pl to derive the conclusion.

Unification and Lifting

Instead of translating the knowledge base to PL, we can redefine the inference rules into FOL.

- Lifting: they only make those substitutions that are required to allow particular inferences to proceed.
- E.g. *generalized Modus Ponens*
  - ✓ To introduce substitutions different logical expressions have to be look identical
  - ✓ To identify the substitution we need unification algorithm.

Unification:

We can get the inference immediately if we can find a substitution α such that *King(x)* and *Greedy(x)* match *King(John)* and *Greedy(y)*

α = {x/John,y/John} works

Unify(α ,β) = θ if $\alpha\theta = \theta\beta$

p	q	θ
Knows(John,x)	Knows(John,Jane)	{x/Jane}
Knows(John,x)	Knows(y,OJ)	{x/OJ,y/John}
Knows(John,x)	Knows(y,Mother(y))	{y/John,x/Mother(John)}}
Knows(John,x)	Knows(x,OJ)	{fail}

Last unification is failed due to overlap of variables. xcan not take the values of John and OJ at the same time.

We can avoid this problem by renaming to avoid the name clashes (standardizing apart)

e.g.

Unify{Knows(John, x) Knows(z, OJ)} = {x/OJ, z/John}

Another complication:

To unify *Knows(John,x)* and *Knows(y,z)*,

Unification of Knows*(John,x)* and *Knows(y,z)* gives α ={y/John, x/z } or α={y/John, x/John, z/John}

First unifier gives the result Knows(John,z) and second unifier gives the resultKnows(John, John). Second can be achieved from first by substituting john in place of z. The first unifier is more general than the second.

There is a single most general unifier (MGU) that is unique up to renaming of variables.

MGU = { y/John, x/z }

Generalized Modus Ponens (GMP)

$$\frac{p_1', p_2', \ldots, p_n', (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{\text{where } p_i'\theta = p_i\theta \text{ for all } i}$$

qθ

p1' is King(John) p1 is King(x)

p2' is Greedy(y) p2 is Greedy(x)

θ is {x/John,y/John} q is Evil(x)

qθ is Evil(John)

GMP used with KB of definite clauses (exactly one positive literal).

Forward Chaining:

We can use FC foe FOPL also:

Consider the following KB:

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 Colonel West, who is American.

Prove that Col. West is a criminal

Solution:

It is a crime for an American to sell weapons to hostile nations:

American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)

Nono … has some missiles, i.e.,

∃x Owns(Nono,x) ∧ Missile(x):0

Use Existential Instantiation:

Owns(Nono,M1) and Missile(M1)

… all of its missiles were sold to it by Colonel West

Missile(x) ∧ Owns(Nono,x) ⇒ Sells(West,x,Nono)

Missiles are weapons:

Missile(x) ⇒ Weapon(x)

An enemy of America counts as "hostile":

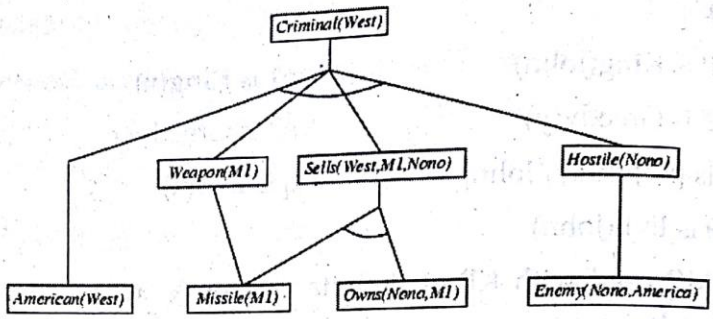
Enemy(x,America) ⇒ Hostile(x)

West, who is American …

American(West)

The country Nono, an enemy of America …

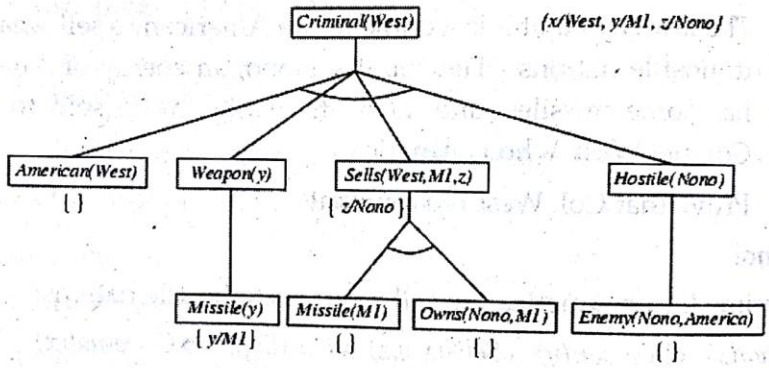
Enemy(Nono, America)



Backward Chaining

We can also use BC algorithm for FOPL.

Eg. For above KB we can use BC al below:



Resolution Algorithms

Probability and Bayesian Network

A probability is a number that reflects the chance or likelihood that a particular event will occur. Probabilities can be expressed as proportions that range from 0 to 1, and they can also be expressed as percentages ranging from 0% to 100%.

A probability of 0 indicates that there is no chance that a particular event will occur, whereas a probability of 1 indicates that an event is certain to occur. A probability of 0.45 (45%) indicates that there are 45 chances out of 100 of the event occurring.

Properties of Probability

- The probability of an event can only be between 0 and 1 and can also be written as a percentage.

- The probability of event A is generally written as P(A)
- If P(A)>P(B) then eventAhas a higher chance of occurring than event B.
- If P(A)=P(B), then events A and B are equally likely to occur.

Probability comes into two types:

1. **Prior Probability:** The probability of event A is the number of ways event A can occur divided by the total number of possible outcomes.

$$P(A) = \frac{\text{The Number Of Ways Event A Can Occur}}{\text{The total number Of Possible Outcomes}}$$

2. **Posterior probability or conditional probability:** Conditional probability is the probability of one event occurring with some relationship to one or more other events.

For example:

Event A is that it is raining outside, and it has a 0.3 (30%) chance of raining today.

Event B is that you will need to go outside, and that has a probability of 0.5 (50%).

A conditional probability would look at these two events in relationship with one another, such as the probability that it is both raining and you will need to go outside.

The formula for conditional probability is:

$$P(B \mid A) = P(A \text{ and } B) / P(A)$$

which can be written as:

$$P(B \mid A) = P(A \cap B) / P(A)$$

Example: In a group of 100 sports car buyers, 40 bought alarm systems, 30 purchased bucket seats, and 20 purchased an alarm system and bucket seats. If a car buyer chosen at random bought an alarm system, what is the probability they also bought bucket seats?

Solution:

Step 1

Figure out P(A). It's given in the question as 40%, or 0.4.

Step 2

Figure out P(A∩B). This is the intersection of A and B: both happening together. It's given in the question 20 out of 100 buyers, or 0.2.

Step 3

Insert your answers into the formula:

$$P(B \mid A) = P(A \cap B) / P(A) = 0.2 / 0.4 = 0.5.$$

The probability that a buyer bought bucket seats, given that they purchased an alarm system, is 50%.

Bayes Theorem

Bayes' theorem is a way to apply conditional probability for prediction. Conditional probability is the probability of an event happening, given that it has some relationship to one or more other events. For example, your probability of getting a parking space is connected to the time of day you park, where you park, and what conventions are going on at any time. Bayes' theorem is slightly more nuanced.

Mathematically, Bayes theorem is defined as:

$$P(A \mid B) = \frac{(B \mid A)P(A)}{P(B)}$$

Example 1: Find out a patient's probability of having liver disease if they are an alcoholic. "Being an alcoholic" is the test for liver disease with following

- Let A could mean the event "Patient has liver disease." Past data tells you that 10% of patients entering your clinic have liver disease. P(A) = 0.10.

- B could mean the litmus test that "Patient is an alcoholic." Five percent of the clinic's patients are alcoholics. P(B) = 0.05.
- Among those patients diagnosed with liver disease, 7% are alcoholics. Then P(B|A): the probability that a patient is alcoholic, given that they have liver disease, is 7%.

Now, By using Bayes' theorem We get,

P(A|B) = (0.07 * 0.1)/0.05 = 0.14

In other words, if the patient is an alcoholic, their chances of having liver disease is 0.14 (14%).

Example 2: Given the following statistics, what is the probability that a woman has cancer if she has a positive mammogram result?

- One percent of women over 50 have breast cancer.
- Ninety percent of women who have breast cancer test positive on mammograms.
- Eight percent of women will have false positives.

Solutions:

Step 1:

Assign events to A or X. You want to know what a woman's probability of having cancer is, given a positive mammogram. For this problem, actually having cancer is A and a positive test result is X.

Step 2:

List out the parts of the equation (this makes it easier to work the actual equation): P(A)=0.01 P(~A)=0.99 P(X|A)=0.9 P(X|~A)=0.08

Step 3:

Insert the parts into the equation and solve. Note that as this is a medical test, we're using the form of the equation from example #2:

(0.9 * 0.01) / ((0.9 * 0.01) + (0.08 * 0.99) = 0.10.

The probability of a woman having cancer, given a positive test result, is 10%.

Example 3 A doctor knows that the disease meningitis causes the patient tohave a stiff neck 50% of the time. The doctor also knows that the probability that a patient has meningitis is 1/50,000, and the probability that any patient has a stiff neck is 1/20.Find the probability that a patient with a stiff neck has meningitis.

Solution:

p(s | m) = 0.5

p(m) = 1/50000

p(s) = 1/20

P(m | s) = P(s | m)P(m)/P(s) = (0.5*1/50000)/(1/20) = 0.0002

Probability Distribution

The probability distribution is the distribution table formed to obtain the occurrence of each event in the sample space. From the probability distribution table, we can find the other statistics such as mean, standard deviation, and expected value.

For example

Example 1: If a coin is tossed, find the probability distribution of getting tails.

Solution:

If a coin is tossed, either 0 tail or 1 tail can be obtained.

Probability of getting zero tail, P(0) = 12

Probability of getting one tail, P(1) = 12

The probability distribution table will be,

X	0	1
P(X)	1/2	1/2

Example 2: If two coins are tossed simultaneously, find the probability distribution of getting head.

Solution:

If two coins are being tossed, there can be either 0 heads, 1 head, or 2 heads.

Sample space, S = {HH, HT, TH, TT}

The probability distribution of getting heads can be shown as:

X	0	1	2
P(X)	1/4	1/4	1/4

Example 3: In a class on 100 students, 80 students passed in all subjects, 10 failed in one subject, 7 failed in two subjects and 3 failed in three subjects. Find the probability distribution of the variable for number of subjects a student from the given class has failed in.

Solution:

For a random student,

The probability of failing in 0 subjects, P(X=0) = 0.8

The probability of failing in 1 subjects, P(X=1) = 0.1

The probability of failing in 2 subjects, P(X=2) = 0.07

The probability of failing in 3 subjects, P(X=3) = 0.03

The probability distribution can be shown as:

X	0	1	2	3
P(X)	0.8	0.1	0.07	0.03

Joint Probability Distribution

Joint probability is a statistical measure that calculates the likelihood of two events occurring together and at the same point in time. Joint probability is the probability of event Y occurring at the same time that event X occurs.

For example if we have two random variables: Weather and Cavity with set of domain for Weather={Rainy, Sunny, Cloudy, Snow} and Cavity={True, False} then P(Weather, Cavity) = a 4 ×2 matrix of values.

Weather=	Sunny	Rainy	Cloudy	Snow
Cavity=True	0.144	0.02	0.016	0.02
Cavity=False	0.576	0.08	0.064	0.08

Bayesian Network

Bayesian networks are a type of probabilistic graphical model that uses Bayesian inference for probability computations. Bayesian networks aim to model conditional dependence, and therefore causation, by representing conditional dependence by edges in a directed graph. Through these relationships, one can efficiently conduct inference on the random variables in the graph through the use of factors.

A data structure to represent the dependencies among variables and to give a concise specification of any full joint probability distribution.

It is also called belief networksor probabilistic networkor casual networkor knowledge map.

Definition:

A Bayesian network is a directed acyclic graph which consists of:

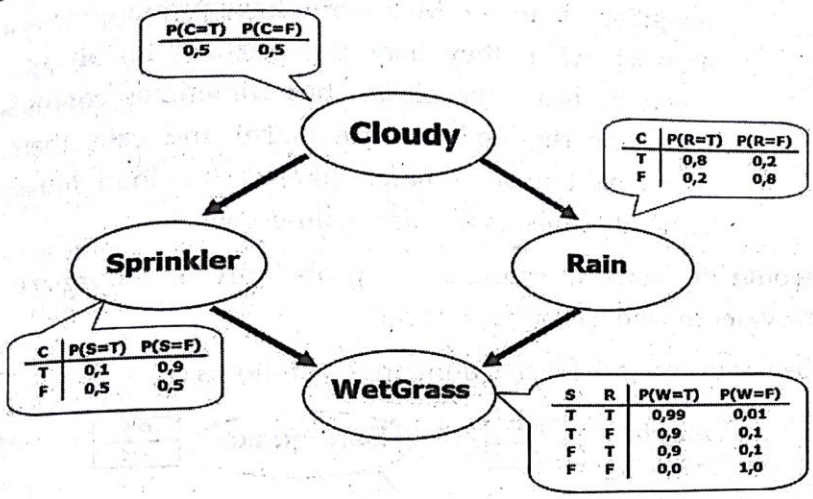
- A set of random variables which makes up the nodes of the network.
- A set of directed links (arrows) connecting pairs of nodes. If there is an arrow from node X to node Y, X is said to be a parent of Y.
- Each node Xihas a conditional probability distribution P(Xi| Parents(Xi)) that quantifies the effect of the parents on the node

Formally, if an edge (A, B) exists in the graph connecting random variables A and B, it means that P(B|A) is a **factor** in the joint probability distribution, so we must know P(B|A) for all values of B and A in order to conduct inference.

Example 1 : Let us consider the following sample domain

The cloud can causes rain or sprinkler. Two events can cause grass to be wet: an active sprinkler or rain. Rain has a direct effect on the use of the sprinkler (namely that when it rains, the sprinkler usually is not

active). This situation can be modeled with a Bayesian network (shown below). Each variable has two possible values, T (for true) and F (for false).



In the above example, since Rain has an edge going into WetGrass, it means that P(WetGrass | Rain) will be a factor, whose probability values are specified next to the WetGrass node in a conditional probability table (attached with node).

Bayesian networks satisfy the local Markov property, which states that a node is conditionally independent of its non-descendants given its parents.

In the above example,

$$P(\text{Sprinkler} \mid \text{Cloudy, Rain}) = P(\text{Sprinkler} \mid \text{Cloudy})$$

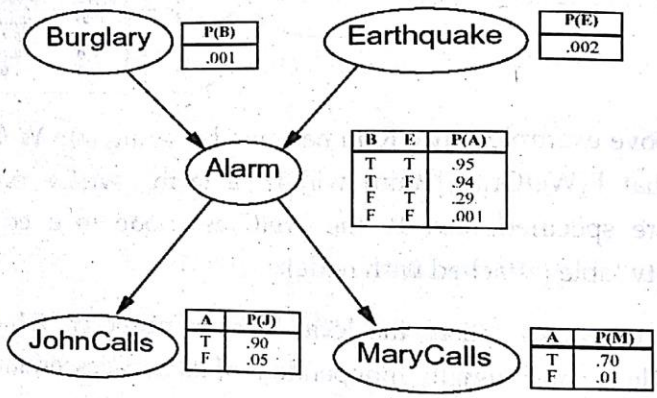
Since Sprinkler is conditionally independent of its non-descendant, Rain, given Cloudy. This property allows us to simplify the joint distribution, obtained in the previous section using the chain rule, to a smaller form. After simplification, the joint distribution for a Bayesian network is equal to the product of P(node | parents(node)) for all nodes, stated below:

$$P(X_1, \ldots X_n) = \prod_{i=1} P(x_i \mid X_i \ldots, X_{i-1}) = \prod_{i=1} P(X \mid \text{Parents}(X_i))$$

Example 2: You have a burglar alarm installed in your home. It is fairly reliable at detecting a burglary, but also responds on occasion to minor earthquakes. You also have two neighbors, John and Mary, who have promised to call you at work when they hear the alarm. John always calls when he hears the alarm, but sometimes confuses the telephone ringing with the alarm and calls then, too. Mary, on the other hand, likes rather loud music and sometimes misses the alarm altogether.

We would like have to estimate the probability of a burglary with given evidence who has or has not call.

The Bayesian network for this domain is as follows:



Inference with Bayesian Network

The first is simply evaluating the joint probability of a particular assignment of values for each variable (or a subset) in the network. For this, we already have a factorized form of the joint distribution, so we simply evaluate that product using the provided conditional probabilities. If we only care about a subset of variables, we will need to marginalize out the ones we are not interested in.

Example:

What is the probability that the alarm has sounded, but neitherburglary nor an earthquake has occurred, and both John and Mary call?

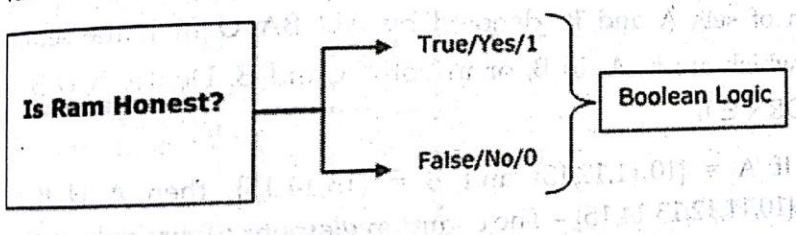
$$P(j, m, a, \neg b, \neg e) = P(j|a)\, P(m|a)\, P(a|, \neg b, \neg e)\, P(\neg b)\, P(\neg e) =$$
$$= 0.90*0.70*0.001*0.999*0.998 = 0.00062$$

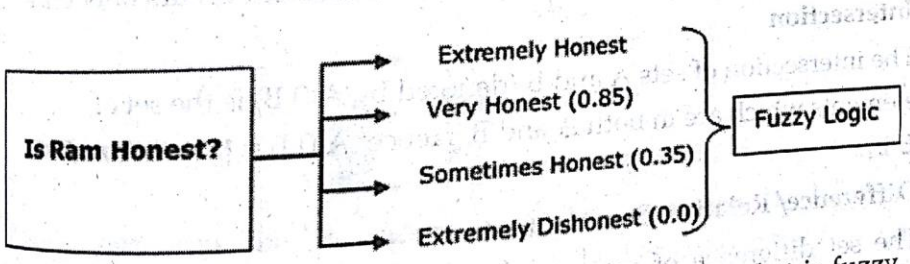
Fuzzy Logic

The word fuzzy refers to things which are not clear or are vague. Any event, process, or function that is changing continuously cannot always be defined as either true or false, which means that we need to define such activities in a Fuzzy manner.

Classical logic only permits conclusions which are either true or false. However, there are also propositions with variable answers, such as one might find when asking a group of people to identify a color.

Take a look at the following diagram. It shows that in fuzzy systems, the values are indicated by a number in the range from 0 to 1. Here 1.0 represents absolute truth and 0.0 represents absolute falseness. The number which indicates the value in fuzzy systems is called the truth value.





In other words, we can say that fuzzy logic is not logic that is fuzzy, but logic that is used to describe fuzziness. There can be numerous other examples like this with the help of which we can understand the concept of fuzzy logic.

Fuzzy Logic was introduced in 1965 by Lofti A. Zadeh in his research paper "Fuzzy Sets". He is considered as the father of Fuzzy Logic

Set

A set is an unordered collection of different elements. It can be written explicitly by listing its elements using the set bracket. If the order of

the elements is changed or any element of a set is repeated, it does not make any changes in the set.

Example

- A set of all positive integers.
- A set of all the planets in the solar system.
- A set of all the states in India.
- A set of all the lowercase letters of the alphabet.

Operations on Classical Sets

Set Operations include Set Union, Set Intersection, Set Difference, Complement of Set, and Cartesian Product.

Union

The union of sets A and B (denoted by AU BA U B) is the set of elements which are in A, in B, or in both A and B. Hence, $A \cup B = \{x \mid x \in A \text{ OR } x \in B\}$.

Example If A = {10,11,12,13} and B = {13,14,15}, then $A \cup B$ = {10,11,12,13,14,15} – The common element occurs only once.

Intersection

The intersection of sets A and B (denoted by $A \cap B$) is the set of elements which are in both A and B. Hence, $A \cap B = \{x \mid x \in A \text{ AND } x \in B\}$.

Difference/ Relative Complement

The set difference of sets A and B (denoted by A–B) is the set of elements which are only in A but not in B. Hence, $A - B = \{x \mid x \in A \text{ AND } x \notin B\}$.

Example If A = {10,11,12,13} and B = {13,14,15}, then (A – B) = {10,11,12} and (B – A) = {14,15}. Here, we can see $(A - B) \neq (B - A)$

Complement of a Set

The complement of a set A (denoted by A′) is the set of elements which are not in set A. Hence, $A' = \{x \mid x \notin A\}$.

More specifically, A′ = (U−A) where U is a universal set which contains all objects.

Example If A = {x | x belongs to set of add integers} then A′ = {y | y does not belong to set of odd integers}

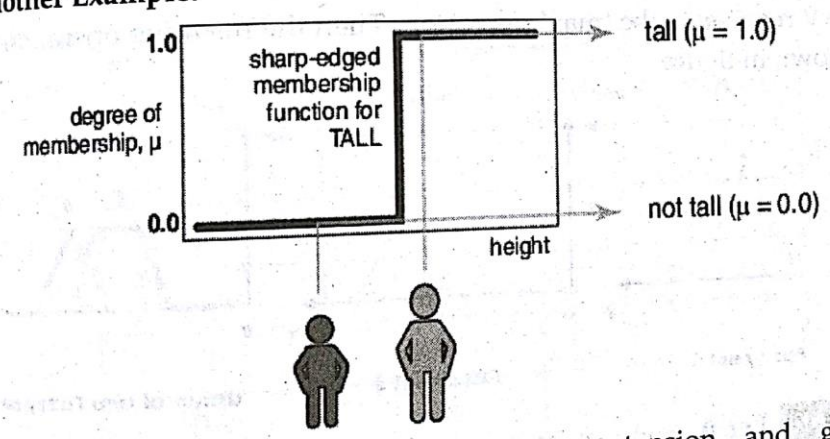
Fuzzy Set

A set X in which each element y has a grade of membership $\mu_X(y)$ in the range 0 to 1, i.e. set membership may be partial.

For Example: If *cold* is a fuzzy set, exact temperature values might be mapped to the fuzzy set as follows:

- 15 degrees → 0.2 (slightly cold)
- 10 degrees → 0.5 (quite cold)
- 0 degrees → 1 (totally cold)

Another Example:



Fuzzy sets can be considered as an extension and gross oversimplification of classical sets. It can be best understood in the context of set membership. Basically it allows partial membership which means that it contain elements that have varying degrees of membership in the set. From this, we can understand the difference between classical set and fuzzy set. Classical set contains elements that satisfy precise properties of membership while fuzzy set contains elements that satisfy imprecise properties of membership.

Formal Fuzzy Logic

Fuzzy logic can be seen as an extension of ordinary (bivalent) logic, where the main difference is that we use fuzzy sets for the

membership of a variable. We can have fuzzy propositional logic and fuzzy predicate logic.

Fuzzy logic can have many advantages over ordinary logic in areas like artificial intelligence where a simple true/false statement is insufficient.

Fuzzy Set Operations

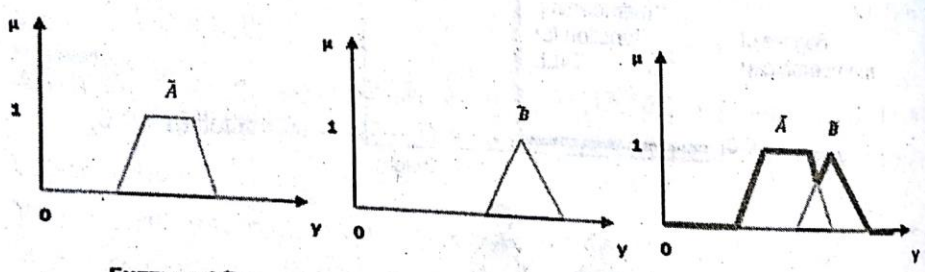
Having two fuzzy sets $\tilde{A}$ and $\tilde{B}$, the universe of information U and an element y of the universe, the following relations express the union, intersection and complement operation on fuzzy sets.

Union/Fuzzy 'OR'

Let us consider the following representation to understand how the **Union/Fuzzy 'OR'** relation works :

$$\mu_{A \cup B} = \mu_A \vee \mu_B$$

Here $\vee$ represents the 'max' operation. Then the resulting operation is as shown in figure



Fuzzy set $\tilde{A}$ Fuzzy set $\tilde{B}$ Union of two Fuzzy sets

Example: Let μ_A and μ_B represents the two fuzzy set with given members as shown in tables then the resulting union of these two sets is $\mu_{A \cup B}$ =max(A,B).

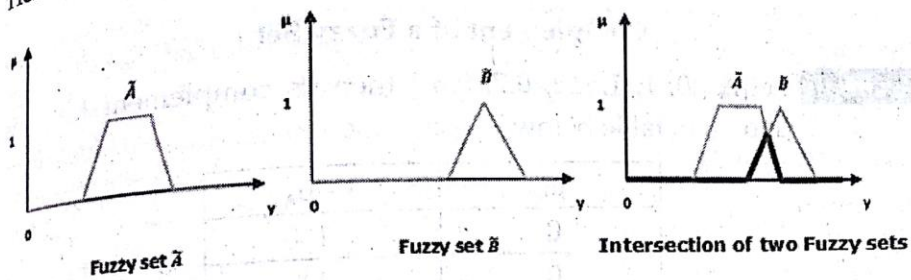
μ_A	μ_B	$\mu_{A \cup B}$ =max(A,B)
0	0	0
0	1	1
1	0	1
1	1	1
0.2	0.5	0.5
0.7	0.2	0.7
0.6	0.6	0.6

Intersection/Fuzzy 'AND'

Let us consider the following representation to understand how the **Intersection/Fuzzy 'AND'** relation works –

$$\mu\tilde{A} \cap \tilde{B}(y) = \mu\tilde{A} \wedge \mu\tilde{B} \; \forall y \in U$$

Here $\wedge$ represents the 'min' operation.



Fuzzy set $\tilde{A}$ Fuzzy set $\tilde{B}$ Intersection of two Fuzzy sets

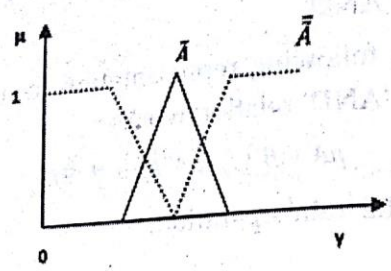
Example: Let μ_A and μ_B represents the two fuzzy set with given members as shown in tables then the resulting union of these two sets is $\mu_{A \cup B}$ =max(A,B).

μ_A	μ_B	$\mu_{A \wedge B}$ =min(A,B)
0	0	0
0	1	0
1	0	0
1	1	1
0.2	0.5	0.2
0.7	0.2	0.2
0.6	0.6	0.6

Complement/Fuzzy 'NOT'

Let us consider the following representation to understand how the **Complement/Fuzzy 'NOT'** relation works –

$$\mu\tilde{A} = 1 - \mu\tilde{A}(y) y \in U$$



Complement of a Fuzzy Set

Example Let μ_A = {0, 1, 1, 0.2, 0.7, 0.6 } then its complement 1 - μ_A is shown in table below.

μ_A	1 - μ_A
0	1
0	1
1	0
1	0
0.2	0.8
0.7	0.3
0.6	0.4

Summary

Traditional logic uses the Boolean operators AND, OR, and NOT to perform the intersect, union and complement operations. These operators work well for bivalent sets and can be essentially defined using the following truth table:

x	y	x AND y	x OR y	NOT x
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

The truth table above works fine for bivalent logic but fuzzy logic does not have a finite set of possibilities for each input; this makes for an infinitely large truth table. The operators need to be defined as functions for all possible fuzzy values, that is, all real numbers from 0 to 1 inclusive. *Fuzzy logic is actually a superset of bivalent logic since it includes the bivalent options (0,1) as* well as all reals in between, so a

generalized form of these operators will be usefull. The generalized form for these three operators are:

x AND y	min(x,y)
x OR y	max(x,y)
NOT x	1 - x

Using these definitions they can be applied to all of the bivalent combinations above as well as some fuzzy number combinations. The truth table for this can be seen below:

r	y	min(x,y)	max(x,y)	1 - x
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0
0.2	0.5	0.2	0.5	0.8
0.7	0.2	0.2	0.7	0.3
0.6	0.6	0.6	0.6	0.4

Properties of Fuzzy Sets

Let us discuss the different properties of fuzzy sets.

Commutative Property

Having two fuzzy sets $\tilde{A}$ and $\tilde{B}$, this property states –

$$\tilde{A} \cup \tilde{B} = \tilde{B} \cup \tilde{A}$$

$$\tilde{A} \cap \tilde{B} = \tilde{B} \cap \tilde{A}$$

Associative Property

Having three fuzzy sets $\tilde{A}$, $\tilde{B}$ and $\tilde{C}$, this property states –

$$\tilde{A} \cup (\tilde{B} \cup \tilde{C}) = (\tilde{A} \cup \tilde{B}) \cup \tilde{C}$$

$$\tilde{A} \cap (\tilde{B} \cap \tilde{C}) = (\tilde{A} \cup \tilde{B}) \cup \tilde{C}$$

Distributive Property

Having three fuzzy sets $\tilde{A}$, $\tilde{B}$ and $\tilde{C}$, this property states –

$$\tilde{A} \cup (\tilde{B} \cap \tilde{C}) = (\tilde{A} \cup \tilde{B}) \cap (\tilde{A} \cup \tilde{C})$$

$$\tilde{A} \cap (\tilde{B} \cup \tilde{C}) = (\tilde{A} \cap \tilde{B}) \cup (\tilde{A} \cap \tilde{C})$$

Idempotency Property

For any fuzzy set $\tilde{A}$, this property states –

$$\tilde{A} \cup \tilde{A} = \tilde{A}$$

$$\tilde{A} \cap \tilde{A} = \tilde{A}$$

Identity Property

For fuzzy set $\tilde{A}$ and universal set U, this property states –

$$\tilde{A} \cup \varphi = \tilde{A}$$

$$\tilde{A} \cap U = \tilde{A}$$

$$\tilde{A} \cap \varphi = \varphi$$

$$\tilde{A} \cup U = U$$

Transitive Property

Having three fuzzy sets $\tilde{A}$, $\tilde{B}$ and $\tilde{C}$, this property states –

$$If \tilde{A} \subseteq \tilde{B} \subseteq \tilde{C}, then \tilde{A} \subseteq \tilde{C}$$

Involution Property

For any fuzzy set $\tilde{A}$, this property states –

$$\overline{\overline{\tilde{A}}} = \tilde{A}$$

De Morgan's Law

This law plays a crucial role in proving tautologies and contradiction. This law states –

$$\overline{\tilde{A} \cap \tilde{B}} = \overline{\tilde{A}} \cup \overline{\tilde{B}}$$

$$\overline{\tilde{A} \cup \tilde{B}} = \overline{\tilde{A}} \cap \overline{\tilde{B}}$$

Fuzzy Logic - Membership Function

We already know that fuzzy logic is not logic that is fuzzy but logic that is used to describe fuzziness. This fuzziness is best characterized by its membership function. In other words, we can say that membership function represents the degree of truth in fuzzy logic.

Following are a few important points relating to the membership function:

- Membership functions were first introduced in 1965 by Lofti A. Zadeh in his first research paper "fuzzy sets".
- Membership functions characterize fuzziness (i.e., all the information in fuzzy set), whether the elements in fuzzy sets are discrete or continuous.
- Membership functions can be defined as a technique to solve practical problems by experience rather than knowledge.
- Membership functions are represented by graphical forms.
- Rules for defining fuzziness are fuzzy too.

Mathematical Notation

We have already studied that a fuzzy set $\tilde{A}$ in the universe of information U can be defined as a set of ordered pairs and it can be represented mathematically as –

$$A\tilde{}=\{(y,\mu A\tilde{}(y)) \mid y\in U\}$$

Here $\mu A\tilde{}(\cdot)$= membership function of $A\tilde{}$; this assumes values in the range from 0 to 1, i.e., $\mu A\tilde{}(\cdot)\in[0,1]$. The membership function $\mu A\tilde{}(\cdot)$ maps U to the membership space M.

The dot (·) in the membership function described above, represents the element in a fuzzy set; whether it is discrete or continuous.

Features of Membership Functions

We will now discuss the different features of Membership Functions.

Core

For any fuzzy set $A\tilde{}$, the core of a membership function is that region of universe that is characterize by full membership in the set. Hence, core consists of all those elements y of the universe of information such that,

$$\mu A\tilde{}(y)=1$$

Support

For any fuzzy set $A\tilde{}$, the support of a membership function is the region of universe that is characterize by a nonzero membership in the set. Hence core consists of all those elements y of the universe of information such that,

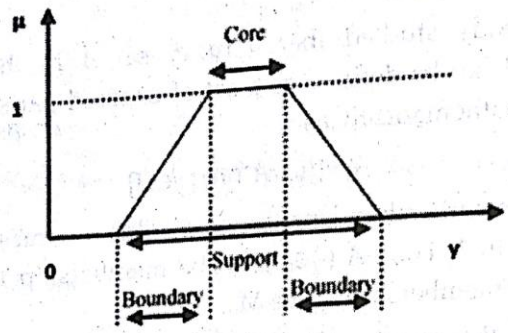
$$\mu A\tilde{}(y)>0$$

Boundary

For any fuzzy set $A^{\sim}$, the boundary of a membership function is the region of universe that is characterized by a nonzero but incomplete membership in the set. Hence, core consists of all those elements y of the universe of information such that,

$$1>\mu A^{\sim}(y)>0$$

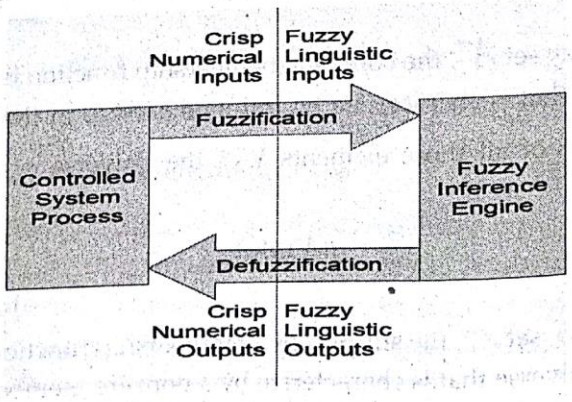
The core, support and boundary of a particular Fuzzy membership function is shown below



Features of Membership Function

Fuzzy Inference:

The figure given below show the process of fuzzy inference.



When making inferences, we want to clump the continuous numerical values into sets.

Fuzzification

The process of translating the measured numerical values into fuzzy linguistic values is called fuzzification. In other words, fuzzification is where membership functions are applied, and the degree of membership is determined.

Following are the two important methods of fuzzification –

Support Fuzzification(s-fuzzification) Method

In this method, the fuzzified set can be expressed with the help of the following relation –

$$A^{\sim}=\mu 1Q(x1)+\mu 2Q(x2)+...+\mu nQ(xn)$$

Here the fuzzy set $Q(xi)$is called as kernel of fuzzification. This method is implemented by keeping μi constant and xi being transformed to a fuzzy set Q(xi)

Grade Fuzzification (g-fuzzification) Method

It is quite similar to the above method but the main difference is that it kept xiconstant and μi is expressed as a fuzzy set.

Defuzzification

Defuzzification is the process of producing a quantifiable result in fuzzy logic. The fuzzy inference will output a fuzzy result, described in terms of degrees of membership of the fuzzy sets. Defuzzification interprets the membership degrees in the fuzzy sets into a specific action or real-value.

We have already studied that the fuzzification process involves conversion from crisp quantities to fuzzy quantities. In a number of engineering applications, it is necessary to defuzzify the result or rather "fuzzy result" so that it must be converted to crisp result. Mathematically, the process of Defuzzification is also called "rounding it off".

The different methods of Defuzzification are described below:

Max-Membership Method

This method is limited to peak output functions and also known as height method. Mathematically it can be represented as follows –

$$\mu A\tilde{}(x*)>\mu A\tilde{}(x) for all x \in X$$

Here, $x*$ is the defuzzified output.

Centroid Method

This method is also known as the center of area or the center of gravity method. Mathematically, the defuzzified output $x*$ will be represented as –

$$x*=\int \mu A\tilde{}(x).xdx / \int \mu A\tilde{}(x).dx$$

Weighted Average Method

In this method, each membership function is weighted by its maximum membership value. Mathematically, the defuzzified output $x*$ will be represented as –

$$x*=\sum \mu A\tilde{}(\overline{x_i}).\ \overline{x_i}\ /\ \sum \mu A\tilde{}(\overline{x_i})$$

Mean-Max Membership

This method is also known as the middle of the maxima.

Mathematically, the defuzzified output $x*$

will be represented as –

$$x*=\sum i=1-n\,\overline{x_i}\,n$$

Advantages of Fuzzy Logic System

- This system can work with any type of inputs whether it is imprecise, distorted or noisy input information.
- The construction of Fuzzy Logic Systems is easy and understandable.
- Fuzzy logic comes with mathematical concepts of set theory and the reasoning of that is quite simple.
- It provides a very efficient solution to complex problems in all fields of life as it resembles human reasoning and decision making.
- The algorithms can be described with little data, so little memory is required.

Disadvantages of Fuzzy Logic Systems

- Many researchers proposed different ways to solve a given problem through fuzzy logic which lead to ambiguity. There is no systematic approach to solve a given problem through fuzzy logic.
- Proof of its characteristics is difficult or impossible in most cases because every time we do not get mathematical description of our approach.
- As fuzzy logic works on precise as well as imprecise data so most of the time accuracy is compromised.

Application

- It is used in the aerospace field for altitude control of spacecraft and satellite.
- It has used in the automotive system for speed control, traffic control.
- It is used for decision making support systems and personal evaluation in the large company business.
- It has application in chemical industry for controlling the pH, drying, chemical distillation process.
- Fuzzy logic are used in Natural language processing and various intensive applications in Artificial Intelligence.
- Fuzzy logic are extensively used in modern control systems such as expert systems.
- Fuzzy Logic is used with Neural Networks as it mimics how a person would make decisions, only much faster. It is done by Aggregation of data and changing into more meaningful data by forming partial truths as Fuzzy sets.

Exercis

1. What is the difference between symbolic and non-symbolic AI? Represent the following knowledge in semantic network.

   Robin is bird

   Clyde is a Robin

   Clyde owns a nest from spring 2010 to fall 2010

2. What do you mean by causal network? Explain it with practical application.

3. Represent the following sentences into a semantic network.

   Birds are animals.

   Birds have feathers, fly and lay eggs.

   Albatross is a bird.

   Donald is a bird.

   Tracy is an albatross.

4. What do you mean by marginalization in probability distribution? Consider in Nepal, 51% of adults are males and rests are females, consider one adult is randomly selected for a survey of drinking alcohol. It is found that 15% of males drink alcohol where as 2% of female drink alcohol. Now find the probability that the selected adult is a male.

5. What are conceptual graphs? Represent the following statements into conceptual graph.

6. Define the Model-Based and Case Based system. Discuss which system is suitable for the following problems.

   i. Electronic circuit testing

   ii. Legal Reasoning.

7. After your yearly checkup, the doctor has bad news and good news. The bad news is that you tested positive for a serious disease, and the test is 99% accurate (i.e. the probability of testing

positive given that you have the disease is 0.99, as the probability of testing negative if you don't have the disease). The good news is that this is a rare disease, striking only one in 10,000 people.

a. Why it is good news that the disease is rare?

b. What are the chances that you actually have the disease?

8. What do you mean by causal network? Explain it with practical application.

9. Briefly describe the approaches of knowledge representation with example.

10. Consider the following sentence:

[(food) = > party] V (drink = > party)] = > [(Food ^ drinks) = > party]

a. Convert the right hand and left hand sides of main implication into CNF.

b. Prove the validity of sentence using resolution.

11. The concept of probability can be illustrated in the context of a study of fatness in people aging between55-60 years who are seeking medical care at a particular pediatric practice. The population (sampling frame) includes all people who were seen in the practice in the past 12 months and is summarized below.

	Age (years)						Total
	55	56	57	58	59	60	
Man	432	379	501	410	420	418	2,560
Women	408	513	412	436	461	500	2,730
Totals	840	892	913	846	881	918	5,290

12. What is the probability of selecting a Man?

13. What is the probability of selecting a 57 year-old?

14. What is the probability of selecting a men who is 60 years of age?

15. What is the probability of selecting a people (men or women) who is at least 58 years of age?
16. Ranbir is a superstar. All superstars are rich. Rich mans have fast cars. Fast cars consume a lot of petrol. Prove that Ranbir's car consumes a lot of petrol.
17. All greedy leaders are autocrat. Autocrats are evils. Ram is a greedy leader. Hari is a honest leader. Prove that Ram is evil.
18. All students of this class takes a course in BSC IT. BSC IT students are good in AI. Ritu is student of this class. Prove that Ritu is good in AI.

Lab Manual

Experiment 1

Rule Based system in prolog.

Problem statement: Consider the following family relation in prolog as described below:

Representing Facts and Rules

A Prolog program consists of a sequence of facts and rules i.e.Everything in *Prolog* is defined in terms of two constructs: the *fact* and the *rule*.

Facts

A fact states that a certain tuple of values satisfies a predicate *unconditionally*.a fact is a predicate name followed by an argument list and a dot sign, ".". It represents a relation among the arguments, or a property of a (single) argument.

For example, "George is funny" can be represented as a Prolog fact:

funny(george).

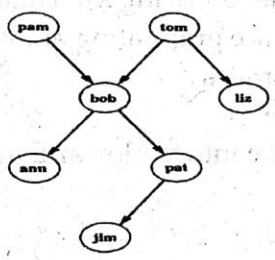
"George likes swimming" can be represented as

likes(george, swimming).

Similarly,The fact thatTom is a parent of Bob can be written in Prolog as:

parent(tom, bob).

Here we choose parent as the name of a relation; tom and bob are its arguments.



Family Tree

The whole tree can be represented in prolog by:

parent(pam, bob).
parent(tom, bob).
parent(tom, liz).
parent(bob, ann).
parent(bob, pat).
parent(pat, jim).

Queries: If we can ask the question to prolog, it will search the answer for us as follows:

?- parent(bob, pat).

Prolog will answer:

Yes.

?- parent(liz, pat).

No.

?- parent(X, liz).

X= tom

Representing Rules:

The syntax for rule is

Head:-body.

A rule can be seen as a "conditional fact". "Susie likes swimming if George likes swimming" is represented as a Prolog rule:

likes(susie, swimming) :- likes(george, swimming).

A rule can have multiple (co-existing) conditions,separated by ",". For example, "Mary likes whatever Susie and George like" is represented as:

likes(mary, X) :- likes(susie, X), likes(george, X).

Note: Prolog symbol	meaning
,	and
;	or
:-	only if
not	not

Example Let we have the following information given

- All people who are graduating are happy
- Happy people smiles
- Ram is graduating

Represent this knowledge into prolog and ask the questions

"Is ram smiles?

Solution:

```
%prolog source code
    Happy(X):-graduating(X).
    Smiles(X):-happy(X).
    Graduating(ram).
Queries
? Smiles(ram)
        yes
```

✿✿✿

UNIT 5

Machine Learning

OUTLINE

- ✓ **Introduction**
  - Machine Learning and Artificial Intelligence
  - General Machine Learning agent Framework or architecture
  - Types of Machine Learning
- ✓ **Statistical-based Learning: Naive Bayes Model**
- ✓ **Neural Networks**
  - Components of a Neuron
  - Activation Functions
  - Types of Feed Neural Networks

Introduction

Machine learning is a sub field of Artificial Intelligence that provides computer the ability to learn and improve its learning from experience without being written rules explicitly.

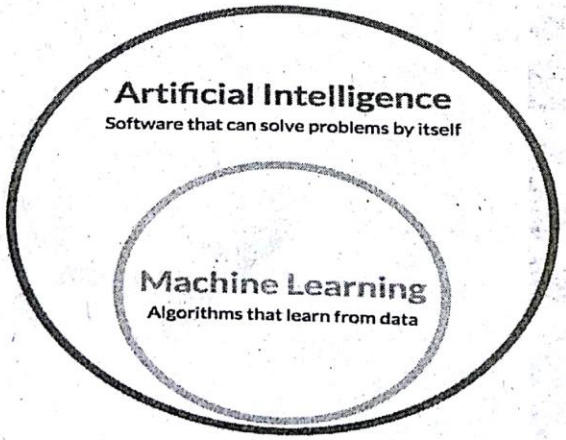
The machine start learning with observations or data, in order to look for patterns in data and make better decisions in the future based on the examples that it has been provided.

Some of the successful application of machine learning are:

- Computer Vision processing
- Natural Language processing
- Forecasting (e.g., stock market trends)
- Pattern recognition
- Games
- Expert systems and so on.

Machine Learning and Artificial Intelligence

Artificial Intelligence refers to a **very large** field of research that encompasses a number of techniques aimed at developing *computers that can learn and solve problems* where as **Machine Learning** is the field of Artificial Intelligence concerned with *learning from data on its own.*



(source: https://www.experfy.com)

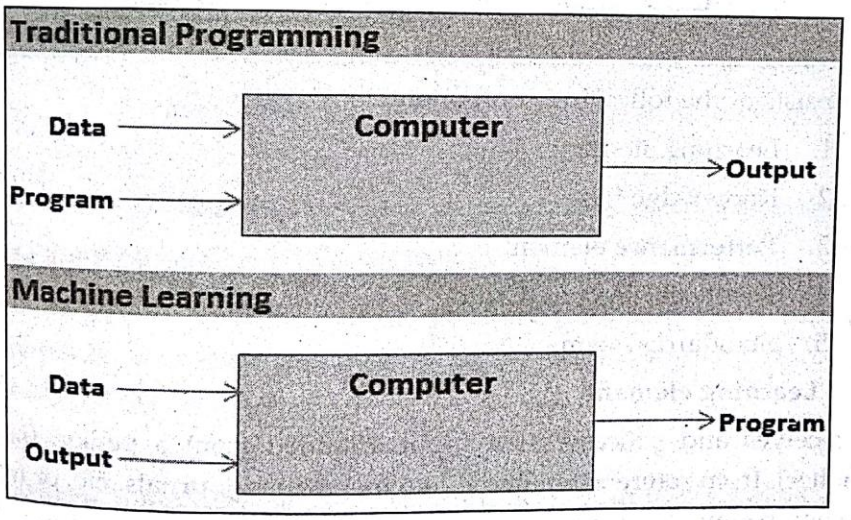
Machine Learning Defined

According to Tom Mitchell "A computer program is said to learn from experience *E* with respect to some class of tasks *T* and performance measure *P* if its performance at tasks in *T*, as measured by *P*, improves with experience *E*".

Machine learning research is part of research on artificial intelligence, seeking to provide knowledge to computers through data, observations and interacting with the world. That acquired knowledge allows computers to correctly generalize to new settings (Yoshua Bengio, **Turing Award Winner-2019)**

Machine learning is the science of getting computers to act without being explicitly programmed, but instead letting them learn a few tricks on their own (Danko Nikolic, **Max-Plank Institute)**

The difference between traditional programming and machine learning can be illustrated with following figures.

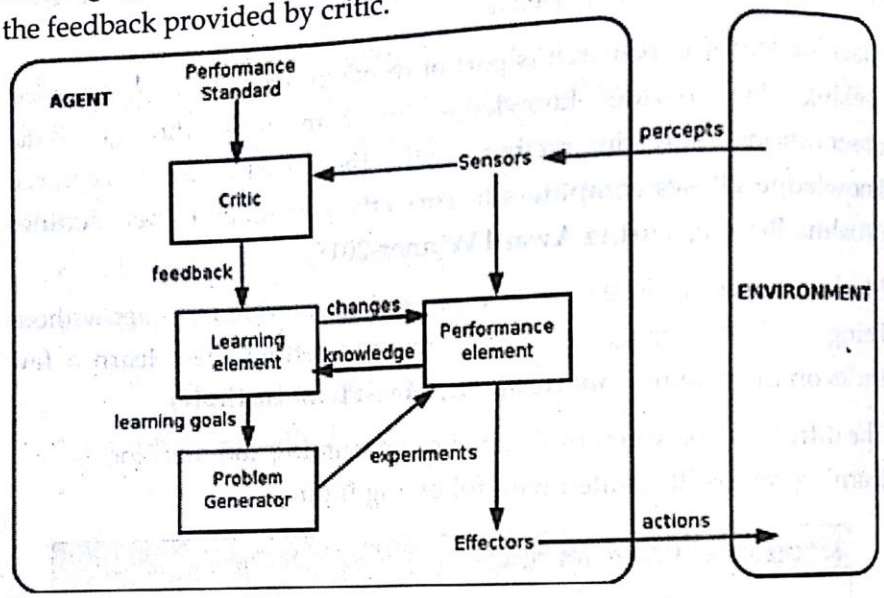


(Source: *www.datasciencecentral.com)*

Machine Learning is the science of getting computers to learn and act like humans do, and improve their learning over time in autonomous fashion, by feeding those data and information in the form of observations and real-world interactions.

General Machine Learning agent Framework or architecture

The following figure represents the general steps involved in machine learning agent. It learns form inputs and corrects its decision based on the feedback provided by critic.



It consists of the following components.

1. Learning element
2. Knowledge base
3. Performance element
4. Feedback element (Critic)
5. Standard system.

1. Learning element

It receives and processes the input obtained from a person (i.e. a teacher), from reference material like magazines, journals, etc, or from the environment at large.

2. Knowledge base

This is somewhat similar to the database. Initially it may contain some basic knowledge. Thereafter it receives more knowledge which may be new and so be added as it is or it may replace the existing knowledge.

3. **Performance element**

It uses the updated knowledge base to perform some tasks or solves some problems and produces the corresponding output.

4. **Feedback element**

It is receiving the two inputs, one from learning element and one from standard (or idealized) system. This is to identify the differences between the two inputs. The feedback is used to determine what should be done in order to produce the correct output.

5. **Standard system**

It is a trained person or a computer program that is able to produce the correct output. In order to check whether the machine learning system has learned well, the same input is given to the standard system. The outputs of standard system and that of performance element are given as inputs to the feedback element for the comparison. Standard system is also called idealized system.

Why Learning?

- To Understand and improve efficiency of human learning.
- To discover new things or structure that is unknown to humans.

Example: finding the particular pattern among the many patterns

Types of Machine Learning

The strategies for learning can be classified according to the amount of inference the system has to perform on its training data. In increasing order we have:

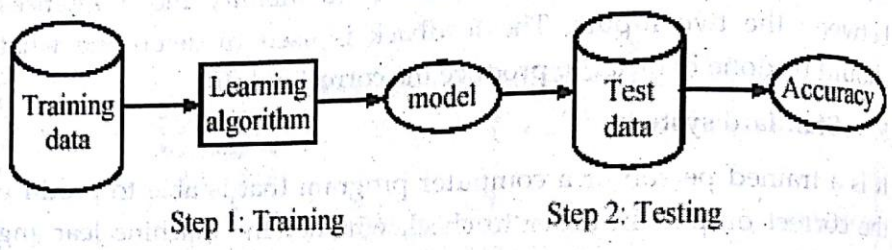
Supervised Learning

The system is supplied with a set of training examples consisting of inputs and corresponding outputs, and is required to discover the relation or mapping between them.

In supervised learning, the machine is presented with inputs together with the target (teacher signal) outputs. Then, the machine network tries to produce an output as close as possible to the target signal by

adjusting the values of internal weights. The most common supervised learning method is the "error correction method".

Error correction method is used for networks which their neurons have discrete output functions. Neural networks are trained with this method in order to reduce the error (difference between the network's output and the desired output) to zero.

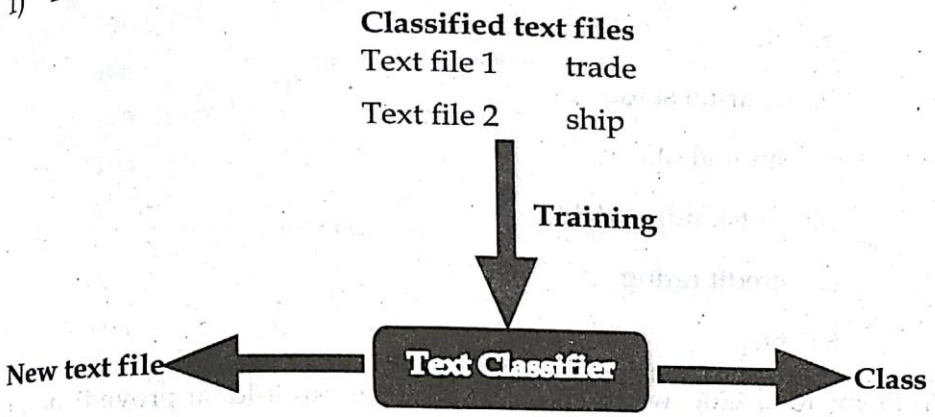


Classification is an example of supervised learning. It is a two-step process:

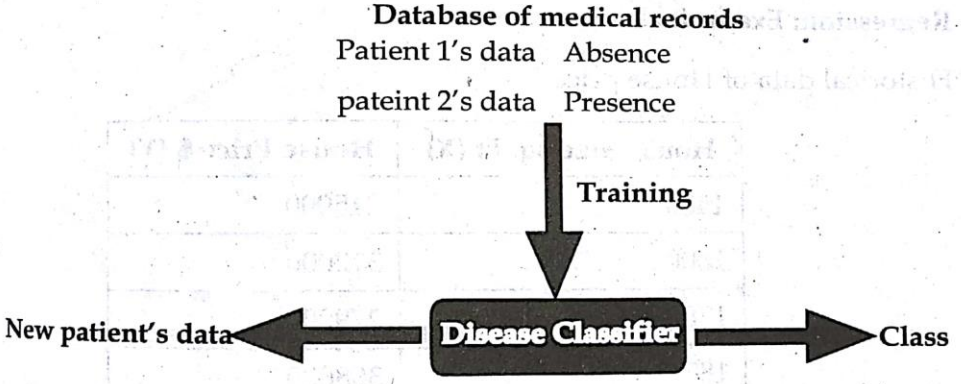
- Model construction: describing a set of predetermined classes
  - Each tuple/sample is assumed to belong to a predefined class, as determined by the class label attribute
  - The set of tuples used for model construction is training set
  - The model is represented as classification rules, decision trees, or mathematical formulae
- Model usage: for classifying future or unknown objects
  - Estimate accuracy of the model
    - The known label of test sample is compared with the classified result from the model
    - Accuracy rate is the percentage of test set samples that are correctly classified by the model
    - Test set is independent of training set, otherwise over-fitting will occur
  - If the accuracy is acceptable, use the model to classify data tuples whose class labels are not known

Examples

1) Text Classification



2) **Medical Disease classification**



3) **An emergency room in Hospital:**

An emergency room in a hospital measures 17 variables (e.g., blood pressure, age, etc) of newly admitted patients.

A decision is needed: whether to put a new patient in an intensive-care unit.

Due to the high cost of ICU, those patients who may survive less than a month are given higher priority.

Problem: to predict high-risk patients and discriminate them from low-risk patients.

4) Cridit card fraud analysis

A credit card company receives thousands of applications for new cards. Each application contains information about an applicant,

a. age

b. Marital status

c. annual salary

d. outstanding debts

e. credit rating

f. etc.

Problem: to decide whether an application should approved, or to classify applications into two categories, approved and not approved

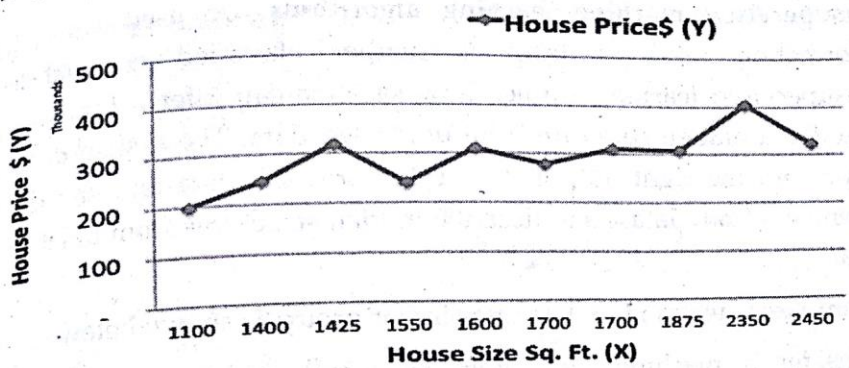
Regression: Example

Historical data of House price

House Size sq. Ft (X)	House Price $ (Y)
1400	245000
1600	312000
1700	279000
1875	308000
1100	199000
1550	219000
2350	405000
2450	324000
1425	319000
1700	255000

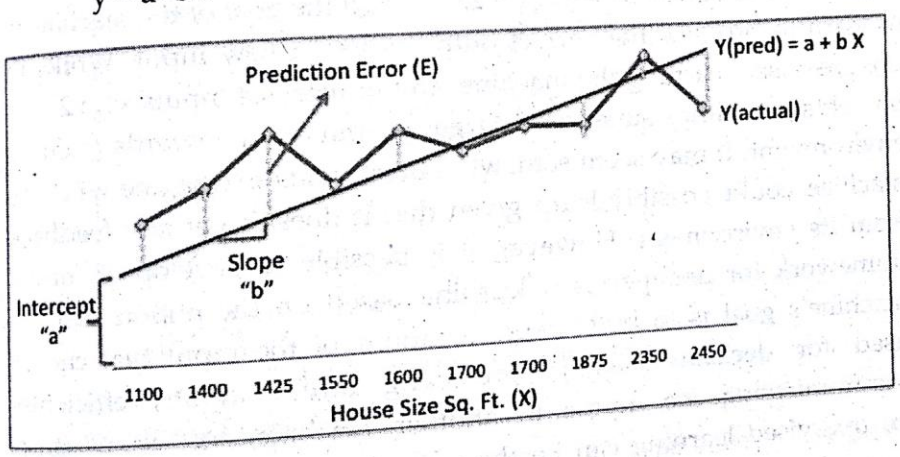
Problem is: for a new house, with its size (X), what will its price (Y) be?

The graphical plot of above data looks like:



Here we try to fit straight line in red color with the equations:

$$y' = a + bx$$



Optimal value of a and b

- we need to find a line with optimal values of a,b (called weights) that best fits the historical data by reducing the prediction error and improving prediction accuracy.
- So, our objective is to find optimal a, b that minimizes the error between actual and predicted values of house price.
- You can find the formula for *a and b*

$$b = \frac{n\,\Sigma xy - \Sigma x \Sigma y}{n\Sigma x^2 - (\Sigma x)^2}$$

$$a = \bar{y} - b\,\bar{x}$$

Unsupervised Learning

Unsupervised machine learning algorithms are used when the information used to train is neither classified nor labeled. Unsupervised learning studies how systems can infer a function to describe a hidden structure from unlabeled data. The system doesn't figure out the right output, but it explores the data and can draw inferences from datasets to describe hidden structures from unlabeled data.

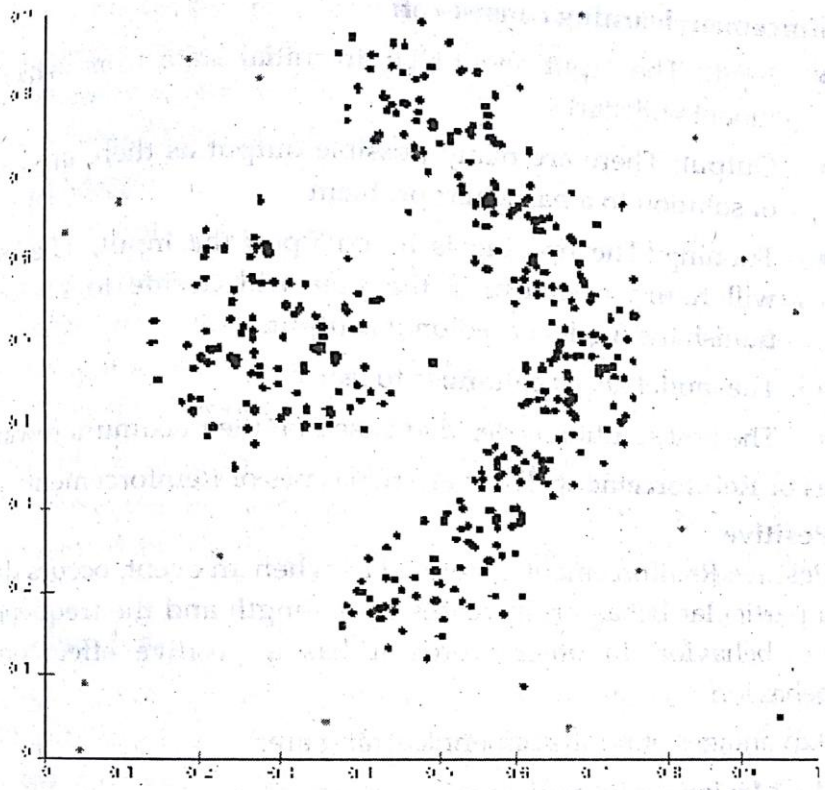
Examples: Clustering of data into similar group as shown below.

Consider a machine (or living organism) which receives some sequence of inputs x1, x2, x3, . . ., where xt is the sensory input at time t. In supervised learning the machine is given a sequence of input & a sequence of desired outputs y1, y2, . . . , and the goal of the machine is to learn to produce the correct output given a new input. While, in unsupervised learning the machine simply receives inputs x1, x2, . . ., but obtains neither supervised target outputs, nor rewards from its environment. It may seem somewhat mysterious to imagine what the machine could possibly learn given that it doesn't get any feedback from its environment. However, it is possible to develop of formal framework for unsupervised learning based on the notion that the machine's goal is to build representations of the input that can be used for decision making, predicting future inputs, efficiently communicating the inputs to another machine, etc. In a sense, unsupervised learning can be thought of as finding patterns in the data above and beyond what would be considered pure unstructured noise.

Semi-supervised machine learning:

Semi-supervised machine learning algorithms fall somewhere in between supervised and unsupervised learning, since they use both labeled and unlabeled data for training – typically a small amount of labeled data and a large amount of unlabeled data. The systems that use this method are able to considerably improve learning accuracy. Usually, semi-supervised learning is chosen when the acquired labeled data requires skilled and relevant resources in order to train it

learn from it. Otherwise, acquiring unlabeled data generally doesn't require additional resources.



(Source: www.medium.com)

Reinforcement Learning

Reinforcement learning is an area of Machine Learning. Reinforcement. It is about taking suitable action to maximize reward in a particular situation. It is employed by various software and machines to find the best possible behavior or path it should take in a specific situation. Reinforcement learning differs from the supervised learning in a way that in supervised learning the training data has the answer key with it so the model is trained with the correct answer itself whereas in reinforcement learning, there is no answer but the reinforcement agent decides what to do to perform the given task. In the absence of training dataset, it is bound to learn from its experience.

Example: The problem is as follows: We have an agent and a reward, with many hurdles in between. The agent is supposed to find the best

possible path to reach the reward. The following problem explains the problem more easily.

Reinforcement learning consists of:

- Input: The input should be an initial state from which the model will start
- Output: There are many possible output as there are variety of solution to a particular problem
- Training: The training is based upon the input, The model will return a state and the user will decide to reward or punish the model based on its output.
- The model keeps continues to learn.
- The best solution is decided based on the maximum reward.

Types of Reinforcement: There are two types of Reinforcement:

1. **Positive**

   Positive Reinforcement is defined as when an event, occurs due to a particular behavior, increases the strength and the frequency of the behavior. In other words it has a positive effect on the behavior.

   Advantages of reinforcement learning are:

   ✓ Maximizes Performance

   ✓ Sustain Change for a long period of time

   Disadvantages of reinforcement learning:

   ✓ Too much Reinforcement can lead to overload of states which can diminish the results

2. **Negative**

   Negative Reinforcement is defined as strengthening of a behavior because a negative condition is stopped or avoided.

   Advantages of reinforcement learning:

   ✓ Increases Behavior

   ✓ Provide defiance to minimum standard of performance

   Disadvantages of reinforcement learning:

   ✓ It Only provides enough to meet up the minimum behavior

Application of Reinforcement Learning (RL)

A variety of different problems can be solved using Reinforcement Learning. Because RL agents can learn without expert supervision, the type of problems that are best suited to RL are complex problems where there appears to be no obvious or easily programmable solution. Two of the main ones are:

Game playing

It determining the best move to make in a game often depends on a number of different factors, hence the number of possible states that can exist in a particular game is usually very large. To cover this many states using a standard rule based approach would mean specifying an also large number of hard coded rules.

RL cuts out the need to manually specify rules, agents learn simply by playing the game. For two player games such as backgammon, agents can be trained by playing against other human players or even other RL agents.

Control problems

Such as elevator scheduling. Again, it is not obvious what strategies would provide the best, most timely elevator service. For control problems such as this, RL agents can be left to learn in a simulated environment and eventually they will come up with good controlling policies. Some advantages of using RL for control problems is that an agent can be retrained easily to adapt to environment changes, and trained continuously while the system is online, improving performance all the time.

Statistical-based Learning: Naive Bayes Model

Basics of Probability

- P(event X) is calculated as:
  - ✓ Numbers of times X occurs / total number of events in sample spaces
  - ✓ P(1)= 1/6 in case of rolling dice.

- Joint probability consider two events A and B that occurs.
  - ✓ P(A ^B)= P(A)*p(B) if the events are independent.
  - ✓ P(A^B)=P(A/B)*P(B)

Bayes Theorem:

$$P(A\backslash B) = \frac{P(B|A)\,P(A)}{P(B)}$$

Where,

P (A | B) is posterior probability of A conditioned on B.

(A) is priori probability of hypothesis A.

P (B | A) is a posterior probability of B conditioned on A.

P (B) is a prior probability of event B.

Naïve Bayes Classifiers:

- To classify an instance x, it calculates
  - ✓ *P(c/x)= p(x/c)*p(c)/p(x)*
- Here p(x/c) is likelihood and other two are prior probability.
- These all probability are calculated form training samples using frequency.
- To make classification decision for event x into either c_1 or c_2 (can be extended to c_n) then,
  - ✓ Calculate $P(c_1/x)$ and $P(c_2/x)$
  - ✓ If $P(c_1/x)>P(c_2/x)$ then x is assigned to class c_1 otherwise assigned to class c_2

Example: Spam Email Filtering

Let us consider that we have following dataset:

D.N.	Massage	Class
1	Send us your password	Spam
2	Send us your review	Ham
3	Send us your account	Spam
4	Review your password	Ham

5	Review us	Spam
6	Send your password	spam
7	Send us review	ham

- How to estimate p(x/c)?
- Naïve Bayes assume the independence between events.
- For instance $x=\{x_1,x_2,x_3.....x_n\}$
  - $p(x/c)=p(x_1,x_2,x_3 ..x_n/C)$

$=P(x_1/c)*p(x_2/c)*p(x_3/c)...*p(x_n/c)$

P(spam/ review your account)
=p(review your account/spam)*p(spam)/ p(RYA)
=p(review/spam)*p(your/spam)*p(account/spam) *p(spam)

P(ham/review your account)
=p(review your account/ham)*p(ham)/p(RYA)
p(review/ham)*p(your/ham)*p(account/ham)*p(ham)

Probability table	Spam	Ham
send	3/4	1/3
Your	¾	2/3
Password	2/4	1/3
Review	1/4	3/3
Us	3/4	1/3
Account	1/4	1/3

Prior probability:
P(spam)=4/6, P(ham)=2/6

P(spam/m)=p(review/spam)*p(your/spam)*p(account/spam)*p(spam) =1/4*3/4*1/4*4/6=0.03125

P(ham/m)=p(review/ham)*p(your/ham)*p(account/ham)*p(ham)
=3/3*2/3*1/3*2/6=0.07438

Since p(m/ham) is greater than p(m/spam), the massage is assigned to class *ham*.

Problem with Naïve Bayes:

- **Zero frequency problem**
  - ✓ When a term is not present in the training set (spam or non spam) then its probability will be zero.
  - ✓ New email: "review password now" here the word now is not in our training sample (neither in spam nor in ham).
  - ✓ Its probability is zero. Hence it makes whole probability zero.
  - ✓ Solution is Laplace Smoothing (add 1 to all count)
- **Attribute independence:**
  - ✓ Naïve Bayes assume the independence between attributes or word during the calculation of conditional probability which may not true in all cases.

Neural Networks

Artificial neural network is non-linear, parallel, distributed, and highly connected network having capability of adaptively, self-organization, fault tolerance, evidential response and Very Large Scale Integration (VLSI) implementation, which closely resembles with physical nervous system.

Neural networks are composed of simple elements operating in parallel. These elements are inspired by biological nervous systems. As in nature, the network function is determined largely by the connections between elements. We can train a neural network to perform a particular function by adjusting the values of the connections (weights) between elements.

Physical nervous system is highly parallel, distributed information processing system having high degree of connectivity with capability of self-learning. Human nervous system contains about 10 billion neurons with 60 trillions of interconnections. These connections are modified based on experience. Typical physical neuron and artificial neural network are shown in figures.

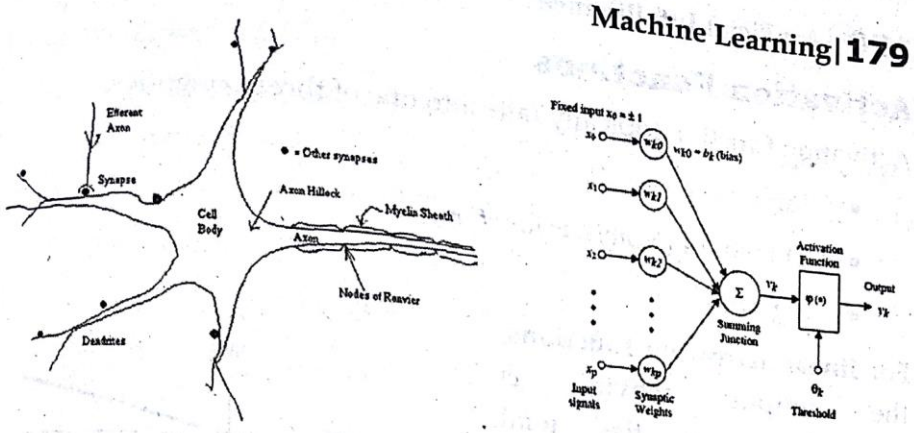


Figure: Biological Neuron and Mathematical Neurons

Components of a Neuron

Inputs:

The numerical value x1, x2, x3, …….xn are the inputs to neuron.

Each link has weight associated with it which determines strength and sign of the connection.

Activation function:

A function which is used to derive output activation from the input activations to a given node is called activation function.

Bias Weight:

Bias weight is used to set the threshold for a unit. Unit is activated when the weighted sum of real inputs exceeds the bias weight.

The calculation for a single neuron goes as follows:

Each unit first computes a weighted sum of it's inputs:

$$in_i = \sum_{J=0}^{n} W_{j,i}\, x_j$$

Then it applies activation function g to this sum to derive the output:

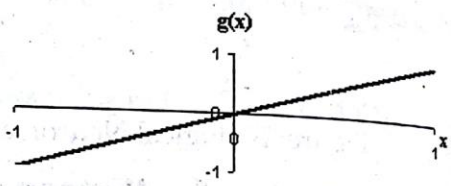
$$o_i = g(in_i) = g\left(\sum_{J=0}^{n} W_{j,i}\, x_j\right)$$

Activation Functions

Activation function typically falls into one of three categories:

- Linear
- Threshold (*Heaviside function*)
- Sigmoid

For **linear activation functions**, the output activity is proportional to the total weighted output.

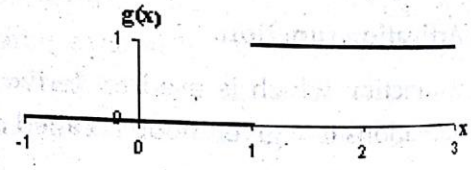


$g(x) = k\,x + c$, where k and x are constant

For **threshold activation functions**, the output are set at one of two levels, depending on whether the total input is greater than or less than some threshold value.

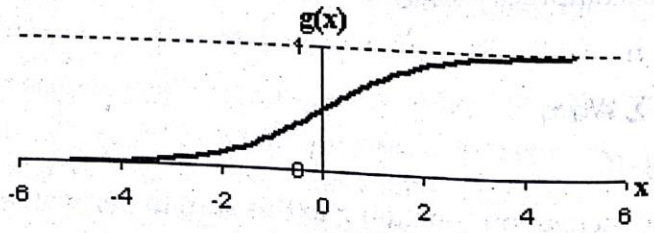
$g(x) = 1$ if $x >= k$

$= 0$ if $x < k$



For **sigmoid activation functions**, the output varies continuously but not linearly as the input changes. Sigmoid units bear a greater resemblance to real neurons than do linear or threshold units. It has the advantage of differentiable.

$g(x) = 1/\ (1 + e^{-x})$



Piecewise linear function

- A piecewise linear function is a real-valued function defined on the real numbers or a segment there of, whose graph is composed of straight-line sections.
- For Example:

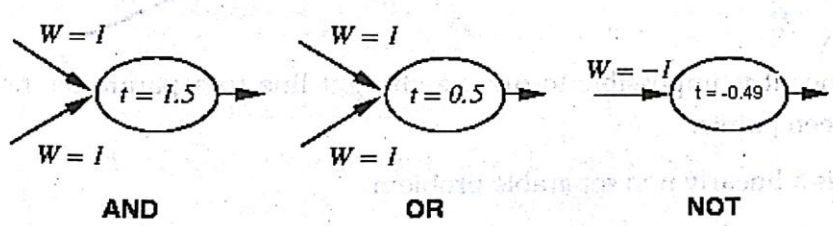
$$g(x) = x-3 \quad \text{if } x<0$$
$$= x+1 \quad \text{if } 0<=x<3$$
$$=x+6 \quad \text{if } x>=3$$

Ramp Activation Function

The activation function defined as the positive part of its argument:

- $f(x) = \max(0, x)$

Example of Simple Neurons: Building Logic Gates



Exercise: Try to build a neuron to perform Logical XOR. (Hint: it is not possible to build a neuron for this problem as it is a non-linearly separable.

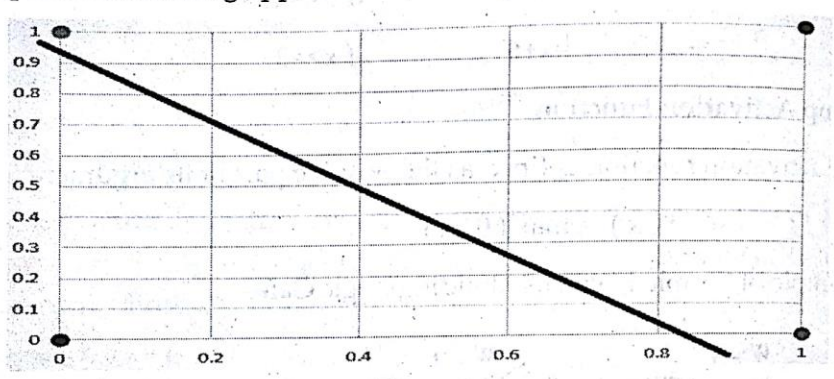
The XOR Problem in Neural Networks

The XOR Problem

The XOR, or "exclusive or", problem is a classic problem in ANN research. It is the problem of using a neural network to predict the outputs of XOR logic gates given two binary inputs. An XOR function should return a true value if the two inputs are not equal and a false value if they are equal. All possible inputs and predicted outputs are shown in figure 1.

Input 1	Input 2	Output
0	0	0
0	1	1
1	1	0
1	0	1

XOR is a classification problem and one for which the expected outputs are known in advance. It is therefore appropriate to use a supervised learning approach.

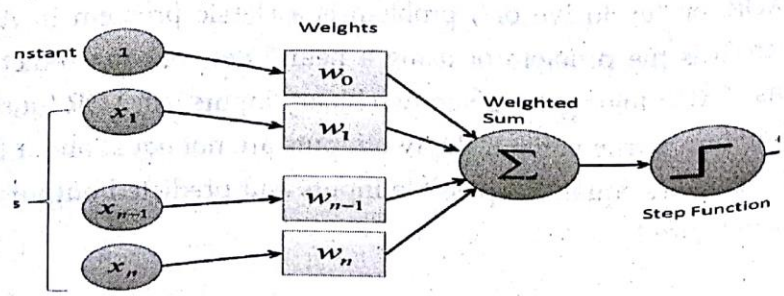


Since it is impossible to draw a straight line to separate the red and green points.

It is a linearly non separable problem.

Learning by Neural Network: Perceptron Learning Algorithm

It was originally developed by Frank Rosenblatt in the late 1950s. Training patterns are presented to the network's inputs; the output is computed. Then the connection weights wj are modified by an amount that is proportional to the product of the difference between the actual output, y, and the desired output, d, and the input pattern, x. The summarized algorithm is given below:



Variables Used

- y=f(x) denotes the *output* from the perceptron for an input vector x.
- D= (x1,d1), (x2,d2),.....(xn, dn) is the *training set* of n samples, where:
  - ✓ xj is the m-dimensional input vector.
  - ✓ dj is the desired output value of the perceptron for that input xj.

We show the values of the features as follows:

- $x_{j,i}$ is the value of the ith feature of the jth training *input vector*.

To represent the weights:

- w_i is the ith value in the *weight vector*, to be multiplied by the value of the ith input feature.
- $w_i(t)$ is the weight i at t time .

Steps in Algorithms

1. Initialize weights and threshold.
2. For each example *j* in our training set *D*, perform the following steps over the input $x_j$and desired output d_j:

   Calculate the actual output:

   ✓ $y(t) = g\ [w0(t)x0(t) + w1(t)x1(t) + \ldots + wn(t)xn(t)]$

   - Update the weights
     - $wi(t+1) = wi(t) + \alpha[d(t) - y(t)]xi(t)$, where $0 <= \alpha <= 1$ (learning rate) is a positive gain function that controls the adaption rate.

Examples: *Perceptron Learning algorithm (implement a neuron to simulate logical OR)*

Back propagation

It is a supervised learning method, and is an implementation of the **Delta rule**. It requires a teacher that knows, or can calculate, the desired output for any given input. It is most useful for feed-forward

networks (networks that have no feedback, or simply, that have no connections that loop). The term is an abbreviation for "backwards propagation of errors". Backpropagation requires that the activation function used by the artificial neurons (or "nodes") is differentiable.

Backpropagation networks are necessarily multilayer perceptrons (usually with one input, one hidden, and one output layer). In order for the hidden layer to serve any useful function, multilayer networks must have non-linear activation functions for the multiple layers: a multilayer network using only linear activation functions is equivalent to some single layer, linear network.

Algorithm:

Input:

D, a data set consisting of the training tuples and their associated target value;

l, the learning rate;

network, a multilayer feed forward network

Output: A trained neural network

Method:

Initialize all weights and biases in network;

while terminating condition is not satisfied

{

for each training sample X in *D*

{

// Propagate the inputs forward:

for each input layer j

$O_j = I_j$; //output of an input unit is its actual input value

for each hidden or output layer unit j

{

$I_j = \sum w_i O_i + \Theta j$; //compute the net input of unit j with respect to previous input layer, i

$O_j = \frac{1}{1+e^{-Ij}}$;

}

// compute the output of each unit j

// Backpropagate the errors:

for each unit j in the output layer

$Err_j = Oj(1 - O_j)(T_j - O_j)$; // compute the error

for each unit j in the hidden layers

$Err_j = Oj(1 - O_j)\sum_k Err_k w_{jk}$; // compute the error

for each weight w_{ij} in network

{

$\Delta w_{ij} = (l)Err_j O_i$; // weight increment

$w_{ij} = w_{ij} + \Delta w_{ij}$; // weight update

}

for each bias _j in network

{

$\Delta \Theta j = (l)Err_j$; // bias increment

$\Theta_j = \Theta j + \Delta \Theta_j$; // bias update

}

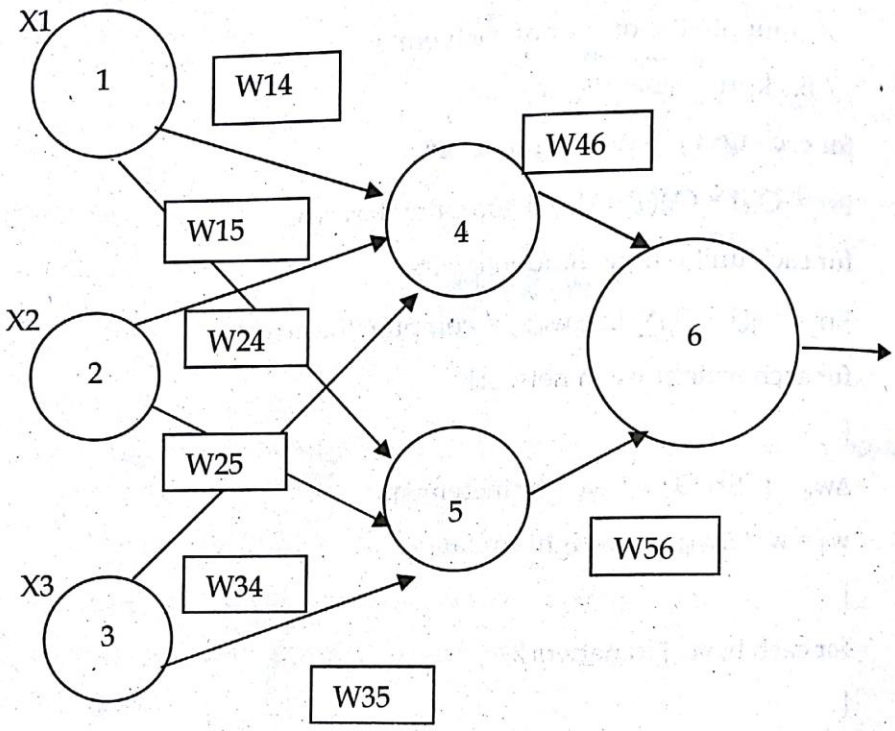
}

}

Note: There are a number of options in the design of a backprop system;

- Initial weights – best to set the initial weights (and all other free parameters) to random numbers inside a small range of values (say –0.5 to 0.5)
- Number of cycles – tend to be quite large for back prop systems
- Number of neurons in the hidden layer – as few as possible

Examples: Consider the following network



<u>We are tracing the BP for input (1,0,1) with label of 1.</u>

Now let us consider the initial weight and bias value as:

X1	X2	X3	W14	W15	W24	W25	W34	W35	W46	W56	θ1	θ2	θ3
1	0	1	0.2	-0.3	0.4	0.1	-0.5	0.2	-0.3	-0.2	-0.4	0.2	0.1

Now calculate the output for each node 4, 5 and 6 as follows:

Node j	Net input Ij	Output Oj
4	0.2+0-0.5-0.4=-0.7	0.332
5	-0.3+0+0.2+0.2=0.1	0.525
6	-0.3x0.332-0.2x0.525+0.1=-0.105	0.474

Then calculate the error at each node:

Node J	ErrJ
6	0.474(1-0.474)(1-0.474)=0.1311 (Oj(1-Oj)(Tj-Oj) for output layer)
5	0.525(1-0.525)(0.1311)(-0.2)=-0.0065 (Oj(1 - Oj)$\sum_k Err_k w_{jk}$ for each hidden layer)
4	0.332(1-0.332)(0.1311)(-0.3)=-0.0087 (Oj(1 - Oj)$\sum_k Err_k w_{jk}$ for each hidden layer)

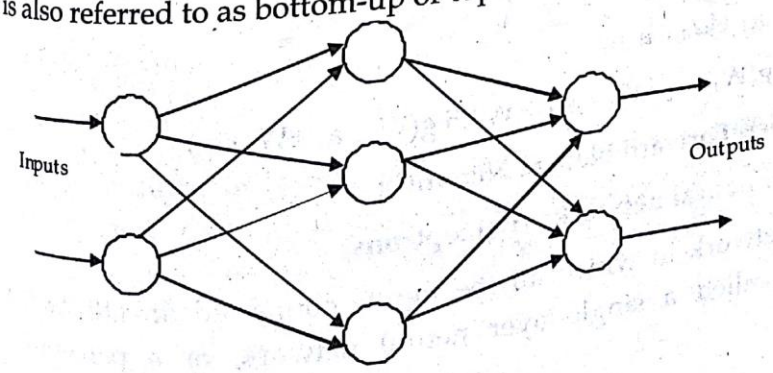
Now update the weight and bias as follows (using formulas: Δw_{ij} = $(l)Err_j O_i$, and $w_{ij} = w_{ij}+\Delta w_{ij}$) with learning rate ($l$)=0.9.

Weight or bias	Ne value
W46	-0.3+0.9×0.1311×0.332=-0.261
W56	-0.2+0.9×0.1311×0.525=-0.138
W14	0.2+0.9×-0.0087×1=0.192
And so on	

Types of Feed Neural Networks

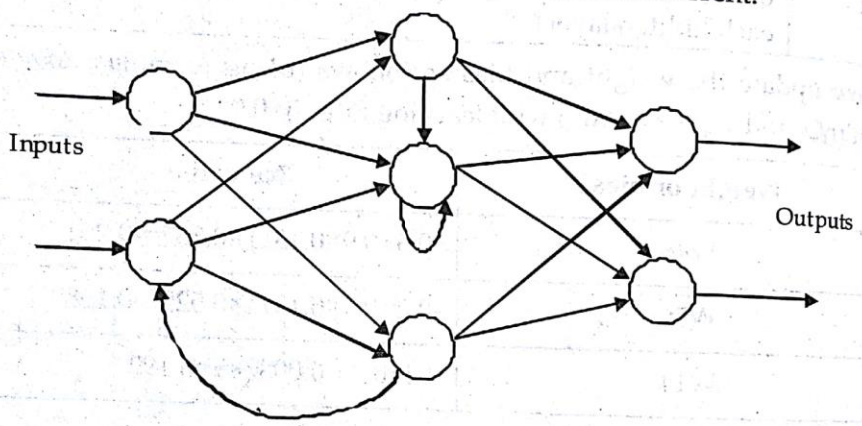
Feed-forward networks

Feed-forward ANNs allow signals to travel one way only; from input to output. There is no feedback (loops) i.e. the output of any layer does not affect that same layer. Feed-forward ANNs tend to be straight forward networks that associate inputs with outputs. They are extensively used in pattern recognition. This type of organization is also referred to as bottom-up or top-down.

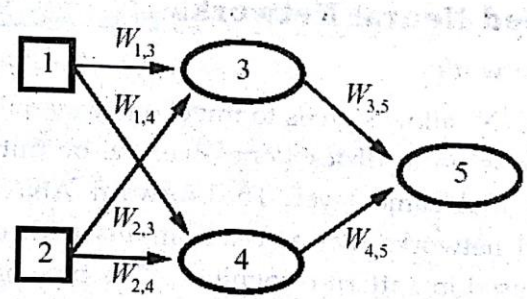


Feedback networks (Recurrent Networks)

Feedback networks (figure 1) can have signals traveling in both directions by introducing loops in the network. Feedback networks are very powerful and can get extremely complicated. Feedback networks are dynamic; their 'state' is changing continuously until they reach an equilibrium point. They remain at the equilibrium point until the input changes and a new equilibrium needs to be found. Feedback architectures are also referred to as interactive or recurrent.



Feed-forward example



Here;

$a5 = g(W_{3;5}\ a_3 + W_{4;5}\ a_4)$

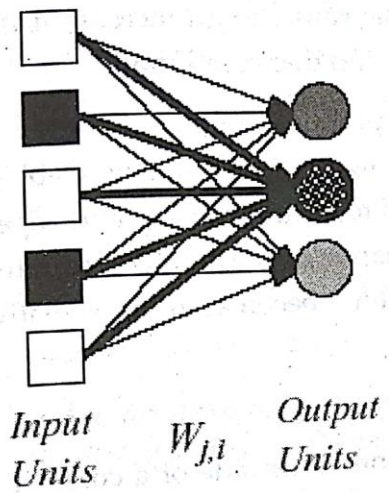
$= g(W_{3;5}\ g(W_{1;3}\ a_1 + W_{2;3}\ a_2) + W4;5\ g(W_{1;4}\ a_1 + W_{2;4}\ a_2)$

Types of Feed Forward Neural Network:

Single-layer neural networks (perceptrons)

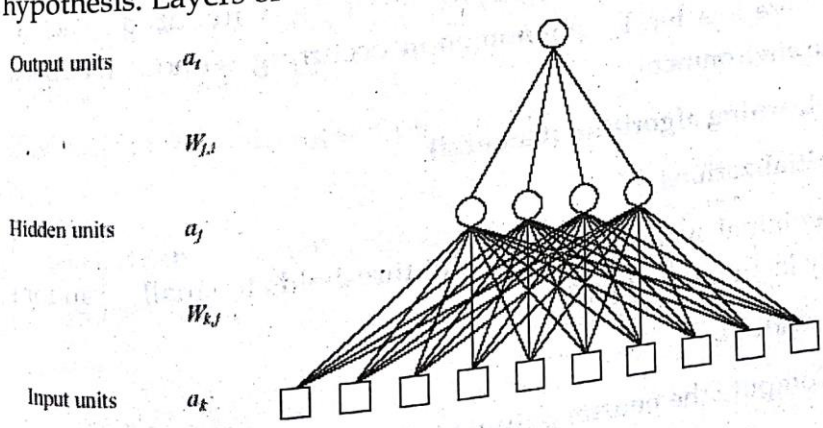
A neural network in which all the inputs connected directly to the outputs is called a single-layer neural network, or a perceptron

network. Since each output unit is independent of the others each weight affects only one of the outputs.



Multilayer neural networks (perceptrons)

The neural network which contains input layers, output layers and some hidden layers also is called multilayer neural network. The advantage of adding hidden layers is that it enlarges the space of hypothesis. Layers of the network are normally fully connected.



Once the number of layers, and number of units in each layer, has been selected, training is used to set the network's weights and thresholds so as to minimize the prediction error made by the network

Hebbian Learning:

Hebbian learning is a general principle that states that the synaptic efficacy between two neurons should increase if the two neurons are 'simultaneously' active, and decrease if not.

In 1949, Donald Hebb proposed one of the key ideas in biological learning, commonly known as **Hebb's Law**. Hebb's Law states that if neuron *i* is near enough to excite neuron *j* and repeatedly participates in its activation, the synaptic connection between these two neurons is strengthened and neuron *j* becomes more sensitive to stimuli from neuron *i*.

Hebb's Law can be represented in the form of two rules:

1. If two neurons on either side of a connection are activated synchronously, then the weight of that connection is increased.
2. If two neurons on either side of a connection are activated asynchronously, then the weight of that connection is decreased.

Hebb's Law provides the basis for learning without a teacher. Learning here is a **local phenomenon** occurring without feedback from the environment.

Hebbian learning algorithm (General)

Step 1: Initializations

Set initial synaptic weights and thresholds to small random values, say in an interval [0, 1].

Step2: Activation.

Compute the neuron output at iteration p

$$y_j(P) = \sum_{i=1}^{n} \times i(p)\, w_{ij}(p) - \theta_j$$

where n is the number of neuron inputs, and $Theta_j$ is the threshold value of neuron j.

Step 3: Learning.

Update the weights in the network

$$w_{ij}(p+1) = w_{ij(p)} + \Delta w_{ij}(p) + \Delta w_{ij(p)}$$

where $Dw_{ij}(p)$ is the weight correction at iteration p.

The weight correction is determined by the generalized activity product rule:

$$\Delta w_{ij}(p) = \varphi\, y_j(p)\, [\lambda\, x_i(p) - w_{ij(p)}]$$

Step 4: Iteration.

Increase iteration p by one, go back to Step 2.

Hebb's Algorithm (Simplified when $\alpha = 1$)

Step 0: initialize all weights to 0

Step 1: Given a training input, s, with its target output, t, set the activations of the input

units: $x_i = s_i$

Step 2: Set the activation of the output unit to the target value: $y = t$

Step 3: Adjust the weights: $w_i(new) = w_i(old) + x_i y$

Exercise: Construc a hebb's net to perform AND like operations.

Step 4: Adjust the bias (just like the weights): $b(new) = b(old) + y$

Pseudocode for Hebb's Net(Supervised)

```
Initialization; for i = 1 to n : {b = 0, wi = 0}
For each of the training sample s: to do
{
/* s is the input pattern, t the target output of the smaple */
   for i = 1 to n {xi := si}                /* set s to input units* /
   y := t                                   /* set y to the target * /
   for i = 1  to n {
        wi =  wi+ xi *y
   }    /* update weight */
   b := b + xi * y /*update bias*/
}
```

Problem: Construct a Hebb Net which performs like an AND function, that is, only

Logic function: AND/Bipolar input, bipolar outp

Input	Target
$(x_1, x_2, 1)$	1
(1 1 1)	-1
(1 -1 1)	-1
(-1 1 1)	-1
(-1 -1 1)	-1

Showing the first input/target pair leads to the following changes

$\Delta W_1 = x_1 * t = 1 * 1 = 1$

$\Delta W_2 = x_2 * t = 1 * 1 = 1$

$\Delta b = t = 1$

Therefore, $w_1 = w_1 + \Delta w_1 = 0 + 1 + 1$

$w_2 = w_2 + \Delta_2 = 0 = 1$

$b = b + \Delta b = 0 + 1 = 1$

Logic function : AND/Bipolar representation

After the first input, the separating line looks like the following:

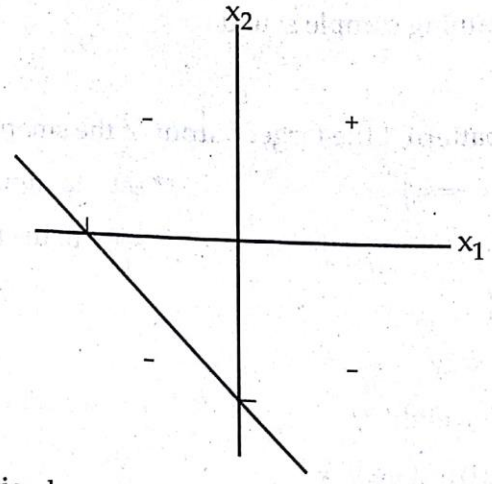


Figure: Decision boundary for the AND function using Hebb rule after first training pair (bipolar inputs and targets)

Logic function: AND/Bipolar representation

Input	Target
(x_1, x_2, 1)	1
(1 1 1)	– 1
(1 – 1 1)	– 1
(–1 1 1)	– 1
(–1 – 1 1)	– 1

Showing the second input/target pair leads to the following changes

$\Delta W1 = x_1 * t = 1 * 1 = -1$

$\Delta W2 = x2 * t = -1 * -1 = 1$

$\Delta b = t = 1$

Therefore, $w1 = w1 + \Delta w1 = 1 - 1 = 0$

$w2 = w2 + \Delta w2 = 1 + 1 = 2$

$b = b + \Delta b = 1 - 1 = 0$

Logic function: AND/Bipolar representation

After the first input, the separating line looks like the following:

- Bipolar Representation/AND function
- After two training examples, the separationg function learned is

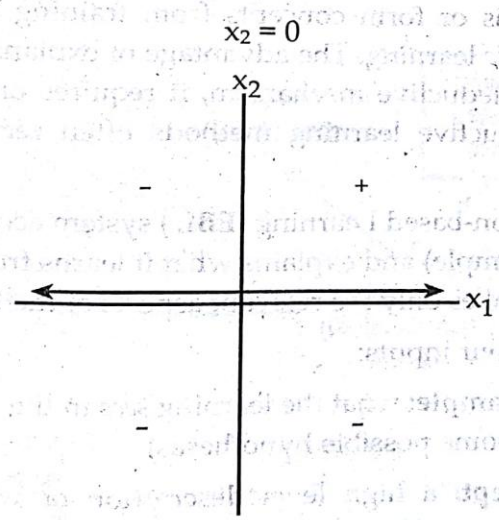


Figure: Decision boundary for bipolar AND function using Hobb rule after second training pattern (boundary is x_1 axis)

- Bipolar representation/AND function
- After four training examples the separating function learned is (this one is correct)

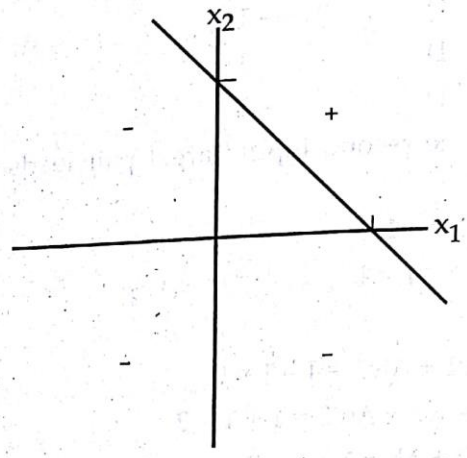


Figure: Decision boundary for bipolar and functions using Hebb rule after third training patters.

Explanation Based Machine Learning

Explanation-based learning (EBL) is a form of machine learning that exploits a very strong, or even perfect, domain theory to make generalizations or form concepts from training examples. This is a type of *analytic* learning. The advantage of explanation-based learning is that, as a deductive mechanism, it requires only a single training example (inductive learning methods often require many training examples)

An Explanation-based Learning (**EBL**) system accepts an example (i.e. a training example) and explains what it learns from the example. The **EBL** system takes only the relevant aspects of the training.

EBL accepts four inputs:

A training example: what the learning *sees* in the world. (specific facts that rule out some possible hypotheses)

A goal concept: a high level description of what the program is supposed to learn. (the set of all possible conclusions)

A operational criterion: a description of which concepts are usable. (criteria for determining which features in the domain are efficiently

recognizable, e.g. which features are directly detectable using sensors).

A domain theory: a set of rules that describe relationships between objects and actions ina domain. (axioms about a domain of interest)

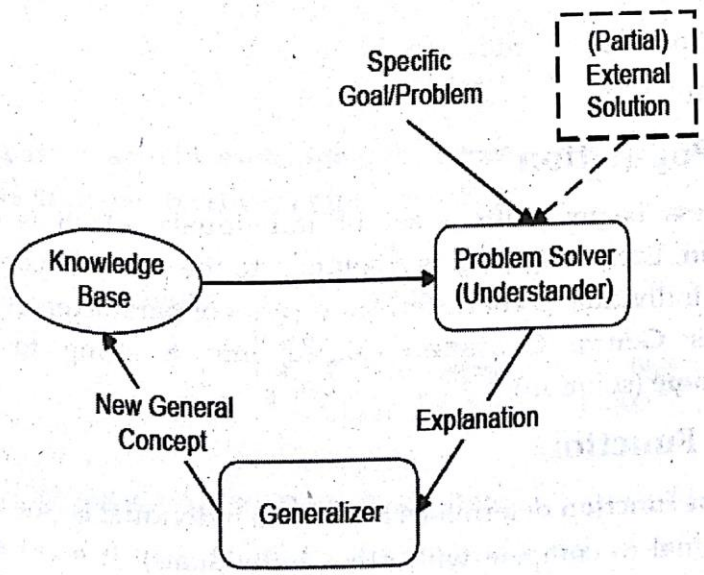
From this EBL computes a generalization of the training example that is sufficient not only to describe the goal concept but also satisfies the operational criterion.

This has two steps: Explanation: the domain theory is used to prune away all unimportant aspects of the training example with respect to the goal concept.

Generalisation: the explanation is generalized as far possible while still describing the goal concept.

An example of EBL using a perfect domain theory is a program that learns to play chess by being shown examples. A specific chess position that contains an important feature, say, "Forced loss of black queen in two moves," includes many irrelevant features, such as the specific scattering of pawns on the board. EBL can take a single training example and determine what the relevant features are in order to form a generalization.

The general architecture of EBL system is;



Learning by simulating Evolutions: Generic Algorithm

Evolution is change in the heritable characteristics of biological populations over successive generations. Evolutionary processes give rise to biodiversity at every level of biological organization, including the levels of species, individual organisms, and molecules.

The Darwin theory of Evolution explain natural selection as

- Individuals pass on traits to offspring
- Individuals have different traits
- Fittest individuals survive to produce more offspring
- Over time, variation can accumulate leading to new species

A genetic or evolutionary algorithm **applies the principles of evolution found in nature** to the problem of finding an optimal solution to a Solver problem. In a "genetic algorithm," the problem is encoded in a series of bit strings that are manipulated by the algorithm;

Five phases are considered in a genetic algorithm as shown in figures.

1. Initial population
2. Fitness function
3. Selection
4. Crossover
5. Mutation

Initial Population

The process begins with a set of individuals which is called a **Population**. Each individual is a solution to the problem you want to solve. An individual is characterized by a set of parameters (variables) known as **Genes**. Genes are joined into a string to form a **Chromosome** (solution)

Fitness Function

The **fitness function** determines how fit an individual is (the ability of an individual to compete with other individuals). It gives a **fitness**

score to each individual. The probability that an individual will be selected for reproduction is based on its fitness score.

Selection

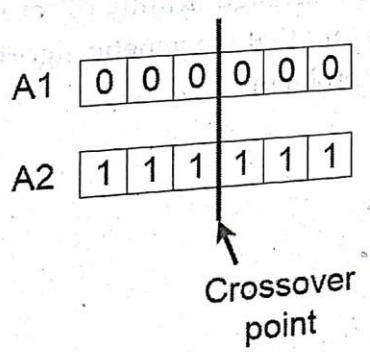
The idea of **selection** phase is to select the fittest individuals and let them pass their genes to the next generation.

Two pairs of individuals (**parents**) are selected based on their fitness scores. Individuals with high fitness have more chance to be selected for reproduction.

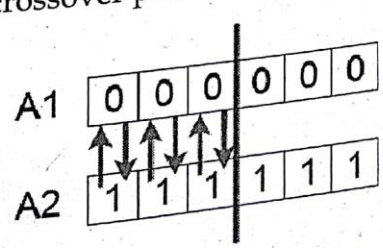
Crossover

Crossover is the most significant phase in a genetic algorithm. For each pair of parents to be mated, a **crossover point** is chosen at random from within the genes.

For example, consider the crossover point to be 3 as shown below.



Offspring are created by exchanging the genes of parents among themselves until the crossover point is reached.



The new offspring are added to the population.

A5	1	1	1	0	0	0

A6	0	0	0	1	1	1

Mutation

In certain new offspring formed, some of their genes can be subjected to a **mutation** with a low random probability. This implies that some of the bits in the bit string can be flipped.

Before Mutation

A5	1	1	1	0	0	0

After Mutation

A5	1	1	0	1	1	0

Termination

The algorithm terminates if the population has converged (does not produce offspring which are significantly different from the previous generation). Then it is said that the genetic algorithm has provided a set of solutions to our problem.

Exercis

1. What is machine learning? Explain the learning from Analogy and Instance based learning.
2. How concepts of most specific consistent hypothesis and most general consistent hypothesis are used in learning through examples. How generalization tree is maintained for these concepts.
3. Define Learning. Why learning frame work is required? Explain about learning frame with block diagram and examples.
4. Why do we require learning? Explain about learning framework with suitable block diagram and examples.
5. Define learning. Why learning frame work is required? Explain about learning frame work with block diagram and examples.
6. What is learning by induction? Explain inductive learning process with example.

Lab Manual

Experiment 1:

Write program for implementing Naive Bayes.

What is Naive Bayes algorithm?

It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as 'Naive'.

Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

Bayes theorem provides a way of calculating posterior probability P(c | x) from P(c), P(x) and P(x | c). Look at the equation below:

Likelihood Class Prior Probability

$$P(C|x) = \frac{P(x|c)\,P(c)}{P(x)}$$

Posterior Probability Predictor Prior Probability

$$P(c|x) = P(x_1 \mid c) \times P(x_2 \mid c) \times \ldots \times P(x_n \mid c) \times P(c)$$

Above,

- $P(c|x)$ is the posterior probability of *class* (c, *target*) given *predictor* (x, *attributes*).
- $P(c)$ is the prior probability of *class*.
- $P(x|c)$ is the likelihood which is the probability of *predictor* given *class*.
- $P(x)$ is the prior probability of *predictor*.

How it Works?

How Naive Bayes algorithm works?

Let's understand it using an example. Below I have a training data set of weather and corresponding target variable 'Play' (suggesting possibilities of playing). Now, we need to classify whether players will play or not based on weather condition. Let's follow the below steps to perform it.

Step 1: Convert the data set into a frequency table

Step 2: Create Likelihood table by finding the probabilities like Overcast probability = 0.29 and probability of playing is 0.64.

Weather	Play
Sunny	No
Overcast	Yes
Rainy	Yes
Sunny	Yes
Sunny	Yes
Overcast	Yes
Rainy	No
Rainy	No
Sunny	Yes
Rainy	Yes
Sunny	No
Overcast	Yes
Overcast	Yes
Rainy	No

Frequency Table		
Weather	No	Yes
Overcast		4
Rainy	3	2
Sunny	2	3
Grand Total	5	9

Likelihood table				
Weather	No	Yes		
Overcast		4	=4/14	0.29
Rainy	3	2	=5/14	0.36
Sunny	2	3	=5/14	0.36
All	5	9		
	=5/14	=9/14		
	0.36	0.64		

Step 3: Now, use Naive Bayesian equation to calculate the posterior probability for each class. The class with the highest posterior probability is the outcome of prediction.

Problem: Players will play if weather is sunny. Is this statement is correct?

We can solve it using above discussed method of posterior probability.

P(Yes | Sunny) = P(Sunny | Yes) * P(Yes) / P (Sunny)

Here we have P (Sunny |Yes) = 3/9 = 0.33, P(Sunny) = 5/14 = 0.36, P(Yes)= 9/14 = 0.64

Now, P (Yes | Sunny) = 0.33 * 0.64 / 0.36 = 0.60, which has higher probability.

Naive Bayes uses a similar method to predict the probability of different class based on various attributes. This algorithm is mostly used in text classification and with problems having multiple classes.

Python Code (Program code)

```
#Import Library of Gaussian Naive Bayes model
fromsklearn.naive_bayesimportGaussianNB
importnumpy as np

#assigning predictor and target variables
x= np.array([[-3,7],[1,5], [1,2], [-2,0], [2,3], [-4,0], [-1,1], [1,1], [-2,2], [2,7], [-4,1], [-2,7]])
Y=np.array([3, 3, 3, 3, 4, 3, 3, 4, 3, 4, 4, 4])

#Create a Gaussian Classifier
model=GaussianNB()

# Train the model using the training sets
model.fit(x,y)

#Predict Output
predicted= model.predict([[1,2],[3,4]])
print predicted
```

Output: ([3,4])

(Note: You may find the real time Nepali Spam classification project at http://csitgraduate.blogspot.com/p/nwomlds.html)

Experiment 2: Implement the Perceptron Learning algorithm for OR gate.

Solution:

The perceptron learning algorithm works as follows:

1. Intilalize the weights for all inputs (including bias)
2. Present the input xi to neural network
3. Calculate the outpput for the given input. say it is zi
4. update the weights as: w=w+a(actual-predicted output)xi
5. Repeat from step 2 until the convergance or number of iteration reached.

Let's implement perceptron for the inputs:

Inputs			Outputs
x1	x2	b	
o	0	1	0
1	0	1	1
0	1	1	1
1	1	1	1

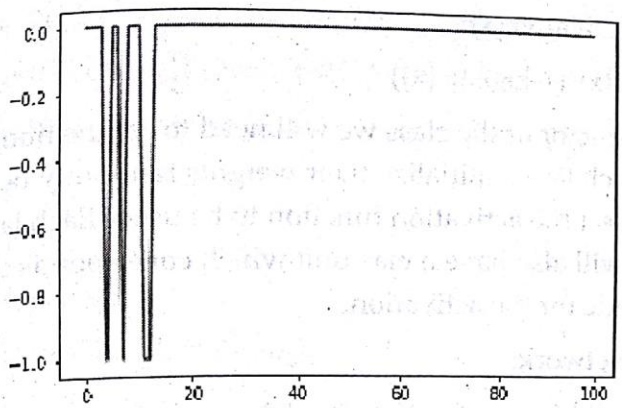
Python code(Program Code)

```
importnumpy as np
import random as rd
unit_step = lambda x: 0 if x < 0 else 1
train_data = [
  (np.array([0,0,1]), 0),
  (np.array([0,1,1]), 1),
```

```
  (np.array([1,0,1]), 1),
  (np.array([1,1,1]), 1),
]
w = np.random.rand(3)
out:w
//array([0.83364474, 0.84316914, 0.79095823])
errors = []
eta = 0.2
n = 100
fori in range(n):
x, expected = rd.choice(train_data)
result = np.dot(w, x)
error = expected - unit_step(result)
errors.append(error)
  w += eta * error * x
for x, _ in train_data:
result=np.dot(x,w)
  z=unit_step(result)
print("{}->{}".format(x[:2], z))
```

Output:

```
[0 0]->0
[0 1]->1
[1 0]->1
[1 1]->1
importmatplotlib.pyplot as plt
plt.plot(errors)
[<matplotlib.lines.Line2D at 0x2e761f72f60>]
```



Here the plot of error show that after the 20th iteration, it becomes zero and remains stable until 100th iterations

Experiment 3: Implement the back-propagation algorithm for XOR gate

The most common algorithm used to train MLPs is the back-propagation algorithm. It has two phases:

1. A forward pass, in which the training data is run through the network to obtain it's output
2. A backward pass, in which, starting from the output, the errors for each neuron are calculated and then used to adjust the weight of the network.

That's the rough summary of the algorithm, so let's start implementing it.

```
# First, define our activation functions and their derivatives, using
numpy.
importnumpyasnp
deftanh(x):
returnnp.tanh(x)
deftanh_deriv(x):
return1.0-np.tanh(x)**2
deflogistic(x):
return1/(1+np.exp(-x))
```

```
deflogistic_derivative(x):
returnlogistic(x)*(1-logistic(x))
```

#In the constructor of the class we will need to set the number of neurons in each layer, initialize their weights randomly between -0.25 and 0.25 and set the activation function to be used. Each layer, except the last one, will also have a bias unit which corresponds to the threshold value for the activation.

```
classNeuralNetwork:
def__init__(self,layers,activation='tanh'):
"""
:param layers: A list containing the number of units in each layer.
    Should be at least two values
:param activation: The activation function to be used. Can be
    "logistic" or "tanh"
    """
ifactivation=='logistic':
self.activation=logistic
self.activation_deriv=logistic_derivative
elifactivation=='tanh':
self.activation=tanh
self.activation_deriv=tanh_deriv
self.weights=[]
foriinrange(1,len(layers)-1):
        self.weights.append((2*np.random.random((layers[i-
        1]+1,layers[i]))-1)*0.25)
self.weights.append((2*np.random.random((layers[i]+1,layers[i+
1]))-1)*0.25)
```

Now let's train the network with back-propagation. i.e Given a set of input vectors X and output values y, adjust the weights appropiately.

```
deffit(self,X,y,learning_rate=0.2,epochs=10000):
```

```
X=np.atleast_2d(X)
temp=np.ones([X.shape[0],X.shape[1]+1])
temp[:,0:-1]=X# adding the bias unit to the input layer
X=temp
y=np.array(y)
forkinrange(epochs):
i=np.random.randint(X.shape[0])
a=[X[i]]
forlinrange(len(self.weights)):
hidden_inputs=np.ones([self.weights[l].shape[1]+1])
hidden_inputs[0:-1]=self.activation(np.dot(a[l],self.weights[l]))
a.append(hidden_inputs)
error=y[i]-a[-1][:-1]
deltas=[error*self.activation_deriv(a[-1][:-1])]
l=len(a)-2
# The last layer before the output is handled separately because of
# the lack of bias node in output
deltas.append(deltas[-
1].dot(self.weights[l].T)*self.activation_deriv(a[l]))
forlinrange(len(a)-3,0,-1):# we need to begin at the second to last layer
deltas.append(deltas[-1][:-
1].dot(self.weights[l].T)*self.activation_deriv(a[l]))
deltas.reverse()
foriinrange(len(self.weights)-1):
layer=np.atleast_2d(a[i])
delta=np.atleast_2d(deltas[i])
self.weights[i]+=learning_rate*layer.T.dot(delta[:,:-1])
# Handle last layer separately because it doesn't have a bias unit
i+=1
```

```
layer=np.atleast_2d(a[i])
delta=np.atleast_2d(deltas[i])
self.weights[i]+=learning_rate*layer.T.dot(delta)
#Now test the network
defpredict(self,x):
a=np.array(x)
forlinrange(0,len(self.weights)):
temp=np.ones(a.shape[0]+1)
temp[0:-1]=a
a=self.activation(np.dot(temp,self.weights[l]))
returna
# Now call the above class and functions
nn=NeuralNetwork([2,2,1],'tanh')
X=np.array([[0,0],[0,1],[1,0],[1,1]])
y=np.array([0,1,1,0])
nn.fit(X,y)
foriin[[0,0],[0,1],[1,0],[1,1]]:
print(i,nn.predict(i))
```

Output is:

```
([0, 0], array([ 4.01282568e-05]))
([0, 1], array([ 0.98765949]))
([1, 0], array([ 0.98771753]))
([1, 1], array([ 0.00490502]))
```

Here the output 0.00480502 is near zero so it is 0, and the outpur 0.9876 is near to 1 so it is 1.

✿✿✿

UNIT 6

Application of AI

OUTLINE

- ✓ **Introduction**
  - Machine Learning and Artificial Intelligence
  - General Machine Learning agent Framework or architecture
  - Types of Machine Learning
- ✓ **Statistical-based Learning: Naive Bayes Model**
- ✓ **Neural Networks**
  - Components of a Neuron
  - Activation Functions
  - Types of Feed Neural Networks

Artificial intelligence, defined as intelligence exhibited by machines, has many applications in today's society. More specifically, it is Weak AI, the form of A.I. where programs are developed to perform specific tasks, that is being utilized for a wide range of activities including medical Expert Systems, robot control, Human Computer Interactions, Pattern Recognition (Neural Networks) and remote sensing. AI has been used to develop and advance numerous fields and industries, including finance, healthcare, education, transportation, and more.

Expert Systems

An expert system's knowledge is obtained from expert sources and coded in a form suitable for the system to use in its inference or reasoning processes. The expert knowledge must be obtained from specialists or other sources of expertise, such as texts, journals, articles and data bases.

An expert system is an "intelligent" program that solves problems in a narrow problem area by using high-quality, specific knowledge rather than an algorithm.

Block Diagram

There is currently no such thing as "standard" expert system. Because a variety of techniques are used to create expert systems, they differ as widely as the programmers who develop them and the problems they are designed to solve. However, the principal components of most expert systems are **knowledge base, an inference engine, and a user interface,** as illustrated in the figure.

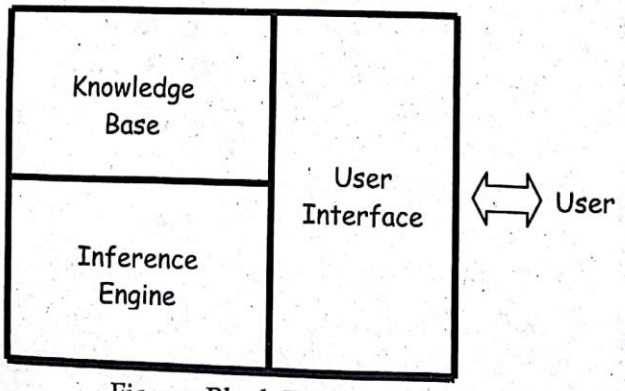


Figure: Block Diagram of expert system

1. **Knowledge Base**

The component of an expert system that contains the system's knowledge is called its knowledge base. This element of the system is so critical to the way most expert systems are constructed that they are also popularly known as *knowledge-based systems*.

To improve the performance of an expert system, we should supply the system with some knowledge about the knowledge it posses, or in other words, meta-knowledge.

2. **Inference Engine**

Simply having access to a great deal of knowledge does not make you an expert; you also must know **how** and **when** to apply the appropriate knowledge. Similarly, just having a knowledge base does not make an expert system intelligent. The system must have another component that directs the implementation of the knowledge. That element of the system is known variously as the *control structure*, the *rule interpreter*, or the *inference engine*.

3. **User Interface**

The component of an expert system that communicates with the user is known as the *user interface*. The communication performed by a user interface is bidirectional. At the simplest level, we must be able to describe our problem to the expert system, and the system must be able to respond with its recommendations. We may want to ask the system to explain its "reasoning", or the system may request additional information about the problem from us.

Stages of Expert System Development

An expert system typically is developed and refined over a period of several years. We can divide the process of expert system development into five distinct stages. In practice, it may not be possible to break down the expert system development cycle precisely. However, an examination of these five stages may serve to provide us with some insight into the ways in which expert systems are developed.

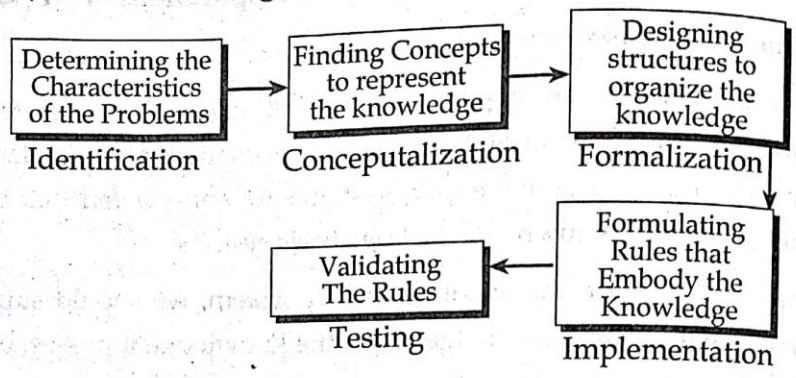


Fig: Different phases of expert system development

Identification

Beside we can begin to develop an expert system, it is important that we describe, with as much precision as possible, the problem that the system is intended to solve. It is not enough simply to feel that the system would be helpful in certain situation; we must determine the exact nature of the problem and state the precise goals that indicate exactly how we expect the expert system to contribute to the solution.

Conceptualization

Once we have formally identified the problem that an expert system is to solve, the next stage involves analyzing the problem further to ensure that its specifics, as well as it generalities, are understood. In the conceptualization stage the **knowledge engineer** frequently creates a diagram of the problem to depict graphically the relationships between the objects and processes in the problem domain. It is often helpful at this stage to divide the problem into a series of sub-problems and to diagram both the relationships among the pieces of each sub-problem and the relationships among the various sub-problems.

Formalization

In the preceding stages, no effort has been made to relate the domain problem to the artificial intelligence technology that may solve it. During the identification and the conceptualization stages, the focus is entirely on understanding the problem. Now, during the formalization stage, the problem is connected to its proposed solution,

an expert system, by analyzing the relationships depicted in the conceptualization stage.

During formalization, it is important that the knowledge engineer be familiar with the following:

- The various techniques of knowledge representation and heuristic search used in expert systems.
- The expert system "tools" that can greatly expedite the development process. And
- Other expert systems that may solve similar problems and thus may be adequate to the problem at hand.

Implementation

During the implementation stage, the formalized concepts are **programmed** onto the computer that has been chosen for system development, using the predetermined techniques and tools to implement a "first pass" prototype of the expert system.

Theoretically, if the methods of the previous stage have been followed with diligence and care, the implementation of the prototype should be as much an art as it is a science, because following all rules does not guarantee that the system will work the first time it is implemented. Many scientists actually consider the first prototype to be a "throw-away' system, useful for evaluating progress but hardly a usable expert system.

Testing

Testing provides opportunities to identify the weakness in the structure and implementation of the system and to make the appropriate corrections. Depending on the types of problems encountered, the testing procedure may indicate that the system was

Features of an expert system:

What are the features of a good expert system? Although each expert system has its own particular characteristics, there are several features common to many systems. The following list from Rule-Based Expert

Systems suggests seven criteria that are important prerequisites for the acceptance of an expert system .

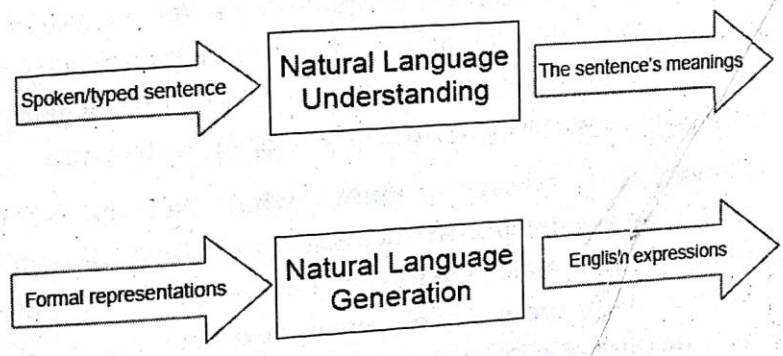
1. "The program should be **useful**." An expert system should be developed to meet a specific need, one for which it is recognized that assistance is needed.
2. "The program should be **usable**." An expert system should be designed so that even a novice computer user finds it easy to use.
3. "The program should be **educational when appropriate**." An expert system may be used by non-experts, who should be able to increase their own expertise by using the system.
4. "The program should be able to **explain its advice**." An expert system should be able to explain the "reasoning" process that led it to its conclusions, to allow us to decide whether to accept the system's recommendations.
5. "The program should be able to **respond to simple questions**." Because people with different levels of knowledge may use the system , an expert system should be able to answer questions about points that may not be clear to all users.
6. "The program should be able to **learn new knowledge**." Not only should an expert system be able to respond to our questions, it also should be able to ask questions to gain additional information.
7. "The program's knowledge should be **easily modified**." It is important that we should be able to revise the knowledge base of an expert system easily to correct errors or add new information.

Natural Language Processing

Natural Language Processing (NLP), is the attempt to extract the fuller meaning representation from the free text. Natural language processing is a technology which involves converting spoken or written human language into a form which can be processed by

computers, and vice versa. Some of the better-known applications of NLP include:

- **Voice recognition software** which translates speech into input for word processors or other applications;
- **Text-to-speech synthesizers** which read text aloud for users such as the hearing-impaired;
- **Grammar and style checkers** which analyze text in an attempt to highlight errors of grammar or usage;
- **Machine translation systems** which automatically render a document such as a web page in another language.



Natural Language Generation

"Natural Language Generation (NLG), also referred to as text generation, is a subfield of natural language processing (NLP; which includes computational linguistics)

Natural Language Generation (NLG) is the natural language processing task of generating natural language from a machine representation system such as a knowledge base or a logical form.

In a sense, one can say that an NLG system is like a translator that converts a computer based representation into a natural language representation. However, the methods to produce the final language are very different from those of a compiler due to the inherent expressivity of natural languages.

NLG may be viewed as the opposite of natural language understanding. The difference can be put this way: whereas in natural

language understanding the system needs to disambiguate the input sentence to produce the machine representation language, in NLG the system needs to make decisions about how to put a concept into words.

The different types of generation techniques can be classified into four main categories:

- Canned text systems constitute the simplest approach for single-sentence and multi-sentence text generation. They are trivial to create, but very inflexible.
- Template systems, the next level of sophistication, rely on the application of pre-defined templates or schemas and are able to support flexible alterations. The template approach is used mainly for multi-sentence generation, particularly in applications whose texts are fairly regular in structure.
- Phrase-based systems employ what can be seen as generalized templates. In such systems, a phrasal pattern is first selected to match the top level of the input, and then each part of the pattern is recursively expanded into a more specific phrasal pattern that matches some subportion of the input. At the sentence level, the phrases resemble phrase structure grammar rules and at the discourse level they play the role of text plans.
- Feature-based systems, which are as yet restricted to single-sentence generation, represent each possible minimal alternative of expression by a single feature. Accordingly, each sentence is specified by a unique set of features. In this framework, generation consists in the incremental collection of features appropriate for each portion of the input. Feature collection itself can either be based on unification or on the traversal of a feature selection network. The expressive power of the approach is very high since any distinction in language can be added to the system as a feature. Sophisticated feature-based generators, however, require very complex input and

make it difficult to maintain feature interrelationships and control feature selection.

Many natural language generation systems follow a hybrid approach by combining components that utilize different techniques.

Natural Language Understanding

Developing programs that understand a natural language is a difficult problem. Natural languages are large. They contain infinity of different sentences. No matter how many sentences a person has heard or seen, new ones can always be produced. Also, there is much ambiguity in a natural language. Many words have several meaning such as can, bear, fly, bank etc, and sentences have different meanings in different contexts.

Example :- A *can* of juice. I *can* do it.

This makes the creation of programs that understand a natural language, one of the most challenging tasks in AI. Understanding the language is not only the transmission of words. It also requires inference about the speakers' goal, knowledge as well as the context of the interaction. We say a program understand natural language if it behaves by taking the correct or acceptable action in response to the input. A word functions in a sentence as a part of speech. Parts of the speech for the English language are nouns, pronouns, verbs, adjectives, adverbs, prepositions, conjunctions and interjections. Three major issues involved in understanding language.

- A large amount of human knowledge is assumed.
- Language is pattern based, phonemes are components of the words and words make phrases and sentences. Phonemes, words and sentences order are not random.
- Language acts are the product of agents (human or machine).

Levels of knowledge used in Language Understanding

A language understanding knowledge must have considerable knowledge about the structures of the language including what the words are and how they combine into phrases and sentences. It must

also know the meanings of the words and how they contribute to the meanings of the sentence and to the context within which they are being used. The component forms of knowledge needed for an understanding of natural languages are sometimes classified according to the following levels.

- **Phonological**
  - ✓ Relates sound to the words we recognize. A phoneme is the smallest unit of the sound. Phones are aggregated to the words.
- **Morphological**
  - ✓ This is lexical knowledge which relates to the word construction from basic units called morphemes. A morpheme is the smallest unit of meaning. Eg:- *friend* + ly = friendly
- **Syntactic**
  - ✓ This knowledge relates to how words are put together or structure red together to form grammatically correct sentences in the language.
- **Semantic**
  - ✓ This knowledge is concerned with the meanings of words and phrases and how they combine to form sentence meaning.
- **Pragmatic**
  - ✓ This is high – level knowledge which relates to the use of sentences in different contexts and how the context affects the meaning of the sentence.
- **World**
  - ✓ Includes the knowledge of the physical world, the world of human social interaction, and the roles of goals and intentions in communication.

Challenges in NLP

Most of the sentence in NL are ambiguous.

Ambiguity occurs at many level in NL

Word level ambiguity: here a word may have many meanings

Sentence level ambiguity: Here one sentence may have many interpretation according to its syntax

Eg: The dealer sold the merchant a dog

Here it is not clear about what is sold to whom, it depends on our interpretation and common sense.

But computer have no common sense and hence it is difficult to understand the meaning of such sentence by a computer program.

- The basic definition of ambiguity, as generally used in natural language processing, is "capable of being understood in more than one way".
- It can be classified into many different types and using various different classification schemes.
- The most widely used classification is probably the one which divides ambiguity into :
  - ✓ Lexical ambiguity, whereby a word may have more than one interpretation;
  - ✓ Semantic ambiguity, whereby several interpretations result from the different ways in which the meanings of words in a phrase can be combined;
  - ✓ Syntactic ambiguity, whereby several different interpretations result from the different ways in which a sequence of words can be grammatically structured; and
  - ✓ Pragmatic ambiguity whereby the context of a phrase results in there being alternative interpretations of that phrase.

Lexeme (Lexicon) & word forms:

The distinction between these two senses of "word" is arguably the most important one in morphology. The first sense of "word", the one

in which *dog* and *dogs* are "the same word", is called a lexeme. The second sense is called *word form*. We thus say that *dog* and *dogs* are different forms of the same lexeme. *Dog* and *dog catcher*, on the other hand, are different lexemes, as they refer to two different kinds of entities. The form of a word that is chosen conventionally to represent the canonical form of a word is called a lemma, or citation form.

A lexicon defines the words of a language that a system knows about. This is includes common words and words that are specific to the domain of the application. Entries include meanings for each word and its syntactic and morphological behavior.

Morphology:

Morphology is the identification, analysis and description of the structure of words (words as units in the lexicon are the subject matter of lexicology.

Morphological analysis is the process of recognizing the suffixes and prefixes that have been attached to a word.

We do this by having a table of affixes and trying to match the input as:

prefixes +root + suffixes.

- For example: adjective + ly -> adverb. E.g.: [Friend + ly]=friendly
- We may not get a unique result.
- "-s, -es" can be either a plural noun or a 3ps verb
- "-d, -ed" can be either a past tense or a perfect participle

Morphological Information:

- Transform part of speech
  - ✓ *green, greenness (adjective, noun)*
  - ✓ *walk, walker (verb, noun)*
- Change features of nouns
  - ✓ *boat, boats (singular, plural)*
- Bill slept , Bill's bed
  - ✓ (subjective case, possessive case)

- Change features of verbs
  - ✓ Aspect
    - *I walk. I am walking.* (present, progressive)
  - ✓ Tense
    - *I walked. I will walk. I had been walking: (past, future, past progressive)*
  - ✓ Number and person
    - *I walk. They walk. (first person singular, third person plural)*

Syntactic Analysis:

Syntactic analysis takes an input sentence and produces a representation of its grammatical structure. A grammar describes the valid parts of speech of a language and how to combine them into phrases. The grammar of English is nearly context free.

A computer grammar specifies which sentences are in a language and their parse trees. A parse tree is a hierarchical structure that shows how the grammar applies to the input. Each level of the tree corresponds to the application of one grammar rule.

It is the starting point for working out the meaning of the whole sentence. Consider the following two sentences.

1. "The dog ate the bone."
2. "The bone was eaten by the dog."

Understanding the structure (via the syntax rules) of the sentences help us work out that it's the bone that gets eaten and not the dog. Syntactic analysis determines possible grouping of words in a sentence. In other cases there may be many possible groupings of words. Consider the sentence "John saw Mary with a telescope". Two different readings based on the groupings.

1. John saw (Mary with a telescope).
2. John (saw Mary with a telescope).

A sentence is syntactically ambiguous if there are two or more possible groupings. Syntactic analysis helps determining the meaning of a sentence by working out possible word structure. Rules of syntax are specified by writing a *grammar* for the language. A parser will check if a sentence is correct according to the grammar. It returns a representation (parse tree) of the sentence's structure. A grammar specifies allowable sentence structures in terms of basic categories such as noun and verbs. A given grammar, however, is unlikely to cover all possible grammatical sentences. Parsing sentences is to help determining their meanings, not just to check that they are correct. Suppose we want a grammar that recognizes sentences like the following.

John ate the biscuit.

The lion ate the zebra.

The lion kissed John

But reject incorrect sentences such as

Ate John biscuit the.

Zebra the lion the ate.

Biscuit lion kissed.

A simple grammar that deals with this is given below

sentence --> noun_phase, verb phrase.

noun_phrase --> proper_noun.

noun_phrase --> determiner, noun.

verb_phrase --> verb, noun_phrase.

proper_noun --> [mary].

proper_noun --> [john].

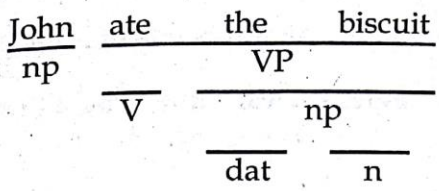
noun --> [zebra].

noun --> [biscuit].

verb --> [ate].

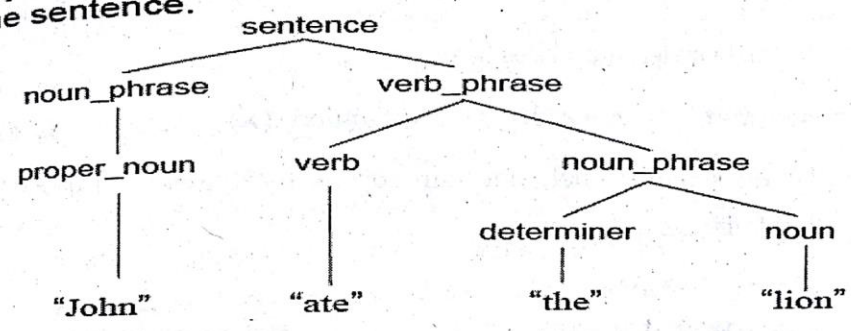
verb --> [kissed].

determiner --> [the].



Incorrect sentences like "biscuit lion kissed" will be excluded by the grammar.

- A ***parse trees*** **illustrates the syntactic structure of the sentence.**



Semantic Analysis:

Semantic analysis is a process of converting the syntactic representations into a meaning representation.

This involves the following tasks:

- Word sense determination
- Sentence level analysis
- Knowledge representation

- **Word sense**

  Words have different meanings in different contexts.

  Mary had a bat in her office.

  ✓ bat = `a baseball thing'

  ✓ bat = `a flying mammal'

- **Sentence level analysis**

  Once the words are understood, the sentence must be assigned some meaning

  I saw an astronomer with a telescope.

- *Knowledge Representation*

  Understanding language requires lots of knowledge.

- **Using predicate logic, for example, one can represent sentence like**

  "John likes Mary" `likes (John, Mary)`

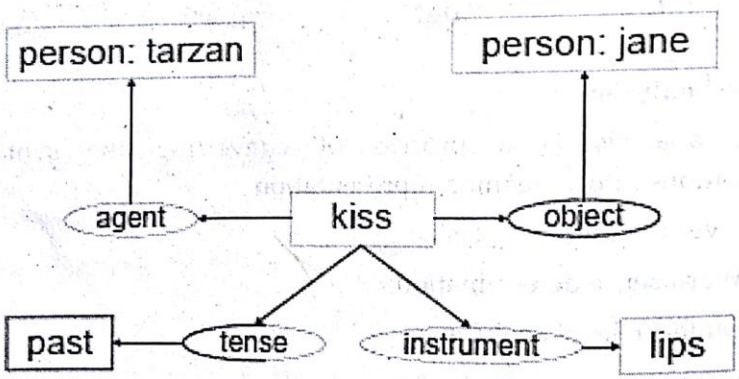
  "The man likes Mary" `Man (ml) ∧ likes (ml, mary)`

  "A man likes Mary" `∃X (man (X) ∧ likes (X, Mary)`

  "A tall bearded man likes Mary"

  `∃X (man (X) ∧ tall (X) ∧ bearded (X) ∧ likes (X, mary)`

- Using semantic net, one can represent sentence "Tarzan kissed Jane" as



Parameters in Natural Language Processing:

- Auditory Inputs
- Segmentation
- Syntax Structure
- Semantic Structure
- Pragmatic Analysis

Auditory Inputs:

This basically consists of input devices such as text input, audio input and video inputs. some cases, an audio output device can be used as an input device, in order to capture produced sound.

- Microphone
- MIDI keyboard or other digital musical instrument

Segmentation:

Text segmentation is the process of dividing written text into meaningful units, such as words, sentences, or topics.

Word segmentation is the problem of dividing a string of written language into its component words. In English and many other languages using some form of the Latin alphabet, the space is a good approximation of a word delimiter. (Some examples where the space character alone may not be sufficient include contractions like *can't* for *can not*.)

However the equivalent to this character is not found in all written scripts, and without it word segmentation is a difficult problem. Languages which do not have a trivial word segmentation process include Chinese, Japanese, where sentences but not words are delimited, and Thai, where phrases and sentences but not words are delimited.

In some writing systems however, such as the Ge'ez script used for Amharic and Tigrinya among other languages, words are explicitly delimited (at least historically) with a non-whitespace character.

Word splitting is the process of parsing concatenated text (i.e. text that contains no spaces or other word separators) to infer where word breaks exist.

Other Segmentation Problems: Processes may be required to segment text into segments besides words, including morphemes (a task usually called morphological analysis), paragraphs, topics or discourse turns.

A document may contain multiple topics, and the task of computerized text segmentation may be to discover these topics

automatically and segment the text accordingly. The topic boundaries may be apparent from section titles and paragraphs. In other cases one needs to use techniques similar to those used in document classification. Many different approaches have been tried.

- **Syntax Structure:**

  Same concept as in the syntactic analysis above

- **Semantic Structure:**

  Same concept as in the semantic analysis above

- **Pragmatic Analysis:**

This is high level knowledge which relates to the use of sentences in different contexts and how the context affects the meaning of the sentences. It is the study of the ways in which language is used and its effect on the listener. Pragmatic comprises aspects of meaning that depend upon the context or upon facts about real world.

- **Pragmatics – Handling Pronouns**

Handling pronouns such as "he", "she" and "it" is not always straight forward. Let us see the following paragraph.

"John buys a new telescope. He sees Mary in the distance. He gets out his telescope. He looks at her through it".

Here, *"her"* refers to *Mary* who was not mentioned at all in the previous sentences. John's telescope was referred to as *"a new telescope"*, *"his telescope"* and *"it"*.

Let us see one more example

"When is the next flight to Sydney?"

"Does it have any seat left?"

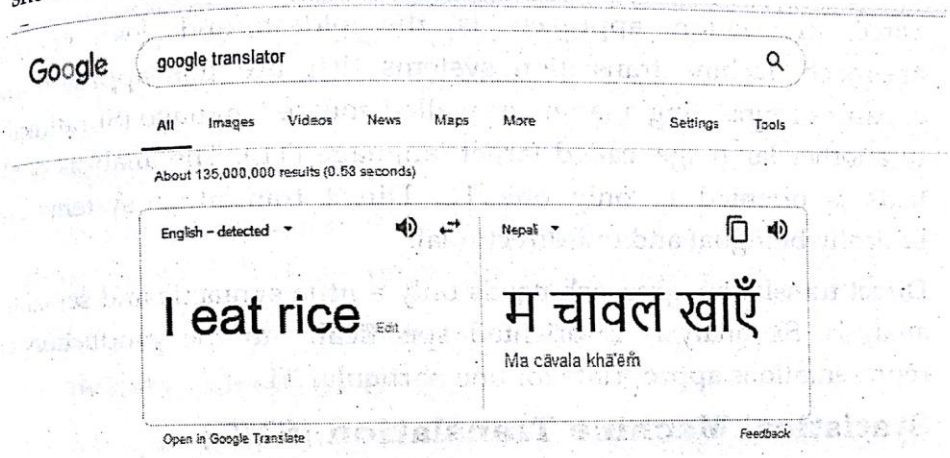
Here, *"it"*, refers to a particular flight to Sydney, not Sydney itself.

Machine Translation

The term *machine translation* (MT) is used in the sense of translation of one language to another. The ideal aim of machine translation systems is to produce the best possible translation without human assistance.Basically every machine translation system requires programs for translation and automated dictionaries and grammars to support translation.

Machine translation, sometimes referred to by the abbreviation MT (not to be confused with computer-aided translation, machine-aided human translation (MAHT) or interactive translation) is a sub-field of computational linguistics that investigates the use of software to translate text or speech from one language to another.

An example of machine translation is "Google Translator" which is shown below:



Pre-editing and Post Editing in MT

The translation quality of the machine translation systems can be improved by pre-editing the input.

Pre-editing means adjusting the input by marking prefixes, suffixes, clause boundaries, etc.

Translation quality can also be improved by controlling the vocabulary. The output of the machine translation should be post-edited to make it perfect.

Post-editing is required especially for health related information.

Types of Machine Translations

There are four types of machine translation–

- Statistical Machine Translation (SMT),
- Rule-based Machine Translation (RBMT),
- Hybrid Machine Translation(HMT),
- Neural Machine Translation (NMT).

Rule Based Machine Translation (RBMT)

RBMT, on the other hand, translates on the basis of grammatical rules. It conducts a grammatical analysis of the source language and the target language to generate the translated sentence. However, RBMT requires extensive proofreading, and its heavy dependence on lexicons means that efficiency is achieved after a long period of time. It is also known as Direct Translation.

Direct translation approach is the oldest and less popular approach.Machine translation systems that use this approach are capable oftranslating a language, called source language (SL) directly to another language, called target language (TL). The analysis of SL texts is oriented to only one TL. Direct translation systems are basically bilingual and uni-directional.

Direct translation approach needs only a little syntactic and semantic analysis. SL analysis is oriented specifically to the production of representations appropriate for one particular TL.

Statistical Machine Translation (SMT)

Statistical machine translation automatically learns lexical and structural preferences from corpora. Statistical models offer good solution to ambiguity problem. They are robust and work well even if there are errors and the presence of new data. IBM researchers pioneered the first statistical approach to machine translation in 1980's. IBM group relies on the source-channel approach, a framework for combining a word-based translation model and a language model. The translation model ensures that the machine translation system produces target hypothesis corresponding to the source sentence. The language model ensures the grammatically correct output.

SMT works by referring to statistical models that are based on the analysis of large volumes of bilingual text. It aims to determine the correspondence between a word from the source language and a word from the target language.

A good example of this is Google Translate. Now, SMT is great for basic translation, but its greatest drawback is that it does not factor in context, which means translations can often be erroneous. In other words, don't expect high-quality translations.

Hybrid Machine Translation (HMT)

HMT, as the term indicates, is a blend of RBMT and SMT. It leverages a translation memory, making it far more effective in terms of quality. However, even HMT has its share of drawbacks, the greatest of which is the need for extensive editing. Human translators will be required.

Neural Machine Translation (NMT)

NMT is a type of machine translation that depends on neural network models (based on the human brain) to develop statistical models for the purpose of translation. The primary benefit of NMT is that it provides a single system that can be trained to decipher the source and target text.

As a result, it does not depend on specialized systems that are common to other machine translation systems, especially SMT.

To put it simply, NMT is an end-to-end translation system.

Discussion of above Methods

As you can see, it can be quite difficult to determine the ideal machine translation approach for your business. Each offers its own set of pros and cons. It ultimately boils down to your own needs.

But, SMT is the most preferred approach today. This is because of the rules of language change which impacts the RBMT approach. These rules need to be constantly updated. SMT, however, does not depend on rules and its systems can be constructed in much less time compared to RBMT systems.

The training data needed to run SMT is also widely available on the Internet due to the publication of multilingual content. Finally, its need for a high amount of processing power is also easy to meet now, thanks to cloud solutions.

NMT, on the other hand, is definitely the most advanced option here. However, training models for NMT is an expensive affair, which means small-to-medium-sized businesses will have to consider the cost to profit ratio.

The other types of machine translation system are as follows:

Empirical Machine translation (EMT)

Empirical approach is the emerging one that uses large amount of raw data in the form of parallel corpora. The raw data consists of texts and their translations.

Example-based MT analogy-based MT, memory-based MT, and case-based MT are the techniques that use empirical approach. Basically all these techniques use a corpus database of translated examples. Statistical machine translation is Corpus based but slightly different in the sense that it depends on statistical modelling of the word order of the target language and of source-target word equivalences.

Interlingua approach

Interlingua approach intends to translate SL texts to that of more than one language.

Translation is from SL to an intermediate form called Interlingua (IL) and then from IL to TL.

Interlingua may be artificial one or auxiliary language like /Esperanto/ with universal vocabulary.

Interlingua approach requires complete resolution of all ambiguities in the SL text.

An Introduction to Machine Vision Systems

Machine vision is the incorporation of computer vision into industrial manufacturing processes, although it does differ substantially from computer vision. In general, computer vision revolves around image processing. Machine vision, on the other hand, uses digital input and output to manipulate mechanical components. Devices that depend on machine vision are often found at work in product inspection, where they often use digital cameras or other forms of automated vision to perform tasks traditionally performed by a human operator.

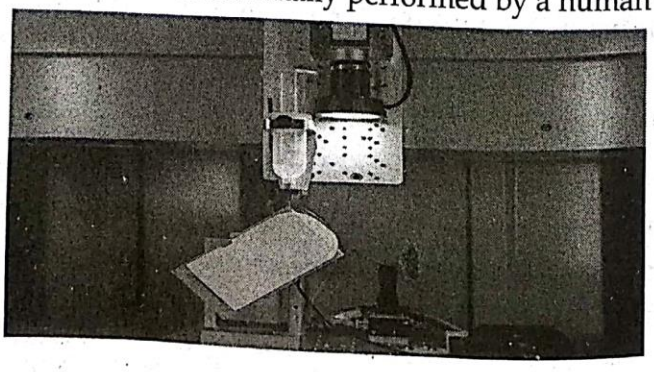


Fig: Example of machine Vision System(Medical Imagining System)

The **components of a machine vision system** can vary, but there are several common factors found in most. These elements include:

Image Sensor

This is also known as a **vision sensor**, which is responsible for analysing images captured by the camera to determine the accuracy of assembly or the presence of defects. Two types of vision sensors are most commonly found on the market, orthographic projection and perspective projection. In an orthographic projection type, the field of vision is rectangular, making it suitable for infrared sensors. In contrast, the perspective type projects a trapezoidal field of view, making it more suitable for camera-type sensors.

Vision Processor

From the name itself, it is clear that the processor is in charge of running machine vision algorithms. It may include a direct interface that has the capacity to take relevant data from a camera device. Basically, it is essential to ensure that the image processing is completed without any mistake or error.

Communication Interface

Another essential component is called the communication interface, which connects the other parts in the vision system. It is necessary to facilitate data transmission. This interface is usually attached to smart camera or other vision devices. Through this, the vision system can link data from the device to particular software that is run in a separate personal computer.

Lighting

Every vision system must have a working lighting analysis sequence. These are the common sources of lighting that is available in most machine vision systems:

- LED
- Mercury (also known as Metal Halide)
- Xenon
- Quartz Halogen
- Fluorescent

The type of lighting devices used depends on the purpose for the creation of the vision system. At the same time, it is also dependent on how big the vision stations are.

Lens

The lens must be matched with the image sensor installed in the vision system. This transparent material has curved sides that are employed to disperse light rays. It can be used singly or can be combined with other lenses. What makes this special is that it is considered as the light-gathering component of the vision system.

All these parts are equally essential in order to guarantee that the vision system will provide utmost satisfaction to all its users. The absence of one component will render the entire system worthless. Having said this, your goal is to ensure that all the important components are working upon the installation of the vision system for industrial manufacturing purposes.

- Input and output hardware
- Lenses
- Light sources, such as LED illuminators or halogen lamps
- An image processing program
- A sensor to detect and trigger image acquisition
- Actuators to sort defective parts

How Machine Vision Systems Work

Although each of these components serves its own individual function and can be found in many other systems, when working together they each have a distinct role in a machine vision system. To understand how a machine vision system works, it may be helpful to envision it performing a typical function, such as product inspection.

First, the sensor detects if a product is present. If there is indeed a product passing by the sensor, the sensor will trigger a camera to capture the image, and a light source to highlight key features.

Next, a digitizing device called a frame grabber takes the camera's image and translates it into digital output, which is then stored in computer memory so it can be manipulated and processed by software.

In order to process an image, computer software must perform several tasks. First, the image is reduced in gradation to a simple black and white format. Next, the image is analyzed by system software to identify defects and proper components based on predetermined criteria. After the image has been analyzed, the product will either pass or fail inspection based on the machine vision system's findings.

Applications of Machine Vision

Use of Machine Vision in Cameras

Facial Recognition: Recognizing faces is the really easy part in which the software has to just look for facial features like eyes, nose, and the mouth and tell whether there's a face in the photo or note. The difficult part is telling whose face is it from the available database.

Use of Machine Vision in Social Media Websites

Facebook was the very first social media website to use facial recognition technology and till day continues to be the most outspoken one. Last year, the social giant made a facial recognition breakthrough use of machine vision technology by integrating new algorithms that allowed it to identify a person even if their face was covered up.

Use of Machine Vision in Manufacturing Industry

The manufacturing industry has the most reasonable and the most repeated use of machine vision technology. Labeling of bottles, cans and any item that has a painted or ingrained label uses MV solutions to help the software decide where to sketch the labels.

Computer vision is also used for product inspection and consistency as well. Items can be broken or incomplete. Laser vision or cameras are used to scan manufactured items and that the packaging has been properly conducted.

Use of Machine Vision in Health Industry

Ultrasound Devices and Cat-scanners should give you the necessary hint. Although in this case, the light-based camera is not used but remember that only humans are limited to light and some animals can see infrared and UV rays as well. The point being, any photograph is just an image for machines doesn't matter it was captured using the visible light or other frequencies of electromagnetic waves.

Use of Machine Vision Technology in Robotics

The use of machine vision technology is most extensive in robotics than any other prospect. The mars exploring robots depends on the clear visual perception and analysis of the terrain and likewise for any robot to navigate the environment without getting bumped into hurdles, accuracy is necessary.

Machine Vision

Machine vision (MV) is the application of computer vision to industry and manufacturing. Whereas computer vision is the general discipline of making computers see (understand what is perceived visually), machine vision, being an engineering discipline, is interested in digital input/output devices and computer networks to control other manufacturing equipment such as robotic arms and equipment to eject defective products.

Machine vision is the ability of a computer to "see." A machine-vision system employs one or more video cameras, analog-to-digital conversion (ADC), and digital signal processing (DSP). The resulting data goes to a computer or robot controller. Machine vision is similar in complexity to voice recognition. The machine vision systems use video cameras, robots or other devices, and computers to visually analyze an operation or activity. Typical uses include automated inspection, optical character recognition and other non-contact applications.

Two important specifications in any vision system are the sensitivity and the resolution. Sensitivity is the ability of a machine to see in dim light, or to detect weak impulses at invisible wavelengths. Resolution is the extent to which a machine can differentiate between objects. In general, the better the resolution, the more confined the field of vision. Sensitivity and resolution are interdependent. All other factors held constant, increasing the sensitivity reduces the resolution, and improving the resolution reduces the sensitivity.

One of the most common applications of Machine Vision is the inspection of manufactured goods such as semiconductor chips, automobiles, food and pharmaceuticals. Just as human inspectors working on assembly lines visually inspect parts to judge the quality of workmanship, so machine vision systems use digital cameras, smart cameras and image processing software to perform similar inspections.

Machine vision systems have two primary hardware elements: the camera, which serves as the eyes of the system, and a computer video analyser. The recent rapid acceleration in the development of machine vision for industrial applications can be attributed to research in the areas of computer technologies. The first step in vision analysis is the conversion of analog pixel intensity data into digital format for processing. Next, an appropriate computer algorithm is employed to understand the image data and provide appropriate analysis or action.

Machine vision encompasses computer science, optics, mechanical engineering, and industrial automation. Unlike computer vision which is mainly focused on machine-based image processing, machine vision integrates image capture systems with digital input/output devices and computer networks to control manufacturing equipment such as robotic arms. Manufacturers favour machine vision systems for visual inspections that require high-speed, high-magnification, 24-hour operation, and/or repeatability of measurements.

A typical machine vision system will consist of most of the following components:

- One or more digital or analogue cameras (black-and-white or colour) with suitable optics for acquiring images, such as lenses to focus the desired field of view onto the image sensor and suitable, often very specialized, light sources
- Input/Output hardware (e.g. digital I/O) or communication links (e.g. network connection or RS-232) to report results
- A synchronizing sensor for part detection (often an optical or magnetic sensor) to trigger image acquisition and processing and some form of actuators to sort, route or reject defective parts
- A program to process images and detect relevant features.

The aim of a machine vision inspection system is typically to check the compliance of a test piece with certain requirements, such as prescribed dimensions, serial numbers, presence of components, etc. The complete task can frequently be subdivided into independent stages, each checking a specific criterion. These individual checks typically run according to the following model:

1. Image Capture
2. Image Preprocessing

3. Definition of one or more (manual) regions of interest
4. Segmentation of the objects
5. Computation of object features
6. Decision as to the correctness of the segmented objects

Naturally, capturing an image, possible several for moving processes, is a pre-requisite for analysing a scene. In many cases these images are not suited for immediate examination and require pre-processing to change certain sizing specific structures etc. In most cases it is at least approximately known which image areas have to be analysed, i.e. the location of a mark to be read or a component to be verified. These are called Regions of Interest (ROIs) (sometimes Area of Interest or AOIs). Of course, such a region can also comprise the entire image if required.

Machine vision is used in various industrial and medical applications. Examples include:

- Electronic component analysis
- Signature identification
- Optical character recognition
- Handwriting recognition
- Object recognition
- Pattern recognition
- Materials inspection
- Currency inspection
- Medical image analysis

Computer Vision

Computer vision is the science and technology of machines that see, where *see* in this case means that the machine is able to extract information from an image that is necessary to solve some task. As a scientific discipline, computer vision is concerned with the theory behind artificial systems that extract information from images. The image data can take many forms, such as video sequences, views from multiple cameras, or multi-dimensional data from a medical scanner.

As a technological discipline, computer vision seeks to apply its theories and models to the construction of computer vision systems. Examples of applications of computer vision include systems for:

- Controlling processes (e.g., an industrial robot or an autonomous vehicle).
- Detecting events (e.g., for visual surveillance or people counting).
- Organizing information (e.g., for indexing databases of images and image sequences).
- Modeling objects or environments (e.g., industrial inspection, medical image analysis or topographical modeling).
- Interaction (e.g., as the input to a device for computer-human interaction).

Computer vision is closely related to the study of biological vision. The field of biological vision studies and models the physiological processes behind visual perception in humans and other animals. Computer vision, on the other hand, studies and describes the processes implemented in software and hardware behind artificial vision systems. Interdisciplinary exchange between biological and computer vision has proven fruitful for both fields.

Computer vision is, in some ways, the inverse of computer graphics. While computer graphics produces image data from 3D models, computer vision often produces 3D models from image data. There is also a trend towards a combination of the two disciplines, e.g., as explored in augmented reality.

Sub-domains of computer vision include scene reconstruction, event detection, video tracking, object recognition, learning, indexing, motion estimation, and image restoration.

Robotics

Robotics is an interdisciplinary branch of engineering and science that includes mechanical engineering, electronic engineering, information engineering, computer science, and others. Robotics deals with the design, construction, operation, and use of robots, as well as computer systems for their control, sensory feedback, and information processing.

Components

A robot generally consists of 5 basic components:

Controller

Every robot is connected to a computer controller, which regulates the components of the arm and keeps them working together. The controller also allows the robot to be networked to other systems; so that it may work together with other machines, processes, or robots.

Almost all robots are pre-programmed using "teaching" devices or offline software programs. In the future, controllers with artificial intelligence (AI) could allow robots to think on their own, even program themselves. This could make robots more self-reliant and independent.

Arm

The arm is the part of the robot that positions the end-effector and sensors to do their pre-programmed business. Many are built to resemble human arms, and have shoulders, elbows, wrists, even fingers. Each joint is said to give the robot 1 degree of freedom. A simple robot arm with 3 degrees of freedom could move in 3 ways: up and down, left and right, forward and backward. Most working robots today have 6 degrees of freedom to allow them to reach any possible point in space within its work envelope. The human arm has 7.

Drive

The links (the sections between the joints) are moved into their desired position by the drive. Typically, a drive is powered by pneumatic or hydraulic pressure, or electricity.

End-Effector

The end-effector could be thought of as the "hand" on the end of the robotic arm. There are many possible end-effectors including a gripper, a vacuum pump, tweezers, scalpel, blowtorch, welder, spray gun, or just about anything that helps it do its job. Some robots can change end-effectors, and be reprogrammed for a different set of tasks.

Sensor

The sensor sends information, in the form of electronic signals back to the controller. Sensors also give the robot controller information about its surroundings and lets it know the exact position of the arm, or the state of the world around it. One of the more exciting areas of sensor development is occurring in the field of computer vision and object recognition. Robot sensors can detect infrared radiation to "see" in the dark.

Exercis

1. Describe role of domain expert, Knowledge engineer and programmer is SDLC of expert system. Explain the features of expert system.
2. how can you construct expert system? Explain knowledge engineering with a bloc diagram.
3. What is neural network? Explain the neural net learning methods.
4. Differentiate between natural language understanding (NLU) and natural language generating (NLG)
5. What do you mean by machine vision? Discuss the components of a machine vision system.
6. Explain the steps of Natural Language Processing with block diagram and examples.
7. Define a natural language processing. Explain the different issues involved in the natural language processing.
8. How can you construct expert system? Explain knowledge engineering with a block diagram.
9. Define natural language processing. Explain the different issues involved in the natural language processing.
10. What is a neural Network? Explain any one type of neural network with practical example.
11. Knowledge consists of facts, beliefs, and heuristics, justify it. Explain the advantages and disadvantages of an expert system.
12. Differentiate between natural language understanding (NLU) and natural language generating (NLG). Why we have to study natural language processing? Explain it.
13. What is an expert system? Explain the architecture and feature of rule-based expert system.
14. Why do we required expert system structure? Draw the block diagram and explain it with practical example.
15. Draw the mathematical model of a neural network. Explain any one type of neural network with its algorithm.

Lab Manual

Natural Language processing with NLTK.

Introduction:

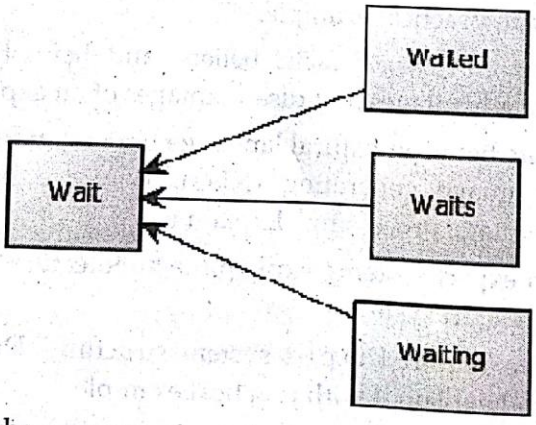
NLTK: A Natural Language Toolkit.(www.nltk.org)

- All level of knowledge in Natural Language processing can be implemented using NLTK, a complete package for NLP. (Open Source)
- Easy to install:
  - import nltk
  - nltk.download()
- We can easily perform following NLP operation in NLTK
  - Steaming
  - PoS tagging
  - NE tagging
  - Parsing

NLTK – stemming

A word stem is part of a word. It is sort of a normalization idea, but linguistic.

For example, the stem of the word waiting is wait.



Stemming is the process of producing morphological variants of a root/base word. Stemming programs are commonly referred to as stemming algorithms or stemmers. A stemming algorithm reduces the words "chocolates", "chocolatey", "choco" to the root word,

"chocolate" and "retrieval", "retrieved", "retrieves" reduce to the stem "retrieve".

Program 1: WAP to find the stem of given words: Waited, Wait, Waiting

Python code:

```
fromnltk.stem importPorterStemmer
fromnltk.tokenize importword_tokenize
ps =PorterStemmer()
# choose some words to be stemmed
words =["wait", "Waited", "Waiting", "waits"]
forw inwords:
  print(w, " : ", ps.stem(w))
```

Program 2: WAP to find the stem of words given in a sentence "gaming, the gamers play games"

Python Code

```
fromnltk.stemimportPorterStemmer
fromnltk.tokenizeimportsent_tokenize,word_tokenize
ps=PorterStemmer()
sentence="gaming, the gamers play games"
words=word_tokenize(sentence)
for word in words:
print(word + ":" + ps.stem(word))
```

Exercise: Try to input the different sentence of your choice and see the output.

NLTK: POS tagging

The module NLTK can automatically tag speech.

Given a sentence or paragraph, it can label words such as verbs, nouns and so on. Some of the POS tag used in English are:

CC coordinating conjunction

CD cardinal digit

DT determiner

EX existential there (like:"there is" ... think of it like "there exists")

FW foreign word

IN preposition/subordinating conjunction
JJ adjective 'big'
JJR adjective, comparative 'bigger'
JJS adjective, superlative 'biggest'
LS list marker 1)
MD modal could, will
NN noun, singular 'desk'
NNS noun plural 'desks'
NNP proper noun, singular 'Harrison'
NNPS proper noun, plural 'Americans'
PDT pre-determiner 'all the kids'
POS possessive ending parent's
PRP personal pronoun I, he, she
VB verb, base form take

And so on.

WAP to find the tag of corresponding word in a given sentence "I love You so much"

Python code

```
importnltk
fromnltk.tokenizeimportPunktSentenceTokenizer
document ="I love you so much"
sentences=nltk.sent_tokenize(document)
for sent in sentences:
print(nltk.pos_tag(nltk.word_tokenize(sent)))
```

Output

[('I', 'PRP'), ('love', 'VBP'), ('you', 'PRP'), ('so', 'RB'), ('much', 'JJ')]

Exercise: Try with different sentence to get output.

NLTK: Parsing (Grammar checking)

- How Compiler know that there is syntactic error?
  - It parse the source code using grammar (rule) and see whether it fits or not.
- Similar thing happen in case of natural language too.

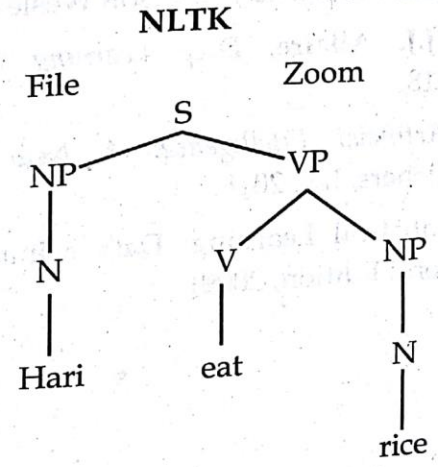
(But here we use natural language grammar instead of grammar of PL)

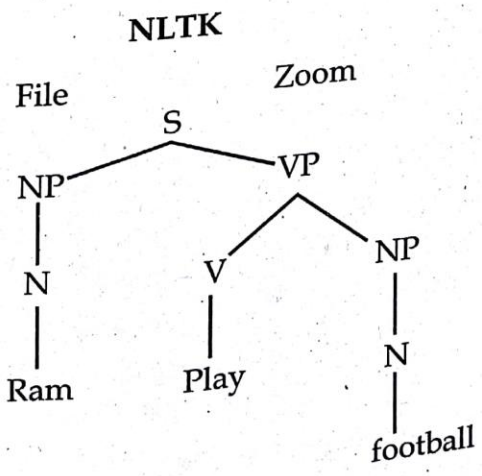
- Given Grammar is:

- Sent→NpVp
- Np→Noun | Dt Noun
- Noun→I | We | rice | football
- Vp→Verb Np
- Verb→eat/go/play
- Dt→a |an |the |my

- Input sentence is:
  - Hari eat rice
  - Ram play football

Output





Bibliography

Stuart Russel and Peter Norvig (Fourth Edition), *Artificial Intelligence :A Modern Approach*, Pearson

George F. Luger (Sixth Edition), *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*, Benjamin/Cummings Publication

E. Rich, K. Knight, Shivashankar B. Nair (Third Edition) ,*Artificial Intelligence*, Tata McGraw Hill.

D. W. Patterson, *Artificial Intelligence and Expert Systems*, Prentice Hall.

P. H. Winston, *Artificial Intelligence*, Addison Wesley

Francois Chollet, J.J. Allaire, *Deep Learning with R*, Manning Publications; 2018.

Nils J. Nilsson, *Artificial Intelligence: A New Synthesis*,Morgan Kaufmann Publishers, Inc, 2018.

The Elements of Statistical Learning: Data Mining, Inference, and Prediction. (Second Edition, 2009).

KEC's B.Sc. CSIT Text Book Series

First Semester
- Introduction to Informatioin Technology
- C Programming
- Digital Logic
- Mathematics I
- Physics

Second Semester
- Discrete Structure
- Object Oriented Progamming
- Microprocessor
- Mathematics II
- Statistics I

Third Semester
- Data Structure and Algorithms
- Numerical Method
- Computer Architecture
- Computer Graphics
- Statistics II

Fourth Semester
- Theory of Computation
- Computer Networks
- Operating System
- Database Management System
- Artificial Intelligence

Fifth Semester
- Design and Analysis of Algorithms
- System Analysis and Design
- Cryptography
- Simulation and Modeling
- Web Technology
- Elective I

Sixth Semester
- Software Engineering
- Complier Design and Construction
- E-Governance
- NET Centric Computing
- Technical Writing
- Elective II

Seventh Semester
- Advanced Jave Programming
- Data Warehousing and Data Mining
- Principles of Management
- Project Work
- Elective III

Eighth Semester
- Advanced Database
- Internship
- Elective IV
- Elective V

KEC
Publication and Distribution (P.) Ltd.
Kathmandu, Nepal, Tel.: 01-4168301, 01-4241777
E-mail: kecpublication14@gmail.com