DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COURSE MATERIALS

CS 464 ARTIFICIAL INTELLIGENCE

VISION OF THE INSTITUTION

To mould true citizens who are millennium leaders and catalysts of change through excellence in education.

MISSION OF THE INSTITUTION

NCERC is committed to transform itself into a center of excellence in Learning and Research in Engineering and Frontier Technology and to impart quality education to mould technically competent citizens with moral integrity, social commitment and ethical values.

We intend to facilitate our students to assimilate the latest technological know-how and to imbibe discipline, culture and spiritually, and to mould them in to technological giants, dedicated research scientists and intellectual leaders of the country who can spread the beams of light and happiness among the poor and the underprivileged.

ABOUT DEPARTMENT

- Established in: 2002
- Course offered : B.Tech in Computer Science and Engineering

  M.Tech in Computer Science and Engineering

  M.Tech in Cyber Security
- Approved by AICTE New Delhi and Accredited by NAAC
- Affiliated to the University of A P J Abdul Kalam Technological University.

DEPARTMENT VISION

Producing Highly Competent, Innovative and Ethical Computer Science and Engineering Professionals to facilitate continuous technological advancement.

DEPARTMENT MISSION

1. To Impart Quality Education by creative Teaching Learning Process
2. To Promote cutting-edge Research and Development Process to solve real world problems with emerging technologies.
3. To Inculcate Entrepreneurship Skills among Students.
4. To cultivate Moral and Ethical Values in their Profession.

PROGRAMME EDUCATIONAL OBJECTIVES

PEO1: Graduates will be able to Work and Contribute in the domains of Computer Science and Engineering through lifelong learning.

PEO2: Graduates will be able to Analyse, design and development of novel Software Packages, Web Services, System Tools and Components as per needs and specifications.

PEO3: Graduates will be able to demonstrate their ability to adapt to a rapidly changing environment by learning and applying new technologies.

PEO4: Graduates will be able to adopt ethical attitudes, exhibit effective communication skills, Teamworkand leadership qualities.

PROGRAM OUTCOMES (POS)

Engineering Graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES (PSO)

PSO1: Ability to Formulate and Simulate Innovative Ideas to provide software solutions for Real-time Problems and to investigate for its future scope.

PSO2: Ability to learn and apply various methodologies for facilitating development of high quality System Software Tools and Efficient Web Design Models with a focus on performance

optimization.

PSO3: Ability to inculcate the Knowledge for developing Codes and integrating hardware/software products in the domains of Big Data Analytics, Web Applications and Mobile Apps to create innovative career path and for the socially relevant issues.

COURSE OUTCOMES

CO1	To understand the scope and limits of the artificial intelligence (AI) field
CO2	To analyze the applicability, strengths, and weaknesses of the basic knowledge representation
CO3	To interpret the role of knowledge representation, problem solving, and learning
CO4	To understand various search algorithms (uninformed, informed, and heuristic) for problem solving
CO5	To acquire knowledge in various learning concepts.
CO6	To comprehend the fundamentals of Natural Language Processing

MAPPING OF COURSE OUTCOMES WITH PROGRAM OUTCOMES

	PO 1	**PO 2**	**PO 3**	**PO 4**	**PO 5**	**PO 6**	**PO 7**	**PO 8**	**PO 9**	**PO 10**	**PO 11**	**PO 12**
CO1	3	3	-	-	-							
CO2	3	2	3	2	3							
CO3	3	3	3	3	2							
CO4	3	3	2	3	2							
CO5	3	3	2	3	3							
CO6	3	3	-	-	-							

Note: H-Highly correlated=3, M-Medium correlated=2, L-Less correlated=1

MAPPING OF COURSE OUTCOMES WITH PROGRAM SPECIFIC OUTCOMES

	PSO1	PSO2	PSO3
CO1	3	-	-
CO2	3	2	-
CO3	3	2	-
CO4	3	-	-
CO5	3	2	-
CO6	3	3	-

SYLLABUS

Course code	Course Name	L-T-P - Credits	Year of Introduction
CS464	ARTIFICIAL INTELLIGENCE	3-0-0-3	2016

Course Objectives:

- To introduce basic principles that drive complex real world intelligence applications.
- To introduce and discuss the basic concepts of AI Techniques and Learning

Syllabus:

Introduction to AI, Solving Problems by Searching-uninformed, informed, heuristic, constraint Satisfaction problems -AI Representational Schemes-Learning-Advanced searches-Alpha beta pruning, Expert Systems-Natural Language Processing Concepts.

Expected Outcome:

The Student will be able to :

i. appreciate the scope and limits of the artificial intelligence (AI) field
ii. assess the applicability, strengths, and weaknesses of the basic knowledge representation
iii. interpret the role of knowledge representation, problem solving, and learning
iv. explain various search algorithms (uninformed, informed, and heuristic) for problem solving
v. comprehend the fundamentals of Natural Language Processing

Text Books:

1. E Rich, K Knight, Artificial Intelligence, 3/e, Tata McGraw Hil, 2009.
2. George.F.Luger, Artificial Intelligence- Structures and Strategies for Complex Problem Solving, 4/e, Pearson Education. 2002.

References:

1. D. Poole and A. Mackworth. Artificial Intelligence: Foundations of Computational Agents, Cambridge University Press, 2010 Available online: *http://artint.info/*
2. Dan W Patterson, Introduction to Artificial Intelligence,Pearson,2009
3. Deepak Khemeni,A First course in Artificial Intelligence,Tata McGraw Hill,2013
4. Maja J. Mataric ,Robotics Primer,MIT press,2007
5. Patrick Henry Winston,Artificial intelligence,Addisson wessley,1992
6. Stefan Edelkamp, Stefan Schroedl, Heuristic Search: Theory and Applications, Morgan Kaufman, 2011.
7. Stuart Jonathan Russell, Peter Norvig, Artificial intelligence, A modern approach,3rd edition, pearson,2010

Course Plan			
Module	**Contents**	**Hours**	**End Sem. Exam Marks**
I	**Introduction**: What is AI, The foundations of AI, History and applications, Production systems. Structures and strategies for state space search. Informed and Uninformed searches.	5	15%
II	**Search Methods:** data driven and goal driven search. Depth first and breadth first search, DFS with iterative deepening. Heuristic search-best first search, A * algorithm.AO* algorithm, Constraint Satisfaction. Crypt Arithmetic Problems	8	15%
FIRST INTERNAL EXAMINATION			
III	**AI representational schemes-** Semantic nets, conceptual dependency, scripts, frames, introduction to agent based problem solving, Machine learning-symbol based-a frame work for symbol based learning.	6	15%
IV	**Advanced Search:** Heuristics in Games, Design of good heuristic-an example. Min-Max Search Procedure, Alpha Beta pruning,	6	15%
SECOND INTERNAL EXAMINATION			
V	**Learning Concepts:** Version space search. Back propagation learning. Social and emergent models of learning-genetic algorithm, classifier systems and genetic programming.	9	20%
VI	**Expert Systems:** rule based expert systems. Natural language processing-natural language understanding problem, deconstructing language. Syntax stochastic tools for language analysis, natural language applications	9	20%
END SEMESTER EXAM			

Question Paper Pattern (End semester exam)

1. There will be ***FOUR*** parts in the question paper – **A, B, C, D**
2. **Part A**
   a. **Total marks : 40**
   b. ***TEN*** questions, each have **4 marks**, covering **all the SIX modules** (***THREE*** questions from **modules I & II**; ***THREE*** questions from **modules III & IV**; ***FOUR*** questions from **modules V & VI**).
   All the TEN questions have to be answered.
3. **Part B**
   a. **Total marks : 18**
   b. ***THREE*** questions, each having **9 marks**. One question is from **module I**; one question is from **module II**; one question ***uniformly*** covers **modules I & II**.
   c. ***Any TWO*** questions have to be answered.
   d. Each question can have ***maximum THREE*** subparts.
4. **Part C**
   a. **Total marks : 18**
   b. ***THREE*** questions, each having **9 marks**. One question is from **module III**; one question is from **module IV**; one question ***uniformly*** covers **modules III & IV**.
   c. ***Any TWO*** questions have to be answered.
   d. Each question can have ***maximum THREE*** subparts.
5. **Part D**
   a. **Total marks : 24**
   b. ***THREE*** questions, each having **12 marks**. One question is from **module V**; one question is from **module VI**; one question ***uniformly*** covers **modules V & VI**.
   c. ***Any TWO*** questions have to be answered.
   d. Each question can have ***maximum THREE*** subparts.
6. There will be ***AT LEAST 60%*** analytical/numerical questions in all possible combinations of question choices.

QUESTION BANK

Q:NO:	QUESTIONS	CO	KL
	MODULE I		
1	Explain Turing Test in detail.	CO1	K1
2	You are given a 4l jug and a 3 l jug. Neither has measuring mark on it.You have to measure exactly 2l of water in the 4l jug.Define the production rules for solving the problem.	CO1	K2
3	Explain the goals of Artificial inteligence in detail.	CO1	K4
4	Explain the production systems in detail.	CO1	K5
5	Explain the application areas in detail.	CO1	K2
6	Distinguish between informed and uninformed search.	CO1	K5
7	Point out the importance of knowledge representation, learning, natural language processing, perception and social intelligence in artificial intelligence	CO1	K2
	MODULE II		
1	Distinguish between data driven search and goal driven search strategies.	CO2	K1
2	Explain the generate and test method.	CO2	K2
3	Describe iterative deepening with DFS.	CO2	K4
4	Explain the concept of DSP.	CO2	K5
5	Explain Best First Search.	CO2	K2
6	Explain problem reduction and AND/OR graph.	CO2	K4
7	Explain DFS and BFS.	CO2	K5
8	Explain A* Algorithm.	CO2	K2
9	Explain AO* Algorithm.	CO2	K2

MODULE III			
1	What are the various components of scripts?	CO3	K2
2	Write short notes on machine learning.	CO3	K2
3	Write short notes on conceptual dependency.	CO3	K2
4	Compare scripts and frames in AI.	CO3	K4
5	Explain the concept of agent based problem solving.	CO3	K5
6	Write short note on Semantic Nets.	CO3	K2
7	Illustrate the method of representation using frames	CO3	K2
8	What are the various components of scripts?	CO3	K5
9	Write short notes on machine learning.	CO3	K2
MODULE IV			
1	Describe the procedure of alpha beta pruning.	CO4	K2
2	Describe in detail about min-max procedure.	CO4	K2
3	Explain n-ply look ahead. Discuss its disadvantages.	CO4	K4
4	How min max procedure is implemented in exhaustively searchable state spaces. Explain using any 2 person game.	CO4	K5
5	Explain the concept of two agent game.	CO4	K1
6	Illustrate the working of min-max procedure in tic-tac-toe game.	CO4	K2
7	Differentiate between min-max and alpha beta procedure.	CO4	K5
8	Explain good heuristic in detail.	CO4	K2
9	Explain heuristic in games.	CO4	K2
10	Explain the design of good heuristic in detail.	CO4	K2
11	Describe the procedure of alpha beta pruning.	CO4	K2

MODULE V			
1	Define version space search .Give three generalization operations used in ML with example.	CO5	K2
2	Give an example for concept space with suitable diagram mentioning its properties and values.	CO5	K4
3	Elaborate on general to specific search algorithm.	CO5	K2
4	Write the algorithm for candidate elimination algorithm.	CO5	K5
5	Explain the network topology of NETtalk.	CO5	K2
6	Describe the concept of back propagation learning.	CO5	K5
7	Differentiate between formal, informal and non-formal learning.	CO5	K2
8	Explain briefly the operators involved in genetic algorithm.	CO5	K2
9	Explain the encoding methods followed in Genetic algorithm.	CO5	K2
10	Explain classifier systems.	CO5	K2
11	Describe the concept of learning with reference to back propagation.	CO5	K5
12	Explain briefly genetic programming.	CO5	K2
13	Define version space search .Give three generalization operations used in ML with example.	CO5	K2
MODULE VI			
1	List out the applications of Natural language processing.	CO6	K2
2	Define noun phrase and verb phrase with example.	CO6	K2
3	Draw the parse for the input | he brought the book | using the given grammer. VD→ verb| verb NP S→ NP VP	CO6	K5

	NP→ Pronoun| Det nominal Nominal→ Noun		
4	Differentiate the expert system from knowledge based system.	CO6	K4
5	Differentiate the syntax and semantic analysis phases in natural language analysis.	CO6	K3
6	Different types of ambiguous present in natural language, processing.	CO6	K5
7	Explain the concept of natural language understanding.	CO6	K3

APPENDIX 1

CONTENT BEYOND THE SYLLABUS

S:NO;	TOPIC	PAGE NO:
1	Reinforcement learning	140

MODULE NOTES

MODULE-1

WHAT IS AI

Artificial Intelligence (AI) is a branch of Science which deals with helping machines finding solutions to complex problems in a more human-like fashion. 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 behaviour appears.

AI is generally associated with Computer Science, but it has many important links with other fields such as Maths, Psychology, Cognition, Biology and Philosophy, among many others. Our ability to combine knowledge from all these fields will ultimately benefit our progress in the quest of creating an intelligent artificial being. AI currently encompasses a huge variety of subfields, from general-purpose areas such as perception and logical reasoning, to specific tasks such as playing chess, proving mathematical theorems, writing poetry, and diagnosing diseases. Often, scientists in other fields move gradually into artificial intelligence, where they find the tools and vocabulary to systematize and automate the intellectual tasks on which they have been working all their lives. Similarly, workers in AI can choose to apply their methods to any area of human intellectual endeavour. In this sense, it is truly a universal field.

10 Definitions of Artificial intelligence

1. AI is the study of how to make computers do things which at the moment people do better. This is ephemeral as it refers to the current state of computer science and it excludes a major area ; problems that cannot be solved well either by computers or by people at the moment.

2. AI is a field of study that encompasses computational techniques for performing tasks that apparently require intelligence when performed by humans.

3. AI is the branch of computer science that is concerned with the automation of intelligent behaviour. A I is based upon the principles of computer science namely data structures used in knowledge representation, the algorithms needed to apply that knowledge and the languages and programming techniques used in their implementation.

4. AI is the field of study that seeks to explain and emulate intelligent behaviour in terms of computational processes.

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

6. A I is the part of computer science concerned with designing intelligent computer systems, that is, computer systems that exhibit the characteristics we associate with intelligence in human behaviour such as understanding language, learning, reasoning and solving problems.

7. A I is the study of mental faculties through the use of computational models.

8. A I is the study of the computations that make it possible to perceive, reason, and act.

9. A I is the exciting new effort to make computers think machines with minds, in the full and literal sense.

10. AI is concerned with developing computer systems that can store knowledge and effectively use the knowledge to help solve problems and accomplish tasks.

HISTORY OF AI

The origin of artificial intelligence lies in the earliest days of machine computations. During the 1940s and 1950s, AI begins to grow with the emergence of the modern computer. Among the first researchers to attempt to build intelligent programs were Newell and Simon. Their first well known program, logic theorist, was a program that proved statements using the accepted rules of logic and a problem solving program of their own design. By the late fifties, programs existed that could do a passable job of translating technical documents and it was seen as only a matter of extra databases and more computing power to apply the techniques to less formal, more ambiguous texts. Most problem solving work revolved around the work of Newell, Shaw and Simon, on the general problem solver (GPS). Unfortunately the GPS did not fulfill its promise and did not because of some simple lack of computing capacity. In the 1970's the most important concept of AI was developed known as Expert System which exhibits as a set rules the knowledge of an expert. The application area of expert system is very large. The 1980's saw the development of neural networks as a method learning examples. Prof. Peter Jackson (University of Edinburgh) classified the history of AI into three periods as:

1. Classical

2. Romantic

3. Modern

1. Classical Period: It was started from 1950. In 1956, the concept of Artificial Intelligence came into existance. During this period, the main research work carried out includes game plying, theorem proving and concept of state space approach for solving a problem.

2. Romantic Period: It was started from the mid 1960 and continues until the mid 1970. During this period people were interested in making machine understand, that is usually mean the understanding of natural language. During this period the knowledge representation technique "semantic net" was developed.

3. Modern Period: It was started from 1970 and continues to the present day. This period was developed to solve more complex problems. This period includes the research on both theories and practical aspects of Artificial Intelligence. This period includes the birth of concepts like Expert system, Artificial Neurons, Pattern Recognition etc. The research of the various advanced concepts of Pattern Recognition and Neural Network are still going on.

HISTORICAL MILESTONES OF AI

Here is the history of AI during 20th century −

Year	Milestone / Innovation
1923	Karel Čapek play named "Rossum's Universal Robots" (RUR) opens in London, first use of the word "robot" in English.
1943	Foundations for neural networks laid.
1945	Isaac Asimov, a Columbia University alumni, coined the term *Robotics*.

1950	Alan Turing introduced Turing Test for evaluation of intelligence and published *Computing Machinery and Intelligence.* Claude Shannon published *Detailed Analysis of Chess Playing* as a search.
1956	John McCarthy coined the term *Artificial Intelligence*. Demonstration of the first running AI program at Carnegie Mellon University.
1958	John McCarthy invents LISP programming language for AI.
1964	Danny Bobrow's dissertation at MIT showed that computers can understand natural language well enough to solve algebra word problems correctly.
1965	Joseph Weizenbaum at MIT built *ELIZA*, an interactive problem that carries on a dialogue in English.
1969	Scientists at Stanford Research Institute Developed *Shakey*, a robot, equipped with locomotion, perception, and problem solving.
1973	The Assembly Robotics group at Edinburgh University built *Freddy*, the Famous Scottish Robot, capable of using vision to locate and assemble models.
1979	The first computer-controlled autonomous vehicle, Stanford Cart, was built.
1985	Harold Cohen created and demonstrated the drawing program, *Aaron*.
1990	Major advances in all areas of AI − • Significant demonstrations in machine learning • Case-based reasoning

	• Multi-agent planning • Scheduling • Data mining, Web Crawler • natural language understanding and translation • Vision, Virtual Reality • Games
1997	The Deep Blue Chess Program beats the then world chess champion, Garry Kasparov.
2000	Interactive robot pets become commercially available. MIT displays *Kismet*, a robot with a face that expresses emotions. The robot *Nomad* explores remote regions of Antarctica and locates meteorites.

C**OMPONENTS OF AI**

There are three types of components in AI

1) Hardware Components of AI

a) Pattern Matching

b) Logic Representation

c) Symbolic Processing

d) Numeric Processing

e) Problem Solving

f) Heuristic Search

g) Natural Language processing

h) Knowledge Representation

i) Expert System

j) Neural Network

k) Learning

l) Planning

m) Semantic Network

2) Software Components

a) Machine Language

b) Assembly language

c) High level Language

d) LISP Language

e) Fourth generation Language

f) Object Oriented Language

g) Distributed Language

h) Natural Language

i) Particular Problem Solving Language

3) Architectural Components

a) Uniprocessor

b) Multiprocessor

c) Special Purpose Processor

d) Array Processor

e) Vector Processor

f) Parallel Processor

g) Distributed Processor

APPLICATIONS OF AI

AI has been dominant in various fields such as −

- **Gaming** − AI plays crucial role in strategic games such as chess, poker, tic-tac-toe, etc., where machine can think of large number of possible positions based on heuristic knowledge.
- **Natural Language Processing** − It is possible to interact with the computer that understands natural language spoken by humans.
- **Expert Systems** − There are some applications which integrate machine, software, and special information to impart reasoning and advising. They provide explanation and advice to the users.
- **Vision Systems** −These systems understand, interpret, and comprehend visual input on the computer. For example,
    - A spying aeroplane takes photographs, which are used to figure out spatial information or map of the areas.

- Doctors use clinical expert system to diagnose the patient.
- Police use computer software that can recognize the face of criminal with the stored portrait made by forensic artist.

- **Speech Recognition** − Some intelligent systems are capable of hearing and comprehending the language in terms of sentences and their meanings while a human talks to it. It can handle different accents, slang words, noise in the background, change in human's noise due to cold, etc.

- **Handwriting Recognition** − The handwriting recognition software reads the text written on paper by a pen or on screen by a stylus. It can recognize the shapes of the letters and convert it into editable text.

- **Intelligent Robots** − Robots are able to perform the tasks given by a human. They have sensors to detect physical data from the real world such as light, heat, temperature, movement, sound, bump, and pressure. They have efficient processors, multiple sensors and huge memory, to exhibit intelligence. In addition, they are capable of learning from their mistakes and they can adapt to the new environment.

GOALS OF AI

Short term goals include:

- **Reasoning, problem solving**

Early researchers developed algorithms that imitated step-by-step reasoning that humans use when they solve puzzles or make logical deductions. By the late 1980s and 1990s, AI research had developed methods for dealing with uncertain or incomplete information, employing concepts from probability and economics.

- **Knowledge representation**

Knowledge representation and knowledge engineering are central to classical AI research. Some "expert systems" attempt to gather together explicit knowledge possessed by

experts in some narrow domain. In addition, some projects attempt to gather the "commonsense knowledge" known to the average person into a database containing extensive knowledge about the world. Among the things a comprehensive commonsense knowledge base would contain are: objects, properties, categories and relations between objects, situations, events, states and time,[causes and effects.

- **Planning**

  Intelligent agents must be able to set goals and achieve them. They need a way to visualize the future—a representation of the state of the world and be able to make predictions about how their actions will change it—and be able to make choices that maximize the utility.

- **Learning**

  Machine learning, a fundamental concept of AI research since the field's inception, is the study of computer algorithms that improve automatically through experience. There are three types of learning schemes:

- Supervised Learning: The learning will be provided under a supervision of teacher, i.e., the inputs as well as output for the machine will be provided and system checks for the errors with the expected output.

- Unsupervised Learning: Here there is no supervision and the system continuously works with the same input until it finds out the output pattern by itself.

- Reinforcement Learning: This is similar to unsupervised learning, as there is no supervision, instead the system receives a response from the environment it is working with. According to the response whether good or bad, the system understands whether errors are present or not.

- **Natural language processing**

  Natural language processing (NLP) gives machines the ability to read and understand human language. A sufficiently powerful natural language processing system would enable natural-language user interfaces and the acquisition of knowledge directly from human-written sources, such as newswire texts. Some straightforward applications of natural language processing include information retrieval, text mining, question answering and machine translation.

- **Perception**

  Machine perception is the ability to use input from sensors (such as cameras (visible spectrum or infrared), microphones, wireless signals, and active lidar, sonar, radar, and tactile sensors) to deduce aspects of the world. Applications include speech recognition, facial recognition, and object recognition.

- **Motion and manipulation**

  AI is heavily used in robotics. Advanced robotic arms and other industrial robots, widely used in modern factories, can learn from experience how to move efficiently despite the presence of friction and gear slippage. A modern mobile robot, when given a small, static, and visible environment, can easily determine its location and map its environment; however, dynamic environments, such as (in endoscopy) the interior of a

patient's breathing body, pose a greater challenge. Motion planning is the process of breaking down a movement task into "primitives" such as individual joint movements. Such movement often involves compliant motion, a process where movement requires maintaining physical contact with an object.

- Long term goals include:
- **Social Intelligence**
- **General Intelligence**

UNINFORMED AND INFORMED SEARCH

- **Uninformed search** is a searching technique which have no additional information about the distance from current state to the goal.
- **Informed Search** is another technique which have additional information about the estimate distance from the current state to the goal.

• **UNINFORMED SEARCH**	• **INFORMED SEARCH**
• It have access only to the problem definition	• It have access to heuristic function as well as problem definition.
• Less efficient	• More efficient
• Every action is equally good	• Every action is not equally good
• Many problems are not solved by uniformed search	• Most of the problem are solved by informed search
• Uninformed search is also known as blind search	• Informed search is also known as heuristic search

• It can handle large search problem	• It cannot handle large search problem
• It use more computation	• It use less computation
• Examples include DFS,BFS,Blind Search.	• Examples include Best first search,A|*,AO *

PRODUCTION SYSTEM AND ITS CHARACTERISTICS

The production system is a model of computation that can be applied to implement search algorithms and model human problem solving. Such problem solving knowledge can be packed up in the form of little quanta called productions. A production is a rule consisting of a situation recognition part and an action part. A production is a situation-action pair in which the left side is a list of things to watch for and the right side is a list of things to do so. When productions are used in deductive systems, the situation that trigger productions are specified combination of facts. The actions are restricted to being assertion of new facts deduced directly from the triggering combination. Production systems may be called premise conclusion pairs rather than situation action pair.

A production system consists of following components.

(a) A set of production rules, which are of the form A→B. Each rule consists of left hand side constituent that represent the current problem state and a right hand side that represent an output state. A rule is applicable if its left hand side matches with the current problem state.

(b) A database, which contains all the appropriate information for the particular task. Some part of the database may be permanent while some part of this may pertain only to the solution of the current problem.

(c) A control strategy that specifies order in which the rules will be compared to the database of rules and a way of resolving the conflicts that arise when several rules match simultaneously.

(d) A rule applier, which checks the capability of rule by matching the content state with the left hand side of the rule and finds the appropriate rule from database of rules.

The important roles played by production systems include a powerful knowledge representation scheme. A production system not only represents knowledge but also action. It acts as a bridge between AI and expert systems. Production system provides a language in which the representation of expert knowledge is very natural.

We can represent knowledge in a production system as a set of rules of the form

If (condition) THEN (condition)

along with a control system and a database. The control system serves as a rule interpreter and sequencer. The database acts as a context buffer, which records the conditions evaluated by the rules and information on which the rules act. The production rules are also known as condition – action, antecedent – consequent, pattern – action, situation – response, feedback – result pairs.

For example, If (you have an exam tomorrow) THEN (study the whole night) The production system can be classified as monotonic, non-monotonic, partially commutative and commutative.

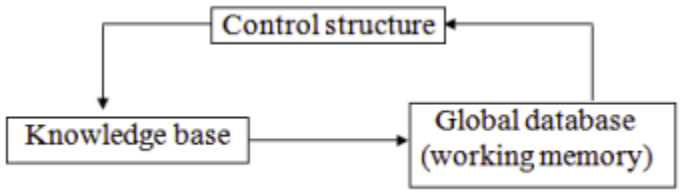


Figure Architecture of Production System

Features of Production System

Some of the main features of production system are: Expressiveness and intuitiveness: In real world, many times situation comes like “if this happen-you will do that”, “if this is so-then this should happen” and many more. The production rules essentially tell us what to do in a given situation.

1. Simplicity: The structure of each sentence in a production system is unique and uniform as they use “IF-THEN” structure. This structure provides simplicity in knowledge representation. This feature of production system improves the readability of production rules.

2. Modularity: This means production rule code the knowledge available in discrete pieces. Information can be treated as a collection of independent facts which may be added or deleted from the system with essentially no deletetious side effects.

3. Modifiability: This means the facility of modifying rules. It allows the development of production rules in a skeletal form first and then it is accurate to suit a specific application.

4. Knowledge intensive: The knowledge base of production system stores pure knowledge. This part does not contain any type of control or programming information. Each production rule is normally written as an English sentence; the problem of semantics is solved by the very structure of the representation. **Disadvantages of production system**

1. Opacity: This problem is generated by the combination ofproduction rules. The opacity is generated because of less prioritization of rules. More priority to a rule has the less opacity.

2. Inefficiency: During execution of a program several rules may active. A well devised control strategy reduces this problem. As the rules of the production system are large in number and they are hardly written in hierarchical manner, it requires some forms of complex search through all the production rules for each cycle of control program.

3. Absence of learning: Rule based production systems do not store the result of the problem for future use. Hence, it does not exhibit any type of learning capabilities. So for each time for a particular problem, some new solutions may come.

4. Conflict resolution: The rules in a production system should not have any type of conflict operations. When a new rule is added to a database, it should ensure that it does not have any conflicts with the existing rules.

AI Technique

Intelligence requires knowledge but knowledge possesses less desirable properties such as -

- It is voluminous
- it is difficult to characterise accurately -
- it is constantly changing -

- it differs from data by being organised in a way that corresponds to its application

An AI technique is a method that exploits knowledge that is represented so that - The knowledge captures generalisations; situations that share properties, are grouped together, rather than being allowed separate representation.

- It can be understood by people who must provide it; although for many programs the bulk of the data may come automatically, such as from readings. In many AI domains people must supply the knowledge to programs in a form the people understand and in a form that is acceptable to the program.

- It can be easily modified to correct errors and reflect changes in real conditions. - It can be widely used even if it is incomplete or inaccurate.

- It can be used to help overcome its own sheer bulk by helping to narrow the range of possibilities that must be usually considered.

Problem Spaces and Search

Building a system to solve a problem requires the following steps

- Define the problem precisely including detailed specifications and what constitutes an acceptable solution;

- Analyse the problem thoroughly for some features may have a dominant affect on the chosen method of solution;

- Isolate and represent the background knowledge needed in the solution of the problem;

- Choose the best problem solving techniques in the solution.

Defining the Problem as state Search

To understand what exactly artificial intelligence is, we illustrate some common problems. Problems dealt with in artificial intelligence generally use a common term called 'state'. A state represents a status of the solution at a given step of the problem solving procedure. The solution of a problem, thus, is a collection of the problem states. The problem solving procedure applies an operator to a state to get the next state. Then it applies another operator to the resulting state to derive a new state. The process of applying an operator to a state and its subsequent transition to

the next state, thus, is continued until the goal (desired) state is derived. Such a method of solving a problem is generally referred to as state space approach For example, in order to solve the problem play a game, which is restricted to two person table or board games, we require the rules of the game and the targets for winning as well as a means of representing positions in the game. The opening position can be defined as the initial state and a winning position as a goal state, there can be more than one. legal moves allow for transfer from initial state to other states leading to the goal state. However the rules are far too copious in most games especially chess where they exceed the number of particles in the universe 10. Thus the rules cannot in general be supplied accurately and computer programs cannot easily handle them. The storage also presents another problem but searching can be achieved by hashing. The number of rules that are used must be minimised and the set can be produced by expressing each rule in as general a form as possible. The representation of games in this way leads to a state space representation and it is natural for well organised games with some structure. This representation allows for the formal definition of a problem which necessitates the movement from a set of initial positions to one of a set of target positions. It means that the solution involves using known techniques and a systematic search. This is quite a common method in AI.

Formal description of a problem

- Define a state space that contains all possible configurations of the relevant objects, without enumerating all the states in it. A state space represents a problem in terms of states and operators that change states

- Define some of these states as possible initial states;

- Specify one or more as acceptable solutions, these are goal states;

- Specify a set of rules as the possible actions allowed. This involves thinking about the generality of the rules, the assumptions made in the informal presentation and how much work can be anticipated by inclusion in the rules.

Control strategies.

A good control strategy should have the following requirement: The first requirement is that it causes motion. In a game playing program the pieces move on the board and in the water jug problem water is used to fill jugs. The second requirement is that it is systematic, this is a clear

requirement for it would not be sensible to fill a jug and empty it repeatedly nor in a game would it be advisable to move a piece round and round the board in a cyclic way. We shall initially consider two systematic approaches to searching.

Monotonic and Non monotonic Learning :

Monotonic learning is when an agent may not learn any knowledge that contradicts what it already knows. For example, it may not replace a statement with its negation. Thus, the knowledge base may only grow with new facts in a monotonic fashion.

The advantages of monotonic learning are: 1.greatly simplified truth-maintenance 2.greater choice in learning strategies

Non-monotonic learning is when an agent may learn knowledge that contradicts what it already knows. So it may replace old knowledge with new if it believes there is sufficient reason to do so.

The advantages of non-monotonic learning are: 1.increased applicability to real domains, 2.greater freedom in the order things are learned in A related property is the consistency of the knowledge. If an architecture must maintain a consistent knowledge base then any learning strategy it uses must be monotonic.

7- PROBLEM CHARACTERISTICS

A problem may have different aspects of representation and explanation. In order to choose the most appropriate method for a particular problem, it is necessary to analyze the problem along several key dimensions. Some of the main key features of a problem are given below. Is the problem decomposable into set of sub problems? Can the solution step be ignored or undone? Is the problem universally predictable? Is a good solution to the problem obvious without comparison to all the possible solutions? Is the desire solution a state of world or a path to a state? Is a large amount of knowledge absolutely required to solve the problem? Will the solution of the problem required interaction between the computer and the person? The above characteristics of a problem are called as 7-problem characteristics under which the solution must take place.

ALGORITHM OF PROBLEM SOLVING

Any one algorithm for a particular problem is not applicable over all types of problems in a variety of situations. So there should be a general problem solving algorithm, which may work for different strategies of different problems.

Algorithm (problem name and specification)

Step 1: Analyze the problem to get the starting state and goal state.

Step 2: Find out the data about the starting state, goalstate

Step 3: Find out the production rules from initial database for proceeding the problem to goal state.

Step 4: Select some rules from the set of rules that can be applied to data.

Step 5: Apply those rules to the initial state and proceed to get the next state.

Step 6: Determine some new generated states after applying the rules. Accordingly make them as current state.

Step 7: Finally, achieve some information about the goal state from the recently used current state and get the goal state.

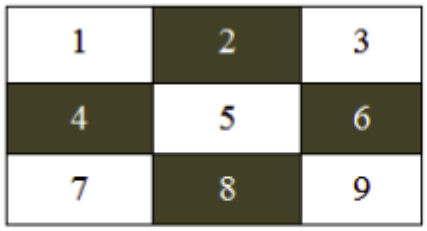
Step 8: Exit. After applying the above rules an user may get the solution of the problem from a given state to another state. Let us take few examples.

VARIOUS TYPES OF PROBLEMS AND THEIR SOLUTIONS

Chess Problem

Definition: It is a normal chess game. In a chess problem, the start is the initial configuration of chessboard. The final state is the any board configuration, which is a winning position for any player. There may be multiple final positions and each board configuration can be thought of as representing a state of the game. Whenever any player moves any piece, it leads to different state of game.

Procedure:



A 3x3 Chess board

Figure

The above figure shows a 3x3 chessboard with each square labeled with integers 1 to 9. We simply enumerate the alternative moves rather than developing a general move operator because of the reduced size of the problem. Using a predicate called move in predicate calculus, whose parameters are the starting and ending squares, we have described the legal moves on the board. For example, move (1, 8) takes the knight from the upper left-hand corner to the middle of the bottom row. While playing Chess, a knight can move two squares either horizontally or vertically followed by one square in an orthogonal direction as long as it does not move off the board. The all possible moves of figure are as follows. Move (1, 8) move (6, 1) Move (1, 6) move (6, 7) Move (2, 9) move (7, 2) Move (2, 7) move (7, 6) Move (3, 4) move (8, 3) Move (3, 8) move (8, 1) Move (4, 1) move (9, 2) Move (4, 3) move (9, 4) The above predicates of the Chess Problem form the knowledge base for this problem. An unification algorithm is used to access the knowledge base. Suppose we need to find the positions to which the knight can move from a particular location, square 2. The goal move (z, x) unifies with two different predicates in the knowledge base, with the substitutions {7/x} and {9/x}. Given the goal move (2, 3), the responsible is failure, because no move (2, 3) exists in the knowledge base. Comments: In this game a lots of production rules are applied for each move of the square on the chessboard. A lots of searching are required in this game. Implementation of algorithm in the knowledge base is very important. 8- Queen Problem Definition: "We have 8 queens and an 8x8 Chess board having alternate black and white squares. The queens are placed on the chessboard. Any queen can attack any other queen placed on same row, or column or diagonal. We have to find the proper placement of queens on the Chess board in such a way that no queen attacks other queen". Procedure: Figure A possible board configuration of 8 queen problem In figure , the possible board configuration for 8-queen problem has been shown. The board has alternative black and white positions on it. The different positions on the board hold the queens. The production rule

for this game is you cannot put the same queens in a same row or same column or in same diagonal. After shifting a single queen from its position on the board, the user have to shift other queens according to the production rule. Starting from the first row on the board the queen of their corresponding row and column are to be moved from their original positions to another position. Finally the player has to be ensured that no rows or columns or diagonals of on the table is same. Comments: This problem requires a lot of space to store the board. It requires a lot of searching to reach at the goal state. The time factor for each queen's move is very lengthy. The problem is very strict towards the production rules.

Water Jug Problem

Definition:

Some jugs are given which should have non-calibrated properties. At least any one of the jugs should have filled with water. Then the process through which we can divide the whole water into different jugs according to the question can be called as water jug problem.

You are given two jugs,a 4 gallon one and a 3 gallon one, Neither has any measuring markers on it.there is a pump that can be used to fill the jugs with water.How can you get exactly 2 gallons of water into the 4 gallon jug ?

The state space for this problem can be described assa the set of ordered pairs of integers(x,y), such that x is 0,1,2,3, or 4 and y is 0,1,2or 3.x represents the number of gallons of water in the 4-gallon jug and y represents the gallons of water in the 3-gallon jug.the start state is(0,0).The goal state is (2,n) for any value of n.

Production rules for the water jug problem

1	(x,y) if x<4	(4,y)	Fill the 4 gallon jug.
2	(x,y) if y<3	(x,3)	Fill the 3 gallon jug.
3	(x,y) if x>0	(x-d,y)	Pour some water out of the 4 gallon jug.
4	(x,y) if y>0	(x,y-d)	Pour some water out of the 3 gallon jug.
5	(x,y)	(0,y)	Empty the 4 gallon jug on the ground.

	if x>0		
6	(x,y) if y>0	(x,0)	Empty the 3 gallon jug in the ground.
7	(x,y) if x+y>=4 and y>0	(4,y-(4-x))	Pour the water from 3 gallon jug into the 4 gallon jug untill the 4 gallon jug is full.
8	(x,y) if x+y>=3 and x>0	(x-(3-y),3)	Pour the water from 4 gallon jug into the 3 gallon jug untill the 3 gallon jug is full.
9	(x,y) if x+y<=4 and y>0	(x+y,0)	Pour all the water from 3 gallon jug into the 4 gallon jug.
10	(x,y) if x+y<=3 and x>0	(0,x+y)	Pour all the water from 4 gallon jug into the 3gallon jug.
11	(0,2)	(2,0)	Pour the 2 gallons from the 3 gallon jug into the 4 gallon jug.
12	(2,y)	(0,y)	Empty the 2 gallons in the 4 gallon jug on the ground.

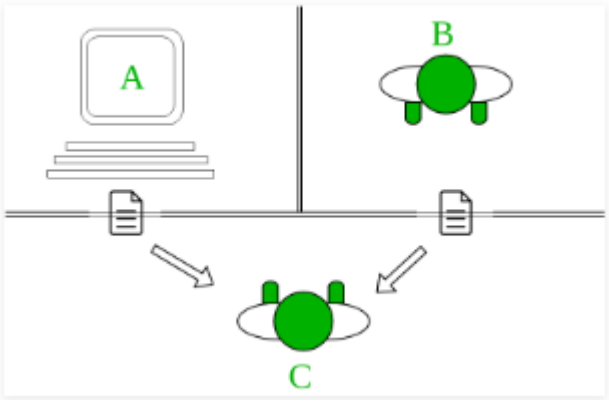
One solution to the water jug problems

Gallons in the 4 gallon jug	Gallons in the 3 gallon jug	Rule applied
0	0	
0	3	2
3	0	9 2
3	3	7
4	2	5 or 12
0	2	
2	0	9 or 11

Turing Test in Artificial Intelligence

The **Turing test** developed by Alan Turing(Computer scientist) in 1950. He proposed that "Turing test is used to determine whether or not computer(machine) can think intelligently like human"?

Imagine a game of three players having two humans and one computer, an interrogator(as human) is isolated from other two players. The interrogator job is to try and figure out which one is human and which one is computer by asking questions from both of them. To make the things harder computer is trying to make the interrogator guess wrongly. In other words computer would try to indistinguishable from human as much as possible.



The "standard interpretation" of the Turing Test, in which player C, the interrogator, is given the task of trying to determine which player – A or B – is a computer and which is a human. The interrogator is limited to using the responses to written questions to make the determination

The conversation between interrogator and computer would be like this:

C(Interrogator): Are you a computer?

A(Computer): No

C: Multiply one large number to another, 158745887 * 56755647

A: After a long pause, an incorrect answer!

C: Add 5478012, 4563145

A: (Pause about 20 second and then give as answer)10041157

If interrogator wouldn't be able to distinguish the answers provided by both human and computer then the computer passes the test and machine(computer) is considered as intelligent as human. In other words, a computer would be considered intelligent if it's conversation couldn't be easily distinguished from a human's. The whole conversation would be limited to a text-only channel

such as a computer keyboard and screen.

MODULE – 2

DATA DRIVEN AND GOAL DRIVEN SEARCH

Data-Driven Search (Forward Chaining) - takes facts of problem and applies rules or legal moves to produce new facts, leading to goal.

Suggested if:

- All or most of data is given in problem statement; e.g. geology solver to determine what minerals at given site
- Large number of potential goals but few rules/ways to use given facts. e.g. DENDRAL finding molecular structure based on formulae.
- Difficult to form goal/hypothesis

Example:
Prove that the following argument is valid:
(A ^ B) -> (C ^ D)
A

B / Therefore, D

1. (A ^ B) -> (C ^ D)
2. A
3. B
4. A ^ B From 2, 3 by Conjunction
5. C ^ D From 1, 4 by Modus ponens
6. D ^ C From 5 by Commutation
7. D From 6 by Simplification

Goal-Driven Search (Backward Chaining) - uses information about goal, finds rules required to produce goal and conditions necessary for these rules. These conditions become new subgoals.

Suggested if:

Goal/hypothesis is given or easily determined
There are large number of rules matching facts of system, which would lead to many conclusions. Starting with goal eliminates many wrong branches.
Problem data are not given and must be determined by solver.

Example:

Prove that the following argument is valid:

(A ^ B) -> (C ^ D)

A

B / Therefore, D

For the same argument, start with the goal state and work backwards

1. D
2. D ^ C This subgoal can produce 1 by Simplification
3. C ^ D This subgoal can produce 2 by Commutation

4. A ^ B This subgoal
5. (A ^ B) -> (C ^ D) and this premise can produce 3 by Modus Ponens
6. A This premise
7. B and this premise can produce 4 by Conjunction

SEARCHING

Problem solving in artificial intelligence may be characterized as a systematic search through a range of possible actions in order to reach some predefined goal or solution. In AI problem solving by searchalgorithms is quite common technique. In the coming age of AI it will have big impact on the technologies of the robotics and path finding. It is also widely used in travel planning. This chapter contains the different search algorithms of AI used in various applications. Let us look the concepts for visualizing the algorithms.

A search algorithm takes a problem as input and returns the solution in the form of an action sequence. Once the solution is found, the actions it recommends can be carried out. This phase is called as the execution phase. After formulating a goal and problem to solve the agent cells a search procedure to solve it. A problem can be defined by 5 components.

a) **The initial state**: The state from which agent will start.

b) **The goal state:** The state to be finally reached.

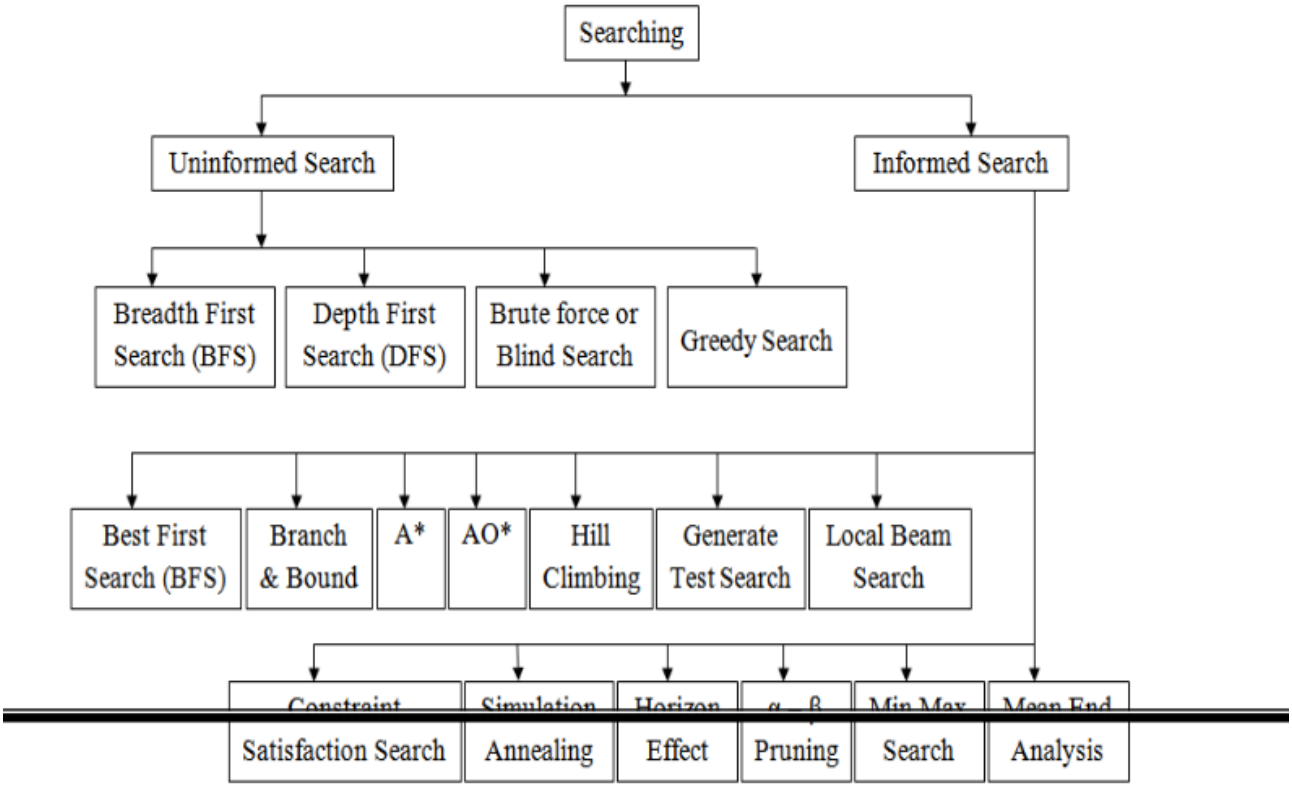
c) **The current state**: The state at which the agent is present after starting from the initial state.

d) **Successor function**: It is the description of possible actions and their outcomes.

e) **Path cost**: It is a function that assigns a numeric cost to each path.

DIFFERENT TYPES OF SEARCHING

the searching algorithms can be various types. When any type of searching is performed, there may some information about the searching or mayn't be. Also it is possible that the searching procedure may depend upon any constraints or rules. However, generally searching can be classified into two types i.e. uninformed searching and informed searching. Also some other classifications of these searches are given below in the figure .



UNINFORMED SEARCH

Breadth First Search (BFS)

Breadth first search is a general technique of traversing a graph. Breadth first search may use more memory but will always find the shortest path first. In this type of search the state space is represented in form of a tree. The solution is obtained by traversing through the tree. The nodes of the tree represent the start value or starting state, various intermediate states and the final state. In this search a queue data structure is used and it is level by level traversal. Breadth first search expands nodes in order of their distance from the root. It is a path finding algorithm that is capable of always finding the solution if one exists. The solution which is found is always the optional solution. This task is completed in a very

memory intensive manner. Each node in the search tree is expanded in a breadth wise at each level.

Concept:

Step 1: Traverse the root node

Step 2: Traverse all neighbours of root node.

Step 3: Traverse all neighbours of neighbours of the root node.

Step 4: This process will continue until we are getting the goal node.

Generalised Algorithm.

1. Create a variable called NODE-LIST and set it to inital state.
2. Until a goal state is found or NODE-LIST is empty do

a) Remove the first element from the NODE-LIST and call it E. If the NODE-LIST was empty ,quit,

b) For each way that each rule can match the state described in E do,

1. Apply the rule to generate a new state.
2. If the new state is a goal state ,quit and return this state.
3. Otherwise,add the new state to the end of the NODE-LIST.

Algorithm:

Step 1: Place the root node inside the queue.

Step 2: If the queue is empty then stops and return failure.

Step 3: If the FRONT node of the queue is a goal node then stop and return success.

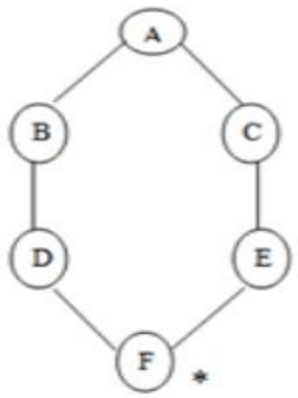
Step 4: Remove the FRONT node from the queue. Process it and find all its neighbours that are in ready state then place them inside the queue in any order.

Step 5: Go to Step 3.

Step 6: Exit.

Implementation:

Let us implement the above algorithm of BFS by taking the following suitable example.



Figure

Consider the graph in which let us take A as the starting node and F as the goal node (*)

Step 1: Place the root node inside the queue i.e. A

A

Step 2:Now the queue is not empty and also the FRONT node i.e. A is not our goal node. So move to step 3.

Step 3:So remove the FRONT node from the queue i.e. A and find the neighbour of A i.e. B and C

B C A

Step 4:Now b is the FRONT node of the queue .So process B and finds the neighbours of B i.e. D.

C D B

Step 5:Now find out the neighbours of C i.e. E

D E C

Step 6: Next find out the neighbours of D as D is the FRONT node of the queue

E F D

Step 7:Now E is the front node of the queue. So the neighbour of E is F which is our goal node.

F E

Step 8: Finally F is our goal node which is the FRONT of the queue. So exit.

F

Advantages:

In this procedure at any way it will find the goal.

It does not follow a single unfruitful path for a long time.

It finds the minimal solution in case of multiple paths.

Disadvantages:

BFS consumes large memory space.

Its time complexity is more.

It has long pathways, when all paths to a destination are on approximately the same search depth.

Depth First Search (DFS)

DFS is also an important type of uniform search. DFS visits all the vertices in the graph. This type of algorithm always chooses to go deeper into the graph. After DFS visited all the reachable vertices from a particular sources vertices it chooses one of the remaining undiscovered vertices and continues the search.DFS reminds the space limitation of breath first search by always generating next a child of the deepest unexpanded nodded. The data structure stack or last in first out (LIFO) is used for DFS. One interestingproperty of DFS is that, the discover and finish time of each vertex from a parenthesis structure. If we use one open parenthesis when a vertex is finished then the result is properly nested set of parenthesis.

Concept:

Step 1: Traverse the root node.

Step 2: Traverse any neighbour of the root node.

Step 3: Traverse any neighbour of neighbour of the root node.

Step 4: This process will continue until we are getting the goal node.

Algorithm:

Step 1: PUSH the starting node into the stack.

Step 2: If the stack is empty then stop and return failure.

Step 3: If the top node of the stack is the goal node, then stop and return success.

Step 4: Else POP the top node from the stack and process it. Find all its neighbours that are in ready state

and PUSH them into the stack in any order.

Step 5: Go to step 3.

Step 6: Exit.

Implementation:

Let us take an example for implementing the above DFS algorithm.

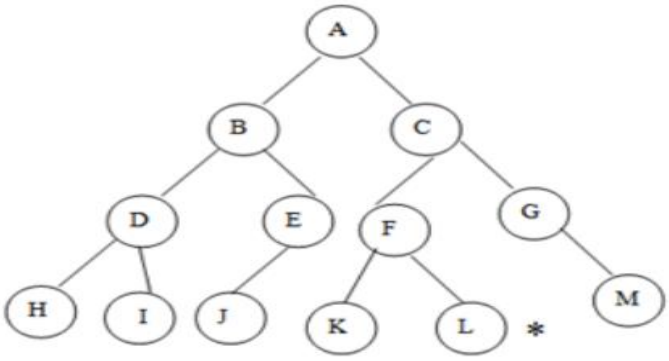
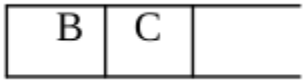


Figure Examples of DFS

Step 1: PUSH the starting node into the stack i.e.

Step 2: Now the stack is not empty and A is not our goal node. Hence move to next step.

Step 3: POP the top node from the stack i.e. A and find the neighbours of A i.e. B and C.

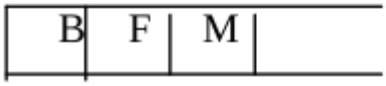


A

Step 4: Now C is top node of the stack. Find its neighbours i.e. F and G.

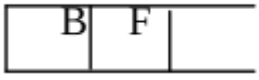
C

Step 5: Now G is the top node of the stack. Find its neighbour i.e. M



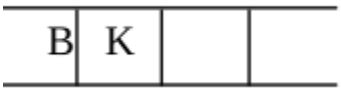
G

Step 6: Now M is the top node and find its neighbour, but there is no neighbours of M in the graph so POP it from the stack.



M

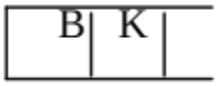
Step 7: Now F is the top node and its neighbours are K and L. so PUSH them on to the stack.



L F

Step 8:

Now L is the top node of the stack, which is our goal node.



L

Generalised Algorithm

1. If the initial state is a goal state ,quit and return sucess.

2. Otherwise, do the following until sucess or failure is signaled.

a) Generate a sucessor,E,of the initial state.If there are no more sucessors signal failure

b) Call DFS with E as the initial state.

c) If the sucess is returned ,signal sucess. Otherwise continue in this loop.

Advantages:

DFSconsumes very less memory space.

It will reach at the goal node in a less time period than BFS if it traverses in a right path.

It may find a solution without examining much of search because we may get the desired solution in the very first go.

Disadvantages:

It is possible that may states keep reoccurring.

There is no guarantee of finding the goal node.

Sometimes the states may also enter into infinite loops.

Brute Force or Blind Search

Brute force or blind search is a uniformed exploration of the search space and it does not explicitly take into account either planning efficiency or execution efficiency. Blind search is also called uniform search. It is the search which has no information about its domain. The only thing that a blind search can do is to differentiate between a non goal state and a goal state. These methods do not need domain knowledge but they are less efficient in result. Uniform strategies

don't use any information about how a close a node might be to a goal. They differ in the order that the nodes are expanded. The most important brute force
techniques are breadth first search, depth first search, uniform search and bidirectional search. All brute force techniques must take (b 0 # time and use o (d) space. This technique is not as efficient as compared to other algorithms.

Difference between BFS and DFS

BFS

It uses the data structure queue.

BFS is complete because it finds the solution if one exists.

BFS takes more space i.e. equivalent to o (b 0) where b is the maximum breath exist in a search tree and d is the maximum depth exit in a search tree.

In case of several goals, it finds the best one.

DFS

It uses the data structure stack.

It is not complete because it may take infinite loop to reach at the goal node.

The space complexity is O (d).

In case of several goals, it will terminate the solution in any order.

Iterative deepening with DFS

iterative deepening search or more specifically **iterative deepening depth-first search** (IDS or IDDFS) is a state space/graph search strategy in which a depth-limited version of depth-first search is run repeatedly with increasing depth limits until the goal is found. IDDFS is optimal like breadth-first search, but uses much less memory; at each iteration, it 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 is effectively breadth-first.

The following pseudocode shows IDDFS implemented in terms of a recursive depth-limited DFS (called DLS) for directed graphs. This implementation of IDDFS does not account for already-visited nodes and therefore does not work for undirected graphs.

```
function IDDFS(root) is
   for depth from 0 to ∞ do
      found, remaining ← DLS(root, depth)
      if found ≠ null then
```

```
        return found
    else if not remaining then
        return null

function DLS(node, depth) is
  if depth = 0 then
    if node is a goal then
      return (node, true)
    else
      return (null, true)    (Not found, but may have children)

  else if depth > 0 then
    any_remaining ← false
    foreach child of node do
      found, remaining ← DLS(child, depth−1)
      if found ≠ null then
        return (found, true)
      if remaining then
        any_remaining ← true    (At least one node found at depth, let IDDFS deepen)
    return (null, any_remaining)
```

If the goal node is found, then **DLS** unwinds the recursion returning with no further iterations. Otherwise, if at least one node exists at that level of depth, the *remaining* flag will let **IDDFS** continue.

2-tuples are useful as return value to signal **IDDFS** to continue deepening or stop, in case tree depth and goal membership are unknown *a priori*. Another solution could use sentinel values instead to represent *not found* or *remaining level* results.

IDDFS combines depth-first search's space-efficiency and breadth-first search's completeness (when the branching factor is finite). If a solution exists, it will find a solution path with the fewest arcs.

Since iterative deepening visits states multiple times, it may seem wasteful, but it turns out to be not so costly, since in a tree most of the nodes are in the bottom level, so it does not matter much if the upper levels are visited multiple times.[4]

The main advantage of IDDFS in game tree searching is that the earlier searches tend to improve

the commonly used heuristics, such as the killer heuristic and alpha-beta pruning, so that a more accurate estimate of the score of various nodes at the final depth search can occur, and the search completes more quickly since it is done in a better order. For example, alpha-beta pruning is most efficient if it searches the best moves first.[4]

A second advantage is the responsiveness of the algorithm. Because early iterations use small values for , they execute extremely quickly. This allows the algorithm to supply early indications of the result almost immediately, followed by refinements as increases. When used in an interactive setting, such as in a chess-playing program, this facility allows the program to play at any time with the current best move found in the search it has completed so far. This can be phrased as each depth of the search *co*recursively producing a better approximation of the solution, though the work done at each step is recursive. This is not possible with a traditional depth-first search, which does not produce intermediate results.

INFORMED SEARCH (HEURISTIC SEARCH)

Heuristic is a technique which makes our search algorithm more efficient. Heuristic is a technique that improves the efficency of a search process ,possibily by sacrificing claims of completeness. Some heuristics help to guide a search process without sacrificing any claim to completeness and some sacrificing it. Heuristic is a problem specific knowledge that decreases expected search efforts. It is a technique which sometimes works but not always. Heuristic search algorithm uses information about the problem to help directing the path through the search space. These searches uses some functions that estimate the cost from the current state to the goal presuming that such function is efficient. *A heuristic function is a function that maps from problem state descriptions to measure of desirability usually represented as number. The purpose of heuristic function is to guide the search process in the most profitable directions by suggesting which path to follow first when more than is available*. Generally heuristic incorporates domain knowledge to improve efficiency over blind search. In AI heuristic has a general meaning and also a more specialized technical meaning. Generally a term heuristic is used for any advice that is effective but is not guaranteed to work in every case. For example in case oftravelling sales man (TSP) problem we are using a heuristic to calculate the nearest neighbour. Heuristic is a method that provides a better guess about the correct choice to make at

any junction that would be achieved by random guessing. This technique is useful in solving though problems which could not be solved in any other way. Solutions take an infinite time to compute.

Generate – and – Test

The generate and test is the simplest of all the techniques.

Algorithm.

1. Generate a possible solution. For some problems it means generating a particular point in the problem space.For others it means generating a path from a start state.
2. Test to see if this is actually a solution by comparing the chosen point or the end point of the chosen path to the set of acceptable goal states.
3. If a solution has been found ,quit.Otherwise reurn to step 1.

The generate and test algorithm is a DFS procedure since complete solutions must be generated before they can be tested.In its most systematic form,it is simply an exhaustive search of the problem space. It can ofcourse also operate by generating solutions randomly,but then there is no guarantee that a solution will be ever found. In this form,it is also known as Bitish museum algorithm, a refernce to a method for finding an object in the british museum by wandering randomly. The most straight forward method to implement systematic generate and test is as a DFS tree with backtracing. For simple problems generate and test is often a reasonable technique.

One early example of sucessful AI program is DENDRALwhich infers the structure of organic compounds using mass spectrogramand nuclear magnetic resonace data.It uses a strategy called plan-generate-test,in which a planning process that uses constarint satisfaction techniques creates a lists of recommended and contradicted substructures,The generate=and -test procedure than uses those lists so that it can explore only a fairly limited set of structures.

Hill Climbing

Hill climbing search algorithm is simply a loop that continuously moves in the direction of increasing value. It stops when it reaches a “peak” where no neighbour has higher value. This algorithm is considered to be one of the simplest procedures for implementing heuristic search. The hill climbing comes from that idea if you are trying to find the top of the hill and you go up

direction from where ever you are. This heuristic combines the advantages of both depth first and breadth first searches into a single method. The name hill climbing is derived from simulating the situation of a person climbing the hill. The person will try to move forward in the direction of at the top of the hill. His movement stops when it reaches atthe peak of hill and no peak has higher value of heuristic function than this. Hill climbing uses knowledge
about the local terrain, providing a very useful and effective heuristic for eliminating much of the unproductive search space. It is a branch by a local evaluation function. The hill climbing is a variant of generate and test in which direction the search should proceed. At each point in the search path, a successor node that appears to reach for exploration.

Algorithm:

Step 1: Evaluate the starting state. If it is a goal state then stop and return success.

Step 2: Else, continue with the starting state as considering it as a current state.

Step 3: Continue step-4 until a solution is found i.e. until there are no new states left to be applied in the current state.

Step 4:

a) Select a state that has not been yet applied to the current state and apply it to produce a new state.

b) Procedure to evaluate a new state.

i. If the current state is a goal state, then stop and return success.

ii. If it is better than the current state, then make it current state and proceed further.

iii. If it is not better than the current state, then continue in the loop until a solution is found.

Step 5: Exit.

Advantages:

Hill climbing technique is useful in job shop scheduling, automatic programming, circuit designing, and vehicle routing and portfolio management.

It is also helpful to solve pure optimization problems where the objective is to find the best state according to the objective function.

It requires much less conditions than other search techniques.

Disadvantages:

The question that remains on hill climbing search is whether this hill is the highest hill possible. Unfortunately without further extensive exploration, this question cannot be answered. This

technique works but as it uses local information that's why it can be fooled. The algorithm doesn't maintain a search tree, so the current node data structure need only record the state and its objective function value. It assumes that local improvement will lead to global improvement.

Best First Search

Best First Search is a way of combining the advantages of both depth first search and breadth first search method.

OR Graphs

Depth first search is good because it allows a solution to be found without all competing branches having to be expanded. Breadth First Search is good because it does not get trapped in dead end paths. One way of combining the two is to follow a single path at a time, but to switch paths whenever some competing path looks more promising than the current one. At each step of Best First Search we select the most promising of the nodes. This is done by applying an appropriate heuristic function to each of them. We then expand the chosen node by using rules to generate its successors. If one of them is a solution we can quit. If not all those new nodes are added to set of nodes generated so far. A bit of Depth First Searching occurs as the most promising branch is explored. If a solution is not found, that branch will start to be less promising than one of the top level branches that had been ignored. The old branch is not forgotten. Its last node remain in the set of generated but unexpected nodes.

Directed Graph Searching (OR Graph)

It is sometimes important to search a graph a graph instead of a tree so that duplicate paths will not be taken. An algorithm to do so will operate by searching a directed graph in which each node represent a point in problem space.

Each node will contains:

l addition to a description of problem state it represents,

l an indication of how promising it is,

l a parent link that points back to best node it came from, and

l a list of nodes that were generated from it

The parent link will make it possible to recover the path to goal once goal is found.

The list of successors will make it possible if a better path is found to an already existing node, to propagate improvement down to its successors. We call this an OR-Graph since each of this branch represents an alternate problem solving path.

To implement this we need two list of nodes

1. OPEN

Nodes that have been generated and have had the heuristic function applied to them but which have not yet been examined(ie had their successors generated). OPEN is a priority queue in which the element with highest priority are those with most promising value of heuristic function.

2. CLOSED

Nodes that have already been examined. We need to keep these in memory if we want to search a graph rather than a tree, since whenever a new node is generated we need to check if it is generated before.

The **heuristic function** must be able to search more promising paths first. Call this function f' (approximation to a function). We can call this function as sum of two components lg measure of cost of getting from initial state to current node.

It is not an estimation It is exact sum of cost of applying each of the rules that were applied along the best path to the node Must be non negative else graph will traverse in a cycles which will appear to get better but will be getting longer **l h'**

It is an estimate of additional cost of getting from current node to goal state.

Good one get low value, bad one get high value

l f' = g + h'

F' represents an estimate of cost of getting from initial state to goal state along the path generated from current node If more than one path generated the node, the best path is record

Algorithm Best First Search

1. Start OPEN containing initial state
2. Until goal is found or there are no nodes present in OPEN do:

a) Pick up the best node on OPEN

b) Generate its successors

c) For each successor do

i. If it has not been generated before, evaluate it add it to open, record its

parent

ii. If it has been generated before, change the parent if the new path is better than previous one. In that case update the cost of getting to this node and to any of its successors that this node may already

The A* Algorithm

The best first search algorithm is a simplification of A* algorithm. This algorithm uses the same heuristic functions f', g and h' and also the list OPEN and CLOSED.

Algorithm A*

1. Start with OPEN containing only the initial node. Set that nodes g value to 0, and its h' value to whatever it is. And its f' value to h'+0 ie h'. Set CLOSED to an empty list

2. Until a goal node is found, repeat the following procedure:

If there is no node on OPEN, report failure

Otherwise pick a node on OPEN with lowest f' value. Call it BESTNODE.

Remove it from OPEN. Place it on CLOSED. See if BESTNODE is goal node. If so exit and report solution. Otherwise generate successor of BESTNODE but do not set BESTNODE to point to them yet.

For each such SUCCESOR do the following:

a) Set SUCCESOR to point back to BESTNODE. These backward links will make it possible to recover the path once a solution is found.

b) Compute g(SUCCESOR) = g(BESTNODE) + cost of getting from BESTNODE to SUCCESOR

c) See if successor is same as any node in OPEN

If so call that node OLD throw SUCCESOR away and add OLD to list of BESTNODE's successors. Now we must decide whether OLDS's parent link should be reset to point to BESTNODE if path we have just found is cheaper than current best path to OLD (since SUCCESOR and OLD are same nodes)

So whether it is cheaper to get to OLD via its current parent or to SUCCESOR via BESTNODE comparing their g values. If OLD is cheaper then do nothing. If SUCCESOR then reset OLD's parent link to point to BESTNODE, record new cheaper path in g(OLD), and update f'(OLD).

d) If SUCCESSOR was not on OPEN, see if it is on CLOSED. If so so call the node on CLOSED OLD node and add it to list of BESTNODES successors.

Check to see if new path or old path is better and set parent link and g and f' function accordingly.

If we have found a better path to CLOSED OLD we must propagate improvements to OLD's Successor. This is a tricky process.

OLD points to its successors. Each successor in turn points to its Successors and so forth until each branch terminates with a node that is either is still on OPEN or has no successors.

while propagation check if path through current parent is still better than new path then stop propagation

e) if SUCCESSOR was not already on either OPEN or CLOSED, then put it on OPEN add its to list of BESTNODES's successor.

Compute

f'(SUCCESSOR)=g(SUCCESSOR)+h'(SUCCESSOR)

A* is a cornerstone name of many AI systems and has been used since it was developed in 1968 by Peter Hart; Nils Nilsson and Bertram Raphael. It is the combination of Dijkstra's algorithm and Best first search. It can be used to solve many kinds of problems. A* search finds the shortest path through a search

space to goal state using heuristic function. This technique finds minimal cost solutions and is directed to a goal state called A* search. In A*, the * is written for optimality purpose. The A* algorithm also finds the lowest cost path between the start and goal state, where changing from one state to another requires some cost. A* requires heuristic function to evaluate the cost of path that passes through the particular state. This algorithm is complete if the branching factor is finite and every action has fixed cost. A* requires heuristic function to evaluate the cost of path that passes through the particular state. It can be defined by following formula.

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

Where

g (n): The actual cost path from the start state to the current state.

h (n): The actual cost path from the current state to goal state.

f (n): The actual cost path from the start state to the goal state.

For the implementation of A* algorithm we will use two arrays namely OPEN and CLOSE.

OPEN:

An array which contains the nodes that has been generated but has not been yet examined.

CLOSE:

An array which contains the nodes that have been examined.

Algorithm:

Step 1: Place the starting node into OPEN and find its f (n) value.

Step 2: Remove the node from OPEN, having smallest f (n) value. If it is a goal node then stop and return success.

Step 3: Else remove the node from OPEN, find all its successors.

Step 4: Find the f (n) value of all successors; place them into OPEN and place the removed node into CLOSE.

Step 5: Go to Step-2.

Step 6: Exit.

Implementation:

The implementation of A* algorithm is 8-puzzle game.

Advantages:

It is complete and optimal.

It is the best one from other techniques.

It is used to solve very complex problems.

It is optimally efficient, i.e. there is no other optimal algorithm guaranteed to expand fewer nodes than A*.

Disadvantages:

This algorithm is complete if the branching factor is finite and every action has fixed cost.

The speed execution of A* search is highly dependant on the accuracy of the heuristic algorithm that is used to compute h (n).

It has complexity problems.

MODULE 3

KNOWLEDGE

Knowledge is the collection of facts, inference rules etc. which can be used for a particular purpose.

Knowledge requires the use of data and information. It combines relationships, correlations, dependencies with data and information.

The basic components of knowledge are:

1) A set of collected data

2) A form of belief or hypothesis

3) A kind of information.

Knowledge is different from data. Data is the collection of raw materials where as knowledge is the

collection of some well specified inference rules and facts. Knowledge is also different from belief and hypothesis. Belief is any meaningful and coherent expression that can be represented. Belief may be true or false. A hypothesis is a justified belief that is not known to be true. A hypothesis is a belief which is backed up with some supporting evidence but it may still be false. So knowledge can be defined as true justified knowledge.

KNOWLEDGE BASED SYSTEMS

Knowledge based systems get their power from the expert knowledge that has been coded into facts, rules, heuristics and procedures. The knowledge is stored in a knowledge base separate from the control and inferencing components. Knowledge is important and essential for knowledge based intelligent behaviour.

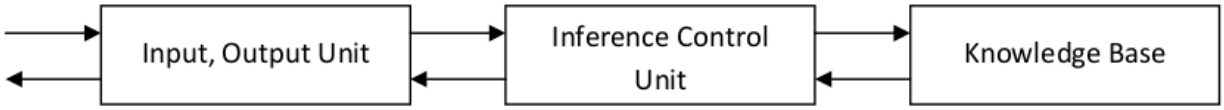


Figure A typical Knowledge based system

Any choice of representation will depend on the type of problem to be solved and the inference methods available. Knowledge may be vague, contradictory or incomplete. Thus, knowledge is information about objects, concepts and relationships that are assumed to exist in a particular area of interest.

KNOWLEDGE REPRESENTATION

Knowledge representation is probably, the most important ingredient for developing an AI. A

representation is a layer between information accessible from outside world and high level thinking

processes. Without knowledge representation it is impossible to identify what thinking processes are, mainly because representation itself is a substratum for a thought. The subject of knowledge representation has been messaged for a couple of decades already. For many applications, specific domain knowledge is required. Instead of coding such knowledge into a system in a

way that it can never be changed (hidden in the overall implementation), more flexible ways of

representing knowledge and reasoning about it have been developed in the last 10 years.

The need of knowledge representation was felt as early as the idea to develop intelligent systems. With the hope that readers are well conversant with the fact by now, that intelligent requires possession of nowledge and that knowledge is acquired by us by various means and stored in the memory using some representation techniques. Putting in another way, knowledge representation is one of the many critical aspects, which are required for making a computer behave intelligently. Knowledge representation refers to the data structures techniques and organizing notations that are used in AI. These include semantic networks, frames, logic, production rules and conceptual graphs.

Properties for knowledge Representation

The following properties should be possessed by a knowledge representation system.

a. Representational Adequacy: It is the ability to represent the required knowledge.

b. Inferential Adequacy: It is the ability to manipulate the knowledge represented to produce new knowledge corresponding to that inferred from the original.

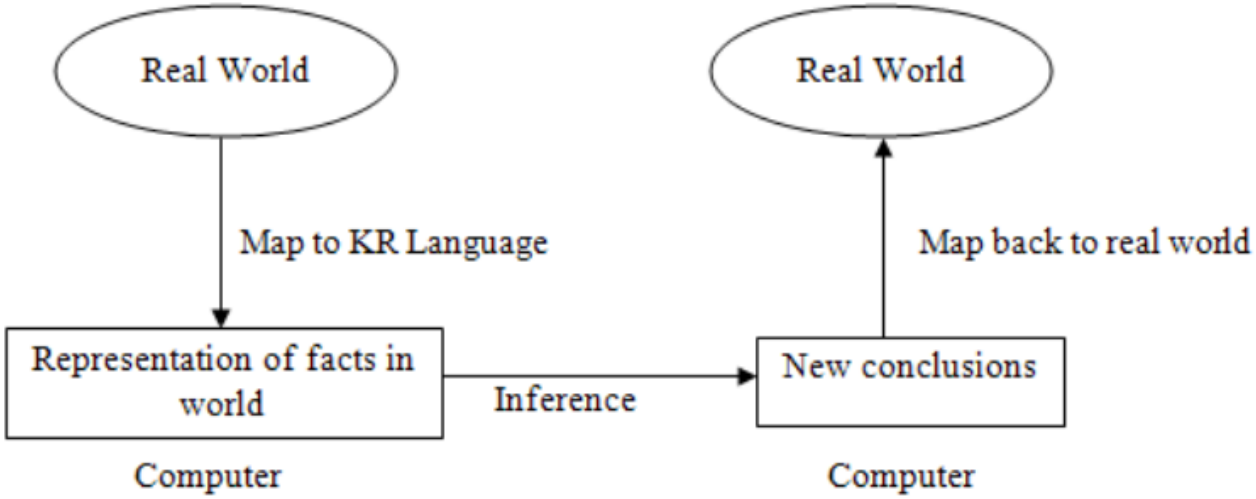
c. Inferential Efficiency: The ability to direct the inferential mechanisms into the most productive directions by storing appropriate guides.

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

Syntax and semantics for Knowledge Representation

Knowledge representation languages should have precise syntax and semantics. You must know exactly what an expression means in terms of objects in the real world. Suppose we have decided that "red 1" refers to a dark red colour, "car1" is my car, car2 is another. Syntax of language will tell you which of the following is legal: red1 (car1), red1 car1, car1 (red1), red1 (car1 & car2)?

Semantics of language tell you exactly what an expression means: for example, Pred (Arg) means that the property referred to by Pred applies to the object referred to by Arg. E.g., properly "dark red" applies to my car.



Types of Knowledge Representation

Knowledge can be represented in different ways. The structuring of knowledge and how designers might view it, as well as the type of structures used internally are considered. Different knowledge

representation techniques are

a. Logic

b. Semantic Network

c. Framed. Conceptual Graphs

e. Conceptual Dependency

f. Script

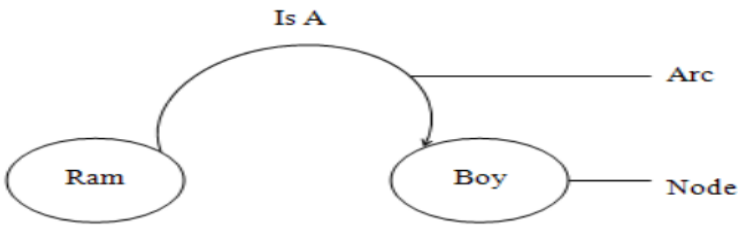
Semantic Network

A semantic network is a graphical knowledge representation technique. This knowledge representation system is primarily on network structure. The semantic networks were basically

developed to model human memory. A semantic net consists of nodes connected by arcs. The arcs are defined in a variety of ways, depending upon the kind of knowledge being represented.

The main idea behind semantic net is that the meaning of a concept comes, from the ways in which it is connected to other concepts. The semantic network consists of different nodes and arcs. Each node should contain the information about objects and each arc should contain the relationship between objects. Semantic nets are used to find relationships among objects by spreading activation about from each of two nodes and seeing where the activation met this process is called intersection search.

For example: Ram is a boy.



Figure

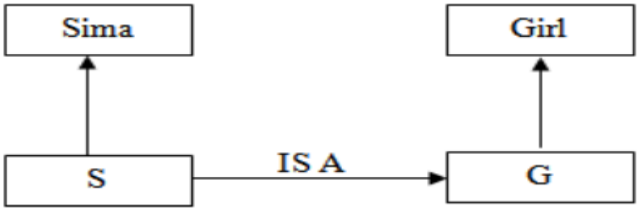
Semantic network by using Instances

The semantic network based knowledge representation mechanism is useful where an object or concept is associated with many attributes and where relationships between objects are important. Semantic nets have also been used in natural language research to represent complex

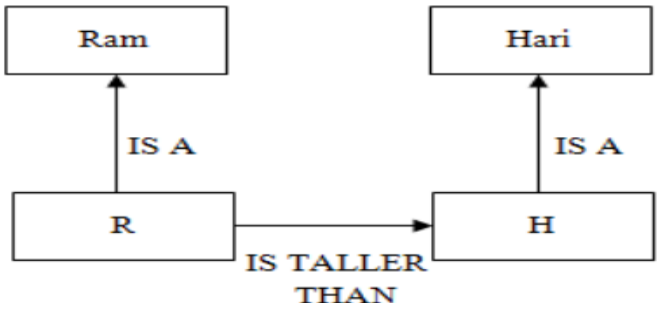
sentences expressed in English. The semantic representation is useful because it provides a standard way of analyzing the meaning of sentence. It is a natural way to represent relationships that would appear as ground instances of binary predicates in predicate logic. In this case we can create one instance of each object. In instance based semantic net representations some keywords are used like: IS A, INSTANCE, AGENT, HAS-PARTS etc.

Consider the following examples:

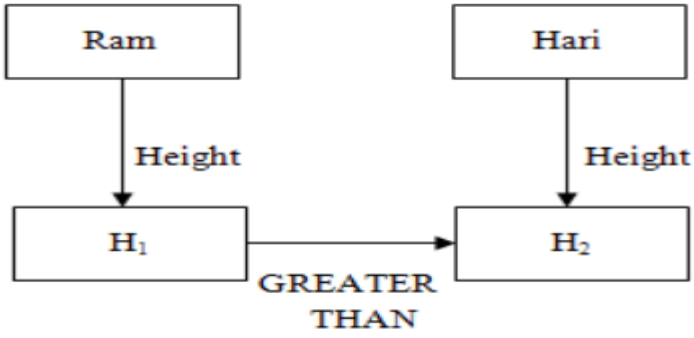
1. Suppose we have to represent the sentence “Sima is a girl”.



2. Ram is taller than Hari



It can also be represented as

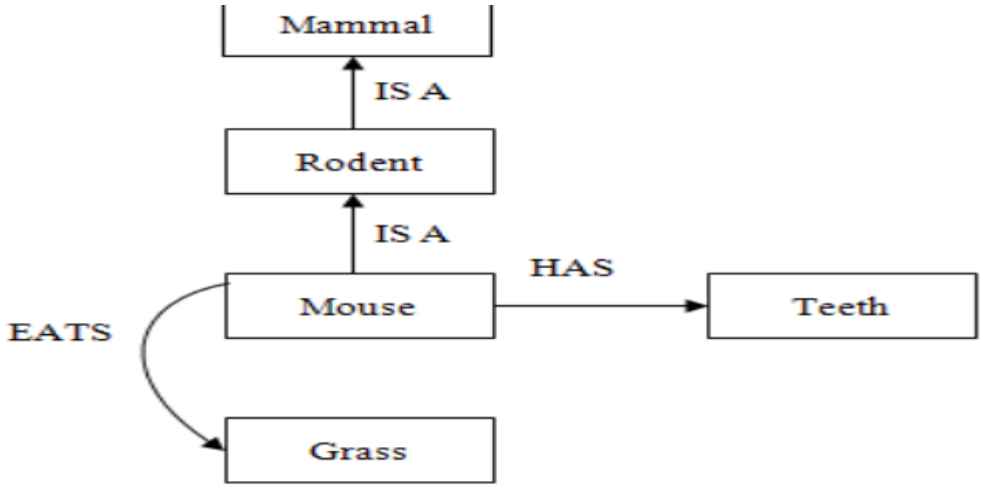


b)

3. “Mouse is a Rodent and Rodent is a mammal. Mouse has teeth and etas grass”. Check whether the sentence mammal has teeth is valid or not.

(c)

Partitioned Semantic Networ



k

Some complex sentences are there which cannot be represented by simple semantic nets and for this we have to follow the technique partitioned semantic networks. Partitioned semantic net allow for

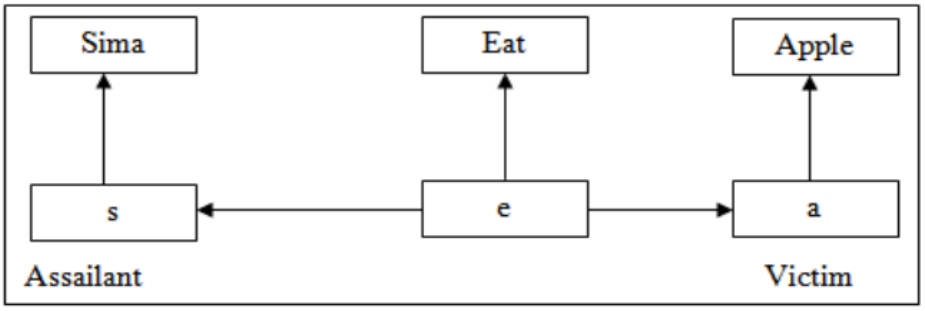
1. Propositions to be made without commitment to truth.

2. Expressions to be quantified.In partitioned semantic network, the network is broken into spaces which consist of groups of nodes and arcs and regard each space as a node.

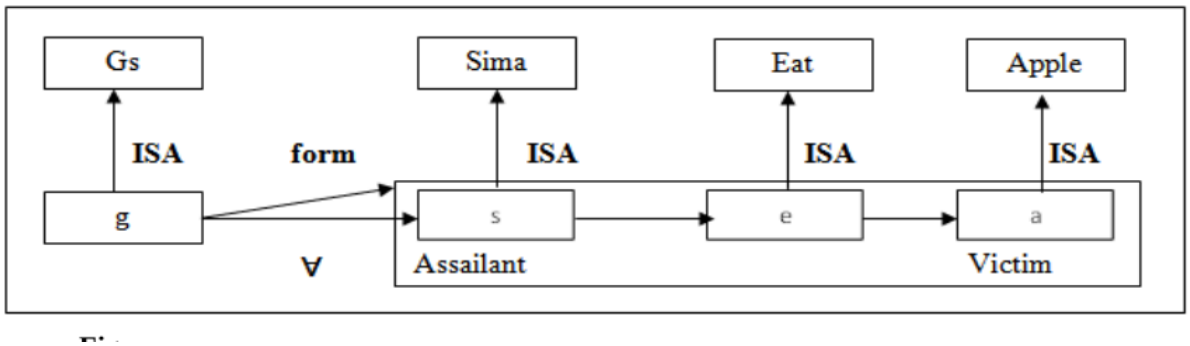
Let us consider few examples.

Draw the partitioned semantic network structure for the followings:

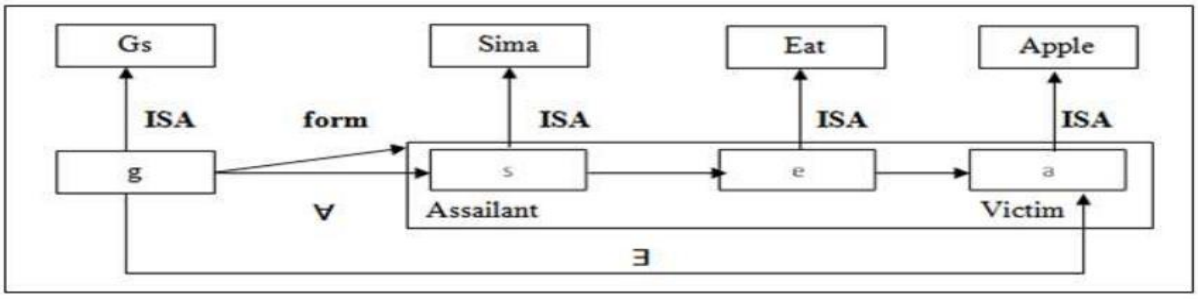
a) Sima is eating an apple.



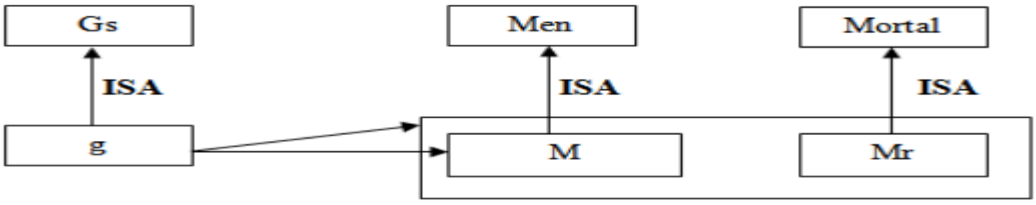
b) All Sima are eating an apple.



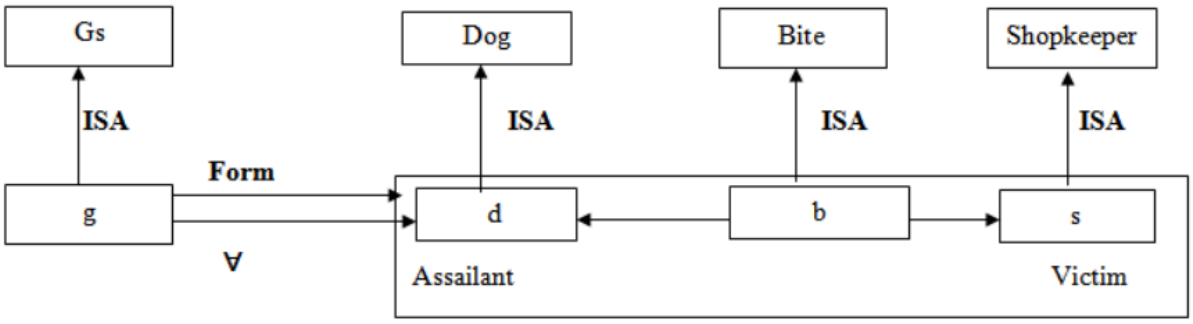
c) All Sima are eating some apple.



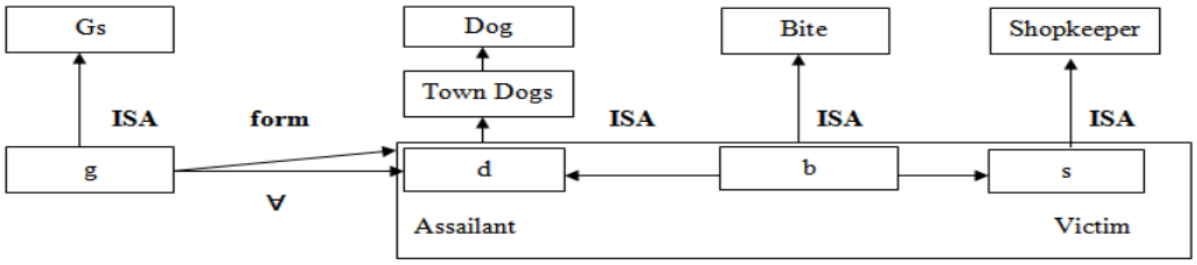
d) All men are mortal



e) Every dog has bitten a shopkeeper



f) Every dog in town has bitten a shopkeeper.



NOTE: On the above semantic network structures, the instance "IS A" is used. Also two terms like

assailant and victim are used. Assailant means "by which the work is done" and that of victim refers to "on which the work is applied". Another term namely GS, which refers to General Statement. For GS, make a node g which is an instance of Gs. Every element will have at least two attributes. Firstly, a form that states which a relation is being asserted. Secondly, one or more for all (∀) or there exists (∃) connections which represent universally quantifiable variables.

FRAME

A frame is a collection of attributes and associated values that describe some entity in the world. Frames are general record like structures which consist of a collection of slots and slot values. The slots may be of any size and type. Slots typically have names and values or subfields called

facets. Facets may also have names and any number of values. A frame may have any number of slots, a slot may have any number of facets, each with any number of values. A slot contains information such as attribute value pairs, default values, condition for filling a slot, pointers to other related frames and procedures that are activated when needed for different purposes. Sometimes a frame describes an entity in some absolute sense, sometimes it represents the entity from a particular point of view. A single frame taken alone is rarely useful. We build frame systems out of collection of frames that are connected to each other by virtue of the fact that the value of an attribute of one frame may be another frame. Each frame should start with an open parenthesis and closed with a closed parenthesis.

Syntax of a frame

```
(<frame name>
(<slot1> (<facet1> <value 1>.....<value n1>)
        (<facet2> <value1>.....<value n2>)
        .
        .
        .
        .
        .
        (<facet n> <value1>..... <value nn>))
(<slot 2> (<facet1> <value 1>.....<value n1>)
        (<facet2><value2>.....<value n2>)
        .
        .
        ))
```

Let us consider the below examples.

1)

Create a frame of the person Ram who is a doctor. He is of 40. His wife name is Sita. They have two children Babu and Gita. They live in 100 kps street in the city of Delhi in India. The zip code is 756005.

```
(Ram
        (PROFESSION (VALUE Doctor))
        (AGE (VALUE 40))
        (WIFE (VALUE Sita))
        (CHILDREN (VALUE Bubu, Gita))
        (ADDRESS
                (STREET (VALUE 100 kps))
                (CITY(VALUE Delhi))
                (COUNTRY(VALUE India))
                (ZIP (VALUE 756005))))
```

2) Create a frame of the person Anand who is a chemistry professor in RD Women's College. His wife name is Sangita having two children Rupa and Shipa.

```
(Anand
        (PROFESSION (VALUE Chemistry Professor))
        (ADDRESS (VALUE RD Women's College))
        (WIFE (VALUE Sangita))
        (CHILDREN(VALUE RupaShipa)))
```

3) Create a frame

of the person Akash who has a white maruti car of LX-400 Model. It has 5 doors. Its weight is 225kg, capacity is 8, and mileage is 15 km /lit.

```
(Akash
    (CAR (VALUE Maruti))
    (COLOUR (VALUE White))
    (MODEL (VALUE LX-400))
    (DOOR (VALUE 5))
    (WEIGHT (VALUE 225kg))
    (CAPACITY (VALUE 8))
    (MILAGE (VALUE 15km/lit)))
```

The frames can be attached with another frame and can create a network of frames. The main task of action frame is to provide the facility for procedural attachment and help in reasoning process. Reasoning using frames is done by instantiation. Instantiation process begins, when the given situation is matched with frames that are already in existence. The reasoning process tries to match the current problem state with the frame slot and assigns them values.The valuesassigned to the slots depict a particular situation and by this, the reasoning process moves towards a goal. The reasoning process can be defined as filling slot values in frames.

Conceptual Graphs

It is a knowledge representation technique which consists of basic concepts and the relationship between them. As the name indicates, it tries to capture the concepts about the events and represents them in the form of a graph. A concept may be individual or generic. An individual concept has a type field followed by a reference field. For example person : Ram. Here person indicates type and Ram indicates reference.An individual concept should be represented within a rectangle in graphical representation and within a square bracket in linear representation. The generic concept should be represented within an oval in graphical representation and within a parenthesis in linear representation. Conceptual graph is a basic building block for associative network. Concepts like AGENT, OBJECT, INSTRUMENT, PART are obtained from a collection of standard concepts. New concepts and relations can be defined from these basic ones. These are also basic building block for associative network. A linear conceptual graph is an elementary form of this structure. A single conceptual graph is roughly equivalent to a graphical diagram of a natural language sentence where the words are depicted as concepts and relationships.

Consider an example

“Ram is eating an apple “

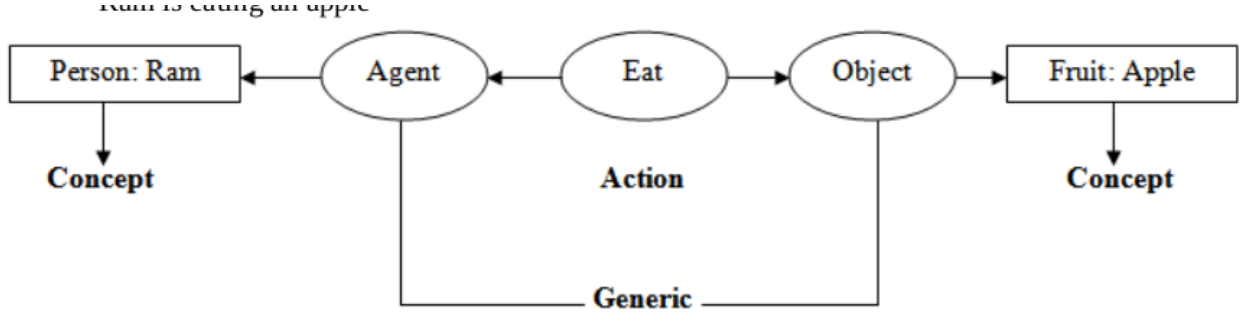
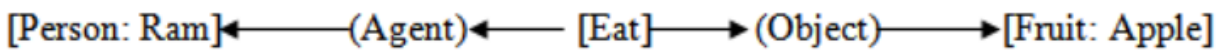


Figure Graphical Representation



Conceptual Dependency

It is an another knowledge representation technique in which we can represent any kind of knowledge. It is based on the use of a limited number of primitive concepts and rules of formation to represent any natural language statement. Conceptual dependency theory is based on the use of knowledge representation methodology was primarily developed to understand and represent natural language structures.

The conceptual dependency structures were originally developed by Roger C SChank in 1977. If a computer program is to be developed that can understand wide phenomenon represented by natural languages, the knowledge representation should be powerful enough to represent these concepts. The conceptual dependency representation captures maximum concepts to provide canonical form of meaning of sentences. Generally there are four primitives from which the conceptual dependency structure can be described. They are

a. ACTS : Actions

b. PPs : Objects (Picture Producers)

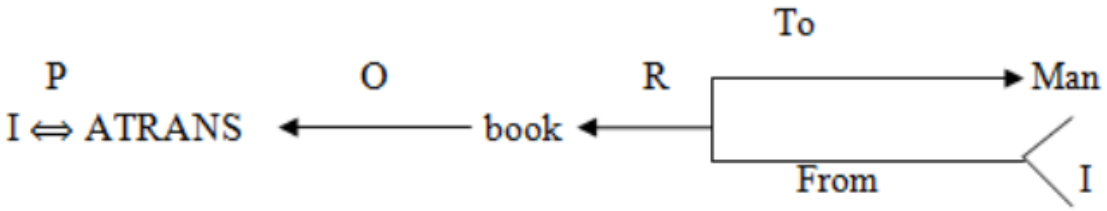
c. AAs : Modifiers of Actions (Action Aiders)

d. PAs : Modifiers of PPs (Picture Aiders)

e. TS : Time of action

Conceptual dependency provides both a structure and a specific set of primitives at a particular level of granularity, out of which representation of particular pieces of information can be constructed.

For example



Where ←: Direction of dependency

Double arrow indicates two way link between actor and action.

P: Past Tense

ATRANS: One of the primitive acts used by the theory

O: The objective case relation

R: Recipient case Relation

In CD, representation of actions are built from a set of primitive acts.

1) **ATRANS**: Transfer of an abstract relationship (give, accept, take)

2) **PTRANS:** Transfer the physical location of an object (Go, Come, Run, Walk)

3) **MTRANS**: Transfer the mental information (Tell)

4) **PROPEL**: Application of physical force to an object (push, pull, throw)

5) **MOVE**: Movement of a body part by its owner (kick).

6) **GRASP:** Grasping of an object by an action (clutch)

7) **INGEST**: Ingestion of an object by an animal (eat)

8) **EXPEL**: Expel from an animal body (cry)

9) **MBUILD**: Building new information out of old (decide)

10) **SPEAK**: Production of sounds (say)

11) **ATTEND**: Focusing of a sense organ towards a stimulus (Listen)

The main goal of CD representation is to capture the implicit concept of a sentence and make it explicit. In normal representation of the concepts, besides actor and object, other concepts of time, location, source and destination are also mentioned. Following conceptual tenses are used in CD representation.

1) O : Object case relationship

2) R : Recipient case relationship

3) P : Past

4) F : Future

5) Nil : Present

6) T : Transition

7) Ts : Start Transition

8) Tf : Finisher Transition

9) K : Continuing

10) ? : Interrogative

11) / : Negative

12) C : Conditional

Also there are several rules in conceptual dependency

Rule 1: PP $\Leftrightarrow$ ACT

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

E.g. Ram ran

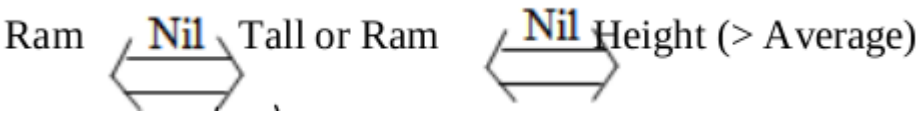
Ram $\overset{P}{\Leftrightarrow}$ PTRANS

Where P: Past Tense

Rule 2: PP $\Leftrightarrow$ PA

It describes the relationship between a PP and PA where the PA indicates one characteristics of PP.

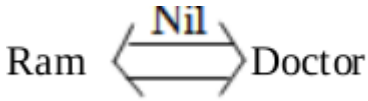
E.g. Ram is tall



Rule 3: PP ⟺ PP

It describes the relationship between two PPs where one PP is defined by other.

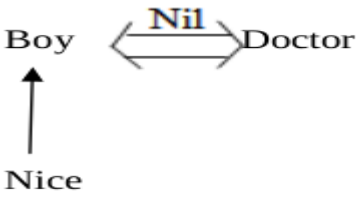
E.g. Ram is a doctor



Rule 4: PP ↑ PA or PA ↓ PP

It describes the relationship between the PP and PA, where PA indicates one attributes of PP.

E.g. A nice boy is a doctor



Rule 5: PP

It describes the relationship between 3 PP's where one PP is the owner of another PP.

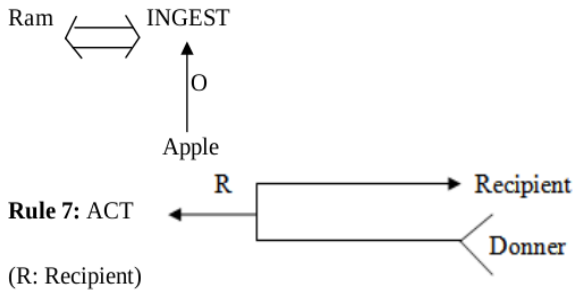
E.g. Ram's Cat

Ram

Rule 6: Act $\xleftarrow{O}$ PP Where O: Object

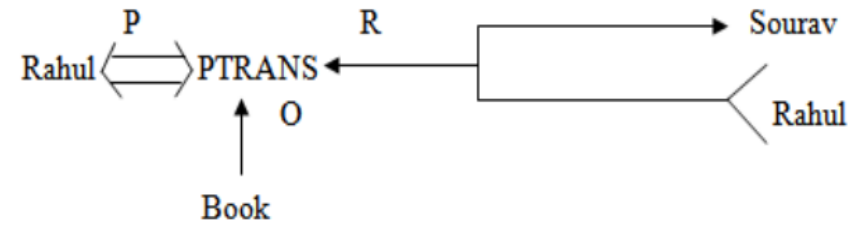
It describes the relationship between the PP and ACT. Where PP indicates the object of that action.

E.g. Ram is eating an apple.

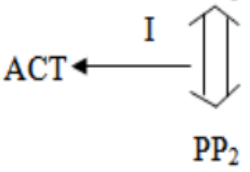


Here one PP describes the recipient and another PP describes the donner

E.g. Rahul gave a book to sourav.



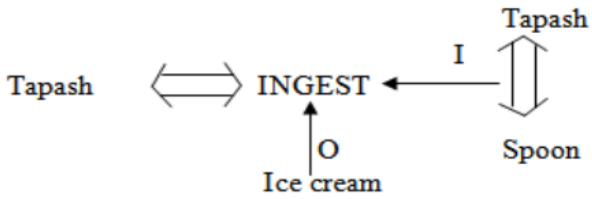
Rule 8:



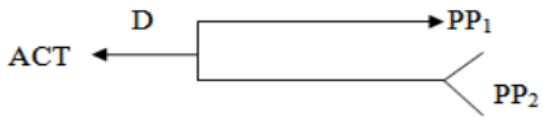
(I: Instrument used in the action)

Here PP 1 indicates the agent and PP 2 indicates the object that is used in the action.

E.g. Tapash ate the ice cream with the spoon.

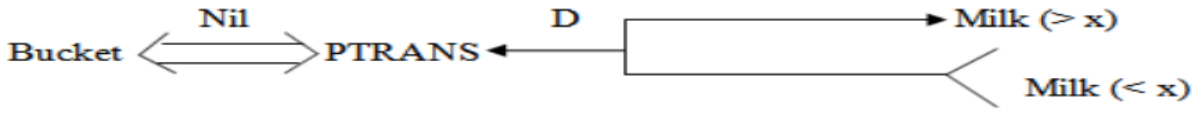


Rule 9:



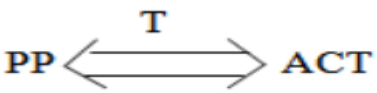
Here D indicates destination, PP 1 indicates destination and PP 2 indicates the source.

E.g. the bucket is filled with milk.



X indicates the average milk and the source i.e. bucket is dry which is hidden.

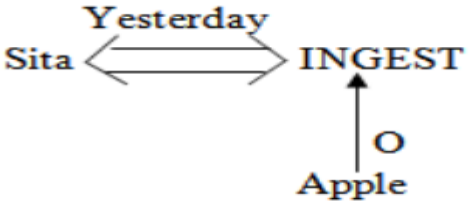
Rule 10:



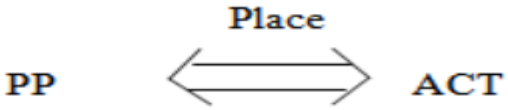
(T: Time)

It describes the relationship between a conceptualization and the time at which the event is described occurs.

E.g. Sita ate the apple yesterday.

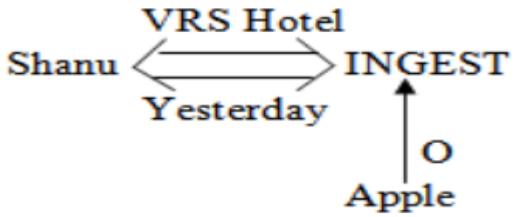


Rule 11:

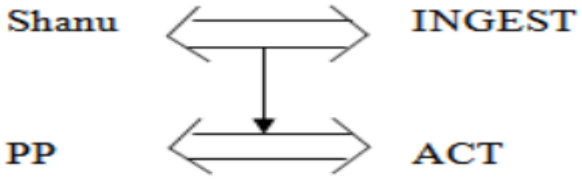


It describes the relationship between a conceptualization and the place at which it is occurred.

E.g. Shanu ate the apple at VRS hotel yesterday

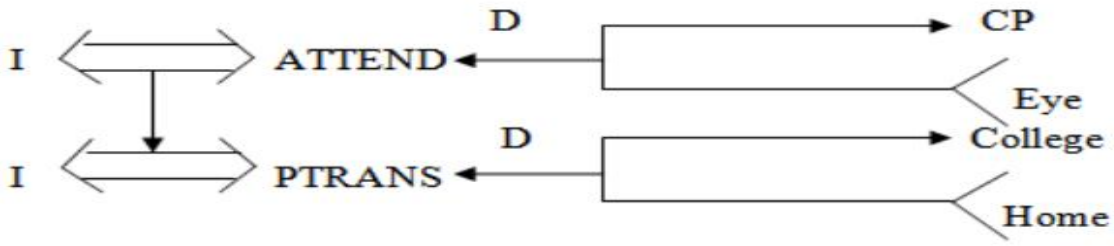


Rule 12:



It describes the relationship between one conceptualization with another.

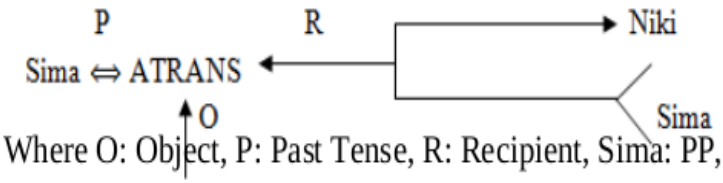
E.g. while I was going to college, I saw a snake



(Where CP: Conscious Processor i.e. the combination of all sense organs like eye, ear, nose etc.)

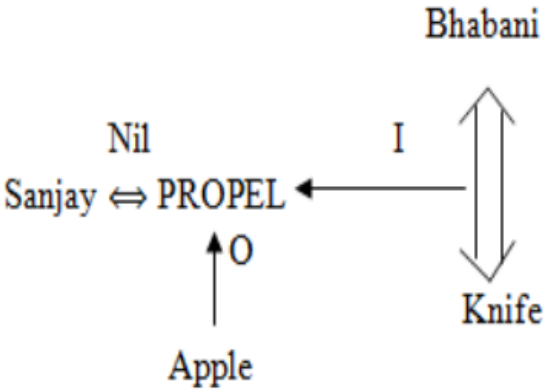
By using the above rules we can represent any sentence. Let us visualize few examples on conceptual dependency.

1) Sima gave a book to Niki

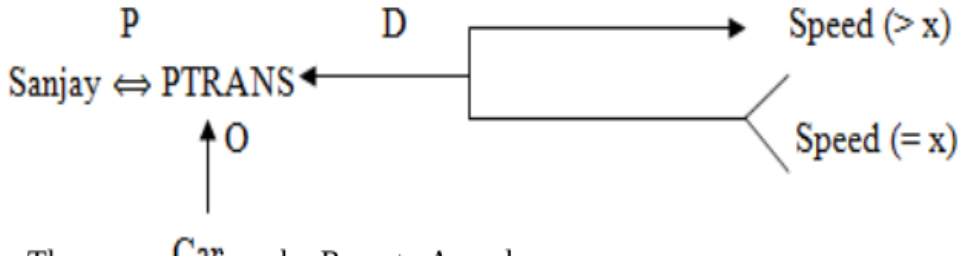


Where O: Object, P: Past Tense, R: Recipient, Sima: PP, Book: PP, Niki: PP, ATRANS: give

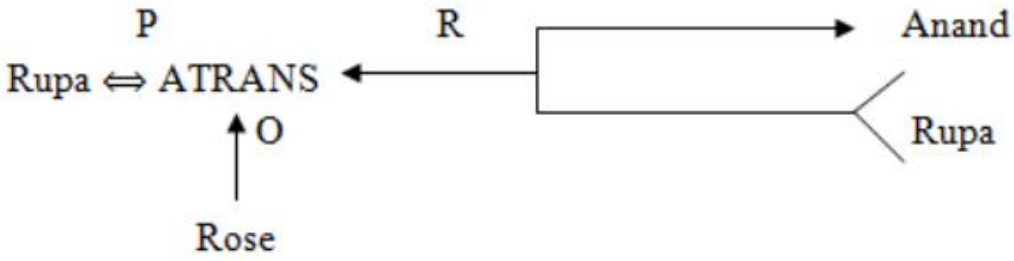
2) Bhabani cuts an apple with a knife



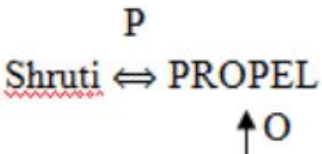
3) Sanjay drove the car fast



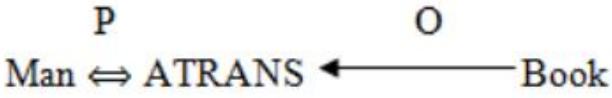
4) The rose was given by Rupa to Anand



5) Shruti pushed the door.

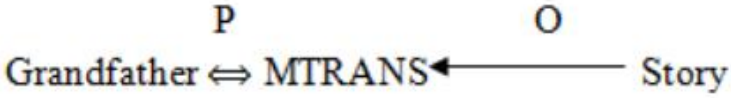


6) The man took a book

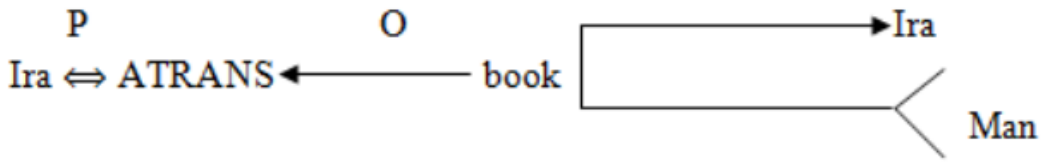


Here man is the doctor and book is the object of the action took.

7) My grandfather told me a story



8) Ira gave the man a dictionary



Advantages of CD:

- Using these primitives involves fewer inference rules.
- Many inference rules are already represented in CD structure.
- The holes in the initial structure help to focus on the points still to be established.

Disadvantages of CD:

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

SCRIPT

It is an another knowledge representation technique. Scripts are frame like structures used to represent commonly occurring experiences such as going to restaurant, visiting a doctor. A script is a structure that describes a stereotyped sequence of events in a particular context. A script consist of a set of slots. Associated with each slot may be some information about what kinds of values it may contain as well as a default value to be used if no other information is available. Scripts are useful because in the real world, there are no patterns to the occurrence of events. These patterns arise because of clausal relationships between events. The events described in a script form a giant casual chain. The beginning of the chain is the set of entry conditions which enable the first events of the script to occur. The end of the chain is the set of results which may enable later events to occur. The headers of a script can all serve as indicators that the script should be activated.

Once a script has been activated, there are a variety of ways in which it can be useful in interpreting a particular situation. A script has the ability to predict events that has not explicitly been observed. An important use of scripts is to provide a way of building a single coherent interpretation from a collection of observation. Scripts are less general structures than are frames

and so are not suitable for representing all kinds of knowledge. Scripts are very useful for representing the specific kinds of knowledge for which they were designed.

A script has various components like:

1) Entry condition: It must be true before the events described in the script can occur. E.g. in a

restaurant script the entry condition must be the customer should be hungry and the customer has

money.

2) Tracks: It specifies particular position of the script e.g. In a supermarket script the tracks may be

cloth gallery, cosmetics gallery etc.

3) Result: It must be satisfied or true after the events described in the script have occurred.

e.g. In a restaurant script the result must be true if the customer is pleased.

The customer has less money.

4) Probs: It describes the inactive or dead participants in the script e.g. In a supermarket script, the

probes may be clothes, sticks, doors, tables, bills etc.

5) Roles: It specifies the various stages of the script. E.g. In a restaurant script the scenes may be

entering, ordering etc.

Now let us look on a movie script description according to the above component.

a) Script name : Movie

b) Track : CINEMA HALL

c) Roles : Customer(c), Ticket seller(TS), Ticket Checker(TC), Snacks Sellers (SS)

d) Probes : Ticket, snacks, chair, money, Ticket, chart

e) Entry condition : The customer has money

The customer has interest to watch movie.

6) Scenes:

a. SCENE-1 (Entering into the cinema hall)

C PTRANS C into the cinema hall

C ATTEND eyes towards the ticket counter

C PTRANS C towards the ticket counters

C ATTEND eyes to the ticket chart

C MBUILD to take which class ticket

C MTRANS TS for ticket

C ATRANS money to TS

TS ATRANS ticket to C

b. SCENE-2 (Entering into the main ticket check gate)

C PTRANS C into the queue of the gate

C ATRANS ticket to TC

TC ATTEND eyes onto the ticket

TC MBUILD to give permission to C for entering into the hall

TC ATRANS ticket to C

C PTRANS C into the picture hall.

c. SCENE-3 (Entering into the picture hall)

C ATTEND eyes into the chair

TC SPEAK where to sit

C PTRANS C towards the sitting position

C ATTEND eyes onto the screen

d. SCENE-4 (Ordering snacks)

C MTRANS SS for snacks

SS ATRANS snacks to C

C ATRANS money to SS

C INGEST snacks

e. SCENE-5 (Exit)

C ATTEND eyes onto the screen till the end of picture

C MBUILD when to go out of the hall

C PTRANS C out of the hall

7) Result:

The customer is happy

The customer has less money

Example 2: Write a script of visiting a doctor in a hospital

1) SCRIPT_NAME : Visiting a doctor

2) TRACKS : Ent specialist

3) ROLES : Attendant (A), Nurse(N), Chemist (C),Gatekeeper(G), Counter clerk(CC), Receptionist(R), Patient(P), Ent specialist Doctor (D), Medicine Seller (M).

4) PROBES: Money, Prescription, Medicine, Sitting chair, Doctor's table, Thermometer, Stetho scope, writing pad, pen, torch, stature.

5) ENTRY CONDITION: The patient need consultation.

Doctor's visiting time on.

6) SCENES:

a. SCENE-1 (Entering into the hospital)

P PTRANS P into hospital

P ATTEND eyes towards ENT department

P PTRANS P into ENT department

P PTRANS P towards the sitting chair

b. SCENE-2 (Entering into the Doctor's Room)

P PTRANS P into doctor's room

P MTRANS P about the diseases

P SPEAK D about the disease

D MTRANS P for blood test, urine test

D ATRANS prescription to P

P PTRANS prescription to P.

P PTRANS P for blood and urine test

c. SCENE-3 (Entering into the Test Lab)

P PTRANS P into the test room

P ATRANS blood sample at collection room

P ATRANS urine sample at collection room

P ATRANS the examination reports

d. SCENE-4 (Entering to the Doctor's room with Test reports)

P ATRANS the report to D

D ATTEND eyes into the report

D MBUILD to give the medicines

D SPEAK details about the medicine to PP ATRANS doctor's fee

P PTRANS from doctor's room

e. SCENE-5 (Entering towards medicine shop)

P PTRANS P towards medicine counter

P ATRANS Prescription to M

M ATTEND eyes into the prescription

M MBUILD which medicine to give

M ATRANS medicines to P

P ATRANS money to M

P PTRANS P from the medicine shop

7) RESULT:

The patient has less money

Patient has prescription and medicine.

- Scripts are useful in describing certain situations such as robbing a bank. This might involve:

    - Getting a gun.
    - Hold up a bank.
    - Escape with the money.

    Here the *Props* might be

    - Gun, *G*.
    - Loot, *L*.
    - Bag, *B*
    - Get away car, *C*.

The *Roles* might be:

- Robber, *S*.
- Cashier, *M*.
- Bank Manager, *O*.
- Policeman, *P*.

The *Entry Conditions* might be:

- *S* is poor.
- *S* is destitute.

The *Results* might be:

- *S* has more money.
- *O* is angry.

- *M* is in a state of shock.
- *P* is shot.

There are 3 scenes: obtaining the gun, robbing the bank and the getaway. The full Script could be described as follows:

Script: ROBBERY	*Track: Successful Snatch*
Props: G = Gun, L = Loot, B= Bag, C = Get away car.	*Roles*: R = Robber, M = Cashier, O = Bank Manager, P = Policeman.
Entry Conditions: R is poor. R is destitute.	*Results*: R has more money. O is angry. M is in a state of shock. P is shot.
Scene 1: Getting a gun R PTRANS R into Gun Shop R MBUILD R choice of G R MTRANS choice. R ATRANS buys G (go to scene 2)	
Scene 2 Holding up the bank R PTRANS R into bank R ATTEND eyes M, O and P R MOVE R to M position R GRASP G R MOVE G to point to M R MTRANS "Give me the money or ELSE" to M P MTRANS "Hold it Hands Up" to R R PROPEL shoots G P INGEST bullet from G M ATRANS L to M M ATRANS L puts in bag, B M PTRANS exit O ATRANS raises the alarm (go to scene 3)	
Scene 3: The getaway M PTRANS C	

Advantages of Scripts:

- Ability to predict events.
- A single coherent interpretation may be build up from a collection of observations.

Disadvantages:

- Less general than frames.
- May not be suitable to represent all kinds of knowledge.

Agent based problem solving:

- It is a kind of goal-based agent where the agent works along in reaching the goal state.
- It uses atomic representations, i.e, the states of the world to search along and reach the goal.
- The functionalities of the problem solving agent includes:
    (a) Goal formulation: This is the first step in problem solving where the agent gathers the information about the current state and goal state of the system.
    (b) Problem formulation: This step includes the actions and changes happening to the states of the system. There are five stages of actions that takes place in this stage and they are:
        (i) Initial state: The initial state of the system is analyzed.
        (ii) Actions: The alternative actions to reach the goal is analyzed.
        (iii) Transition model: The path to reach the next node is finalized.
        (iv) Goal test: At each node a test is conducted to check whether it is the goal node or not.
        (v) Path cost: The cost of reaching the current state from the previous state is calculated.
    (c) Search: The sequence of actions to perform the search is analyzed.
    (d) Solution: The search algorithm for performing the search is analyzed.
    (e) Execution: The search is executed

Machine Learning:

- Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed.
- Machine learning focuses on the development of computer programs that can access data and use it learn for themselves.
- The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide.
- The primary aim is to allow the computers learn automatically without human intervention or assistance and adjust actions accordingly.
- Some machine learning methods

Machine learning algorithms are often categorized as supervised or unsupervised.

(a) **Supervised machine learning algorithms** can apply what has been learned in the past to new data using labeled examples to predict future events. Starting from the analysis of a known training dataset, the learning algorithm produces an inferred function to make predictions about the output values. The system is able to provide targets for any new input after sufficient training. The learning algorithm can also compare its output with the correct, intended output and find errors in order to modify the model accordingly.

(b) **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.

(c) **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.

(d) **Reinforcement machine learning algorithms** is a learning method that interacts with its environment by producing actions and discovers errors or rewards. Trial and error search and delayed reward are the most relevant characteristics of reinforcement learning. This method allows machines and software agents to automatically determine the ideal behavior within a specific context in order to maximize its performance. Simple reward feedback is required for the agent to learn which action is best; this is known as the reinforcement signal.

- Machine learning enables analysis of massive quantities of data. While it generally delivers faster, more accurate results in order to identify profitable opportunities or dangerous risks, it may also require additional time and resources to train it properly.
- Combining machine learning with AI and cognitive technologies can make it even more effective in processing large volumes of information.

MODULE-4

Game playing

In many environments, there are multiple agents. In this, a given agent must consider the actions of other agents. If the agents are competing with each other, then such an environment is called a competitive environment. Competitive environments in which the agents goals are in conflict, results in problems known as games.

Consider the game of chess. If one player wins the game of chess, then utility function value is +1. in this the opposite player loses. His utility function value is -1. if the game ends in a draw, the utility function value is 0. Like this, in AI, games are turn taking, 2 player, zero sum games.

For AI researchers, the nature of games makes them an attractive subject for study. The state of a game is easy to represent and agents are limited to a small number of actions. Precise rules are there for making these actions.

By 1950, chess playing program was developed by Zuse, Shannon, Wiener and Turing. After that systems for playing checkers, Othello, Backgammon and Go were developed. Games are interesting because they are too hard to solve. For example, chess has around 35 100 states. Generating all these states is impossible. Therefore game playing research has brought a number of interesting ideas on how to make the best possible use of time.

Formulating Game Playing as Search

- Consider 2-person, zero-sum, perfect information (i.e., both players have access to complete information about the state of the game, so no information is hidden from either player) games. Players alternate moves and there is no chance (e.g., using dice) involved
- Examples: Tic-Tac-Toe, Checkers, Chess, Go, Nim, and Othello
- **Iterative** methods apply here because search space is too large for interesting games to search for a "solution." Therefore, search will be done before EACH move in order to select the best next move to be made.
- **Adversary** methods needed because alternate moves are made by an opponent who is trying to win. Therefore must incorporate the idea that an adversary makes moves that are "not controllable" by you.
- **Evaluation function** is used to evaluate the "goodness" of a configuration of the game. Unlike in heuristic search where the evaluation function was a non-negative estimate of the cost from the start node to a goal and passing through the given node, here the evaluation function, also called the *static evaluation function* estimates board quality in leading to a win for one player.
- Instead of modeling the two players separately, the zero-sum assumption and the fact that we don't have, in general, any information about how our opponent plays, means we'll use a single evaluation function to describe the goodness of a board with respect to BOTH players. That is, *f(n)* = large positive value means the board associated with node *n* is good for me and bad for you. *f(n)* = large negative value means the board is bad for me

and good for you. *f(n)* near 0 means the board is a neutral position. *f(n)* = +infinity means a winning position for me. *f(n)* = -infinity means a winning position for you.

- Example of an Evaluation Function for Tic-Tac-Toe: *f(n)* = [number of 3-lengths open for me] - [number of 3-lengths open for you] where a 3-length is a complete row, column, or diagonal.
- Most evaluation functions are specified as a weighted sum of "features:" (w1 * feat1) + (w2 * feat2) + ... + (wn * featn). For example, in chess some features evaluate piece placement on the board and other features describe configurations of several pieces. Deep Blue has about 6000 features in its evaluation function.

Design of good heuristic

In state space search, heuristics are formalized as rules for choosing those branches in a state space that are most likely to lead to an acceptable problem solution.

AI problem solvers employ heuristics in two basic situations:

1. A problem may not have an exact solution because of inherent ambiguities in theproblem statement or available data. Medical diagnosis is an example of this. A given set of symptoms may have several possible causes; doctors use heuristics to choose the most likely diagnosis and formulate a plan of treatment. Vision is another example of an inexact problem. Visual scenes are often ambiguous, allowing multiple interpretations of the connectedness, extent, and orientation of objects. Optical

illusions exemplify these ambiguities. Vision systems often use heuristics to select the most likely of several possible interpretations of as scene.

2. A problem may have an exact solution, but the computational cost of finding it may be prohibitive. In many problems (such as chess), state space growth is combinatorially explosive, with the number of possible states increasing exponentially or factorially with the depth of the search. In these cases,

exhaustive, brute-force search techniques such as depth-first or breadth-first search may fail to find a solution within any practical length of time. Heuristics attack this complexity by guiding the search along the most "promising" path through the space. By eliminating unpromising states and their descendants from consideration, a heuristic algorithm can (its designer hopes) defeat this combinatorial explosion and find an acceptable solution.

Minimax search procedure

We will consider games with 2 players. The opponents in a game are referred to as MIN and MAX.

MAX represents the player trying to win or to MAXimize his advantage. MIN is the opponent who attempts to MINimize MAX's score. MAX moves first, and then they take turns moving until the game is over. At the end of the game, points are awarded to the winning player and penalties are given to the loser.

A game can be formally defined as a kind of search problem with the following components.

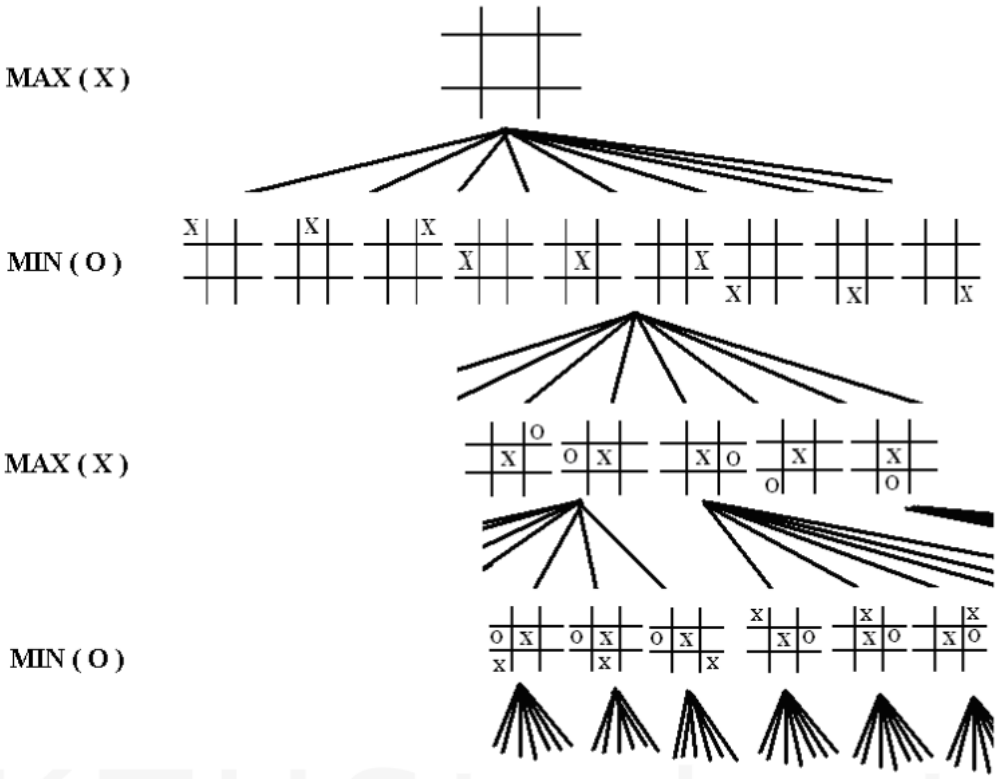
The initial state, which identifies the board position and identifies the player to move.

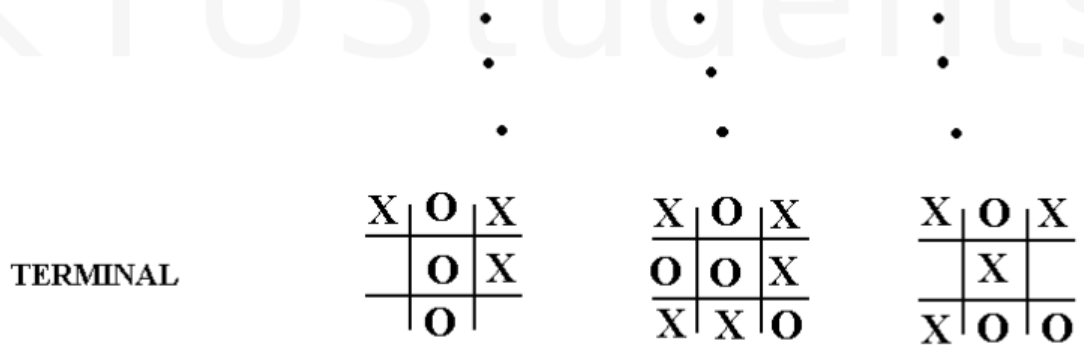
A successor function, which returns a list of (move, state) pairs, each indicating a legal move and the resulting state.

A terminal test, which determines when the game is over. States where the game has ended are called terminal states.

A utility function (also called an objective function) which gives a numeric value for the terminal states. In chess, the outcome is a win, loss or draw with values +1, -1 or 0. some games have a wider variety of possible outcomes.

The initial state and the legal moves for each side define the game tree for the game. Figure below shows part of the game tree for tic-tac-toe.


MAX (X)
MIN (O)
MAX (X)
MIN (O)

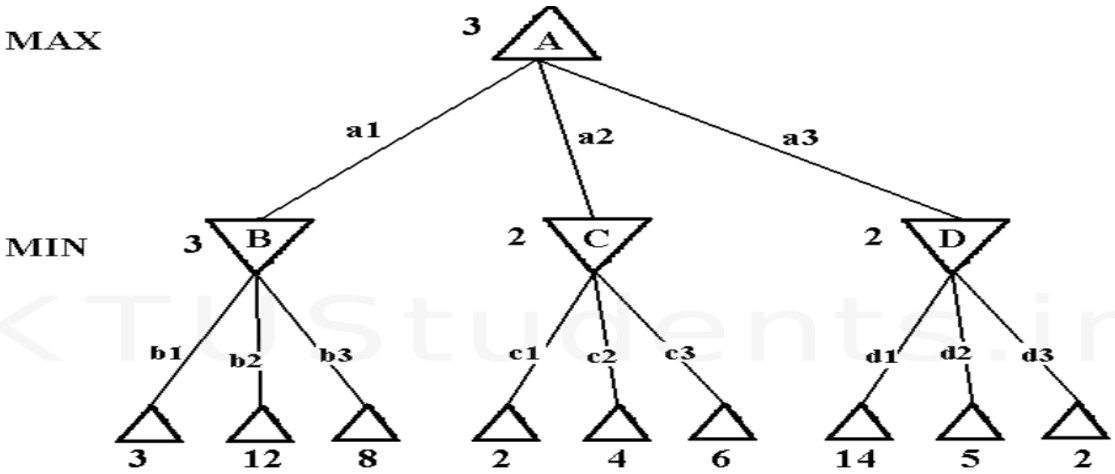


The top node is the initial state, and MAX moves first, placing an X in an empty square. The figure shows part of the search tree giving alternating moves by MIN (O) and MAX (X), until we reach terminal states, which can be assigned utilities according to the rules of the game. Play alternates between MAX's placing an X and MIN's placing an O until we reach leaf nodes corresponding to terminal states such that one player has there in a row or all the aquares are filled. The number on each leaf node indicates the utility value of the terminal state from the point of view of MAX. High values are assumed to be good for MAX and bad for MIN. it is MAX's job to use the search tree to determine the best move.

Search strategies

In a normal search problem, the best solution is a sequence of moves leading to a goal state. In a game, on the other hand, MIN has a role. MAX therefore must find a contingent strategy. We will see how to find this optimal strategy.

Even a simple game like tic-tac-toe is too complex for us to draw the entire game tree. So we will use a game tree as shown below.



The Δ nodes are MAX nodes, in which it is MAX's turn to move and the

nodes are MIN nodes. The terminal states show the utility values for MAX.

The possible moves for MAX at the root node are labeled a1, a2, a3. the possible replies to a1 for MIN are b1, b2, b3 and so on. This game ends after one move each by MAX and MIN. Given a game tree, the optimal strategy can be determined by examining the minimax value of each node, which we write as minimax-value (n). The minimax-value of a node is the utility for MAX of being in the corresponding state. The minimax-value of a terminal state is just its utility. Given a choice, MAX will prefer to move to a state of maximum value, whereas MIN prefers a state of minimum value. So we have

$$\text{Minimax-value}(n) = \begin{cases} \text{Utility}(n), & \text{if } n \text{ is a terminal state} \\ \max_{s \in \text{successors}(n)} \text{minimax-value}(s), & \text{If } n \text{ is a MAX node} \\ \min_{s \in \text{successors}(n)} \text{minimax-value}(s), & \text{If } n \text{ is a MIN node} \end{cases}$$

Let us apply these definitions to the above game tree. The terminal nodes on the bottom level are labeled with their utility values. The first MIN node, labeled B, has 3 successors wit hvalues3, 12 and 8. so its minimax-value is 3. similarly c has a minimax-value 2 and D has minimax-value 2. the root node is a MAX node; its successors have minimax values 3, 2 and 2. So it has a minimax value of 3.

That is Minimax-cvalue(A) = max [min (3,12,8), min (2,4,6), min (14,5,2)]

= max [3, 2, 2]

= 3

As a result, action a1 is the optimal choice for MAX because it leads to the successor with the highest minimax-value. This is the minimax decision at the root. The definition of optimal play for MAX assumes that MIN also plays optimally – it maximizes the worst outcome for MAX. suppose MIN does not play optimally. Then it is easy to show that MAX will do even better.

The minimax algorithm

The minimax algorithm given below computes the minimax decision from the

current state.

function minimax-decision (state)

returns an action

```
        {
                v = max-value (state);
                return an action in successors (state) with value v;
        }
function max-value (state)
        returns a utility value
        {
                if terminal-test (state) then
                return utility (state);
                v = - α ;
                for a,s in successors (state)
        {
                v = max (v, min-value (s);
        }
        return v;}

function min-value (state)
        returns a utility value
        {
                If terminal-test (state) then
                return utility (state);
```

```
        v = + α ;
        for a,s in successors (state)
        {v = min (v, max-value (s);}
        return v;
}
```

The above procedure uses a simple recursive computation of the minimax values of each successor state. The recursion proceed s all the way down to the leaves of the tree, and then the minimax values are backed up through the tree.

For example, for the above game tree, the algorithm first recourses down to the three bottom left nodes, and uses the utility function on them to discover that their values are 3, 12 and 8 respectively. Then it takes the minimum of these values, 3, and returns it as the backed up value of node B. a similar process gives the backed up values of 2 for C and 2 for D. finally, we take the maximum of 3, 2 and 2 to get the backed up

value of 3 for the root node.

Minimaxing to fixed ply depth

In applying minimax to more complicated games it is seldom possible to expand the state space graph out to the leaf nodes . Instead the state space is searched to a predefines number of levels , as determined by available resources of time and memory . This strategy is called an **n-ply look – ahead**, where n is the number of levels explored. As the leaves of this subgraph are not final states of the game , it is not possible to give them values that reflect a win or a loss .Instead , each node is given a value according to some heuristic evaluation function. The value that is propagated back to the root node is not an indication of whether or not a win can be achieved but is simply the heuristic value of the best state that can be reached in n moves from the root. Look ahead increases the power of a heuristic by allowing it to be applied over a greater area of the state space. Minimax consolidates these separate evaluations for the ancestor state.

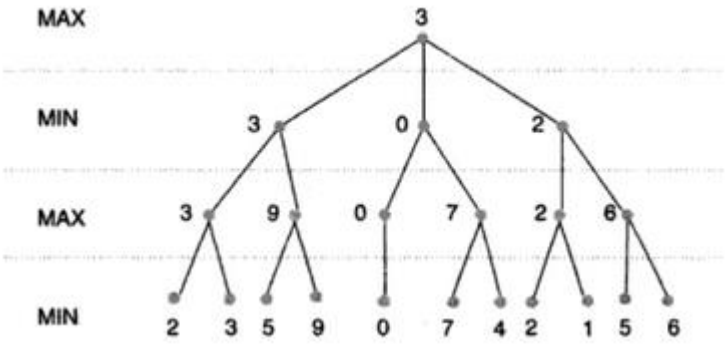
In a game of conflict , each player attempts to overcome the other, so many game heuristics directly measure the advantage of one player over another, In checkers or chess , piece advantage is important , so a simple heuristic might take the difference in the number of pieces belonging to max and min and try to maximize the difference between these piece measure. A more sophisticated strategy might assign different values to the pieces depending on their value or location on the board.

Game graphs are searched by level,or ply.MIN and MAX alternatively select moves.Each move by a player defines a new ply of the graph. Game playing programs typically look ahead a fixed ply depth, often determined by the space or limitations of the computer. The states on that ply are measured heuristically and the values are propagated back up the graph using minimax. The search algorithm then uses

these derived values to select among possible next moves.

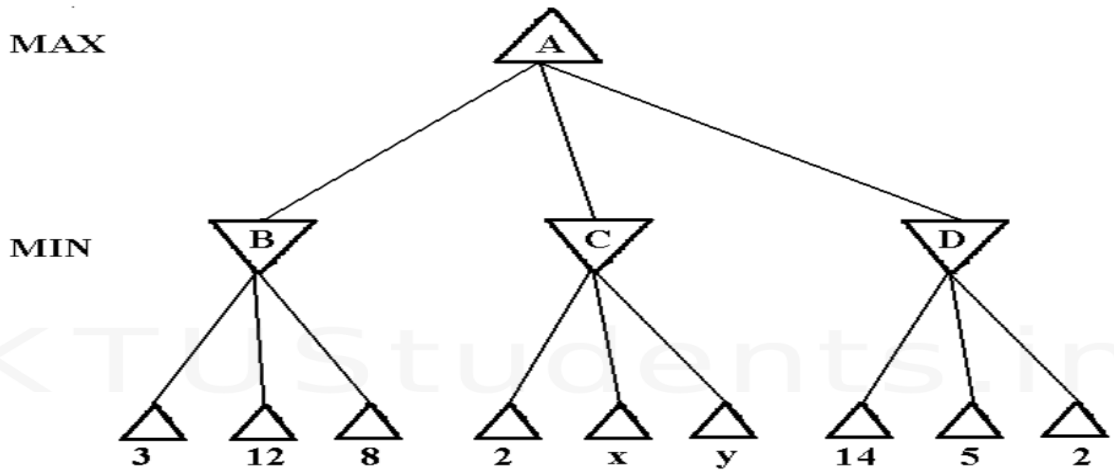
After assigning an evaluation to each state on the selected ply, the program propagates a value up to each parent state.If the parent is on a MIN level, the minimum value of the children is backed up . If the parent is a MAX node,minimax assigns it the maximum value of its children.

Maximising for MAX parents and minimising for MIN , the values go back up the graph to the children of the current state. Below figure shows minimax on a hypothetical state space with a four-ply look-ahead.



Alpha- Beta pruning

The problem with minimax search is that a large number of states need to be examined. We can effectively cut it in half using a technique called alpha beta pruning. Here the trick is that it is possible to compute the correct minimax decision without looking at every node in the game tree.



Consider again the game tree shown below.

Let us calculate the minimax value at a. one way is to simplify the formula for minimax value. Let 2 successors of node C have values x and y. the value of the root node is given by

Minimax-value (A) = max [min (3, 12, 8), min (2, x, y), min (14, 5, 2)]

=max [3, min (2, x, y), 2)]

Suppose minimum of x and y is z.

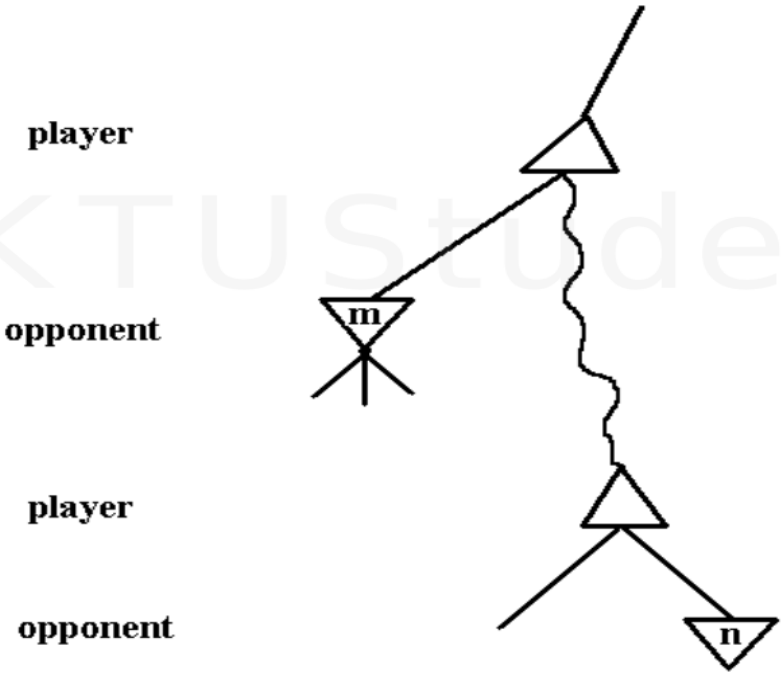
Then minimax-value (A) = max [3, min (2,z), 2]

If z <=2.

= max [3, z, 2]

= 3

From this, it is clear that the value of the root node A and hence the minimax decision are independent of the values of the pruned leaves x and y. 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.



The general principle is this. Consider a node n somewhere in the tree; such that player has a choice of moving to that node. If player has a better choice m either at the parent node of n or at any choice further up, then n will never be reached in actual play.

Alpha beta pruning gets its name from the following 2 parameters, α and β.

The algorithm for alpha beta search is given below.

function alpha-beta-search (state)

returns an action

```
{
v = max-value (state, -α, +α );
return the action in successors (state) with value v;
}
function max-value (state, α, β )
returns a utility value
{
if terminal-test (state) then
return utility (state);
v = -α ;
for a,s in successors (state)
{
v = max (v, min-value (s, α, β) );
if v >= β then
return v;
α = max (α, v);
}
return v;
}
function min-value (state, α, β )
returns a utility value
```

{

if terminal-test (state) then

return utility (state);

v = +α ;

for a, s in successors (state)

{

v = min (v, max-value (s, α, β));

if v <= α then

return v;

β = min (β, v);

}

return v;

}

α – the value of the best (ie. (ie. Highest value) choice we have found so far at any choice point along the path for MAX.

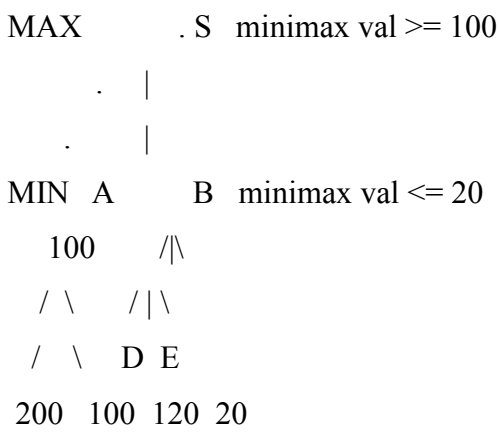
β – the value of the best (ie. Lowest value) choice we have found so far at any choice point along the path for MIN. alpha-beta search updates the values of α and β as it goes along and prunes the remaining branches at a node as soon as the value of the current node is known to be worse than the current α or β value for MAX or MIN, respectively.

Alpha-Beta Pruning

- Minimax computes the optimal playing strategy but does so inefficiently because it first generates a complete tree and then computes and backs up static-evaluation-function

values. For example, from an average chess position there are 38 possible moves. So, looking ahead 12 plies involves generating $1 + 38 + 38^2 + ... + 38^{12} = (38^{12}-1)/(38-1)$ nodes, and applying the static evaluation function at 38^{12} = 9 billion billion positions, which is far beyond the capabilities of any computer in the foreseeable future. Can we devise another algorithm that is guaranteed to produce the same result (i.e., minimax value at the root) but does less work (i.e., generates fewer nodes)? Yes---Alpha-Beta.

- Basic idea: "If you have an idea that is surely bad, don't take the time to see how truly awful it is." -- Pat Winston
- Example of how to use this idea for pruning away useless work:

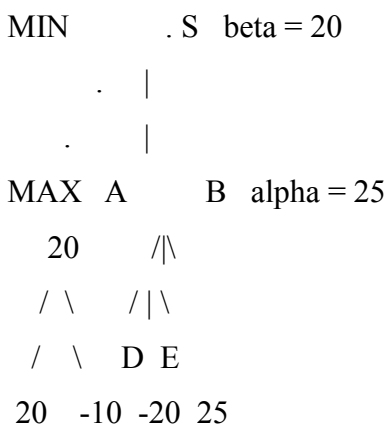


In the above example we are performing a depth-first search to depth (ply) 2, where children are generated and visited left-to-right. At this stage of the search we have just finished generating B's second child, E, and computed the static evaluation function at E (=20). Before generating B's third child notice the current situation: S is a MAX node and its left child A has a minimax value of 100, so S's minimax value **must** eventually be some number >= 100. Similarly, B has generated two children, D and E, with values 120 and 20, respectively, so B's final minimax value must be <= min(120, 20) = 20 since B is a MIN node.

The fact that S's minimax value must be at least 100 while B's minimax value must be no greater than 20 means that no matter what value is computed for B's third child, S's minimax value will be 100. In other words, S's minimax value does not depend on knowing the value of B's third child. Hence, we can cutoff the search below B, ignoring generating any other children after D and E.

Alpha-Beta Algorithm

- Traverse the search tree in depth-first order
- Assuming we stop the search at ply *d*, then at each of these nodes we generate, we apply the static evaluation function and return this value to the node's parent
- At each non-leaf node, store a value indicating the best backed-up value found so far. At MAX nodes we'll call this **alpha**, and at MIN nodes we'll call the value **beta**. In other words, alpha = best (i.e., maximum) value found so far at a MAX node (based on its descendant's values). Beta = best (i.e., minimum) value found so far at a MIN node (based on its descendant's values).
- The alpha value (of a MAX node) is *monotonically non-decreasing*
- The beta value (of a MIN node) is *monotonically non-increasing*
- Given a node *n*, cutoff the search below *n* (i.e., don't generate any more of *n*'s children) if
    - *n* is a MAX node and alpha(n) >= beta(i) for some MIN node ancestor *i* of *n*. This is called a **beta cutoff**.
    - *n* is a MIN node and beta(n) <= alpha(i) for some MAX node ancestor *i* of *n*. This is called an **alpha cutoff**.
- In the example shown above an alpha cutoff occurs at node B because beta(B) = 20 < alpha(S) = 100
- An example of a beta cutoff at node B (because alpha(B) = 25 > beta(S) = 20) is shown below:



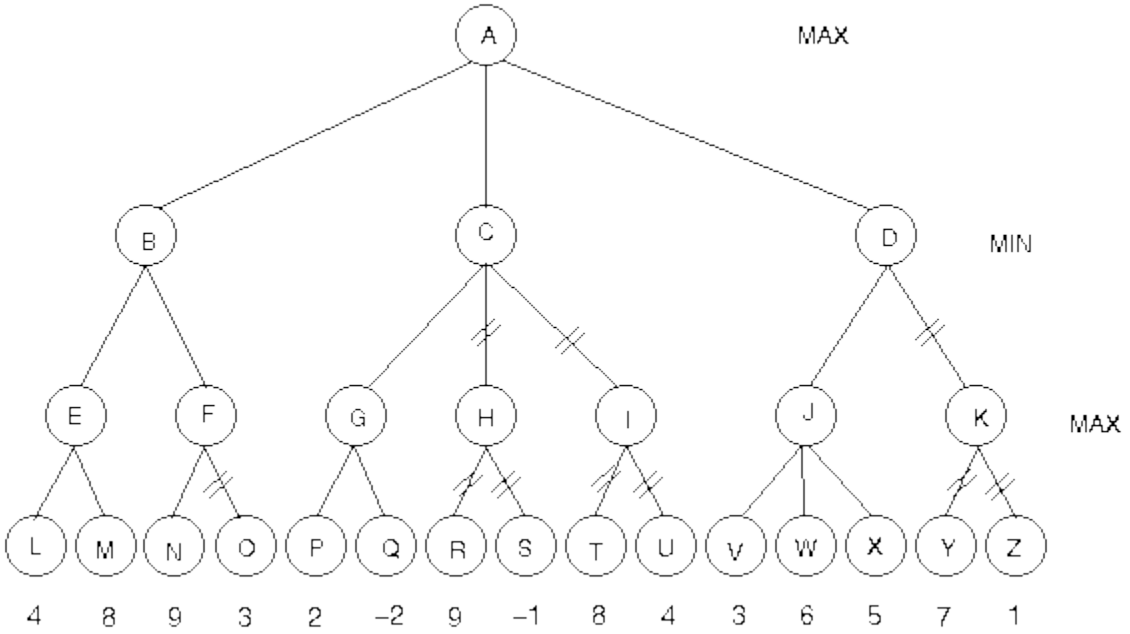
- To avoid searching for the ancestor nodes in order to make the above tests, we can carry *down* the tree the best values found so far at the ancestors. That is, at a MAX node *n*, beta

= minimum of all the beta values at MIN node ancestors of *n*. Similarly, at a MIN node *n*, alpha = maximum of all the alpha values at MAX node ancestors of *n*. Thus, now at each non-leaf node we'll store both an alpha value and a beta value.

- Initially, assign to the root values of alpha = -infinity and beta = +infinity
- See the text for a pseudocode description of the full Alpha-Beta algorithm

Example of Alpha-Beta Algorithm on a 3-Ply Search Tree

Below is a search tree where a beta cutoff occurs at node F and alpha cutoffs occur at nodes C and D. In this case we've pruned 10 nodes (O,H,R,S,I,T,U,K,Y,Z) from the 26 that are generated by Minimax.



Effectiveness of Alpha-Beta

- Alpha-Beta is guaranteed to compute the same minimax value for the root node as computed by Minimax
- In the worst case Alpha-Beta does NO pruning, examining *b^d* leaf nodes, where each node has *b* children and a *d*-ply search is performed
- In the best case, Alpha-Beta will examine only *(2b)^(d/2)* leaf nodes. Hence if you hold fixed the number of leaf nodes (as a measure of the amount of time you have allotted before a decision must be made), then you can search twice as deep as Minimax!

- The best case occurs when each player's best move is the leftmost alternative (i.e., the first child generated). So, at MAX nodes the child with the largest value is generated first, and at MIN nodes the child with the smallest value is generated first.
- In the chess program Deep Blue, they found empirically that Alpha-Beta pruning meant that the average branching factor at each node was about 6 instead of about 35-40

State of the art game programs

Some researchers believe that game playing has no importance in main stream AI. But game playing programs continue to generate excitement and a steady stream of innovations that have been adopted by a wider community.

Chess

Deep blue program

In 1997, the Deep Blue program defeated world chess champion, Garry Kasparov. Deep Blue was developed by Campbell, Hsu and Hoane at IBM. The machine was a parallel computer with 30 IBM processors and 480 VLSI chess processors. Deeep Blue used iterative deepening alpha beta search procedure. Deep Blue searched 126 million nodes per second on average. Search reached depth 14 routinely. The evaluation function had over 8000 features. The success of Deep Blue shows that progress in computer game playing has come from powerful hardware, search extensions and good evaluation function.

Fritz

In 2002, the program Fritz played against world champion Vladimir Kramnik. The game ended in a draw. The hardware was an ordinary PC.

Checkers

Arthur Samuel of IBM, developed a checkers program. It learned its own evaluation function by playing itself thousands of times. In 1962, it defeated Nealy, a champion in checkers.

Chinook was developed by Schaeffer. Chinook played against world champion Dr. Tinsley in 1990. Chinook won the game.

Othello

Is a popular computer game. It has only 5 to 15 moves. In 1997, the Logistello program defeated human world champion, Murakami.

Backgammon

Garry Tesauro developed the program TD- gammon. It is ranked among the top 3 players in the world.

Go

It is the most popular board game in Asia. The programs for playing Go are Geomate and Go4++.

Bridge

Bridge is a multiplayer game with 4 players. Bridge Baron program won the 1997 bridge championship. GIB program won the 2000 championship.

MODULE-5

Genetic Algorithm (GA) is a search-based optimization technique based on the principles of **Genetics and Natural Selection**.

It is frequently used to find optimal or near-optimal solutions to difficult problems which otherwise would take a lifetime to solve.

It is frequently used to solve optimization problems, in research, and in machine learning.

- Introduction to Optimization

Optimization is the process of **making something better**. In any process, we have a set of inputs and a set of outputs as shown in the following figure.

Optimization refers to finding the values of inputs in such a way that we get the "best" output values. The definition of "best" varies from problem to problem, but in mathematical terms, it refers to maximizing or minimizing one or more objective functions, by varying the input parameters.

The set of all possible solutions or values which the inputs can take make up the search space. In this search space, lies a point or a set of points which gives the optimal solution. The aim of optimization is to find that point or set of points in the search space.

- What are Genetic Algorithms?

- Nature has always been a great source of inspiration to all mankind. Genetic Algorithms (GAs) are search based algorithms based on the concepts of natural selection and genetics.
- GAs are a subset of a much larger branch of computation known as **Evolutionary Computation**.
- GAs were developed by John Holland and his students and colleagues at the University of Michigan, most notably David E. Goldberg and has since been tried on various optimization problems with a high degree of success.
- In GAs, we have a **pool or a population of possible solutions** to the given problem.
- These solutions then undergo recombination and mutation (like in natural genetics), producing new children, and the process is repeated over various generations.

- Each individual (or candidate solution) is assigned a fitness value (based on its objective function value) and the fitter individuals are given a higher chance to mate and yield more "fitter" individuals.
- This is in line with the Darwinian Theory of "Survival of the Fittest".
- In this way we keep "evolving" better individuals or solutions over generations, till we reach a stopping criterion.
- Genetic Algorithms are sufficiently randomized in nature, but they perform much better than random local search (in which we just try various random solutions, keeping track of the best so far), as they exploit historical information as well.

➢ Advantages of GAs

GAs have various advantages which have made them immensely popular. These include −

- Does not require any derivative information (which may not be available for many real-world problems).
- Is faster and more efficient as compared to the traditional methods.
- Has very good parallel capabilities.
- Optimizes both continuous and discrete functions and also multi-objective problems.
- Provides a list of "good" solutions and not just a single solution.
- Always gets an answer to the problem, which gets better over the time.
- Useful when the search space is very large and there are a large number of parameters involved.
-

Limitations of GAs

Like any technique, GAs also suffer from a few limitations. These include −

- GAs are not suited for all problems, especially problems which are simple and for which derivative information is available.

- Fitness value is calculated repeatedly which might be computationally expensive for some problems.
- Being stochastic, there are no guarantees on the optimality or the quality of the solution.
- If not implemented properly, the GA may not converge to the optimal solution.

➢ OPERATORS IN GAs

(a) Encoding or Representation

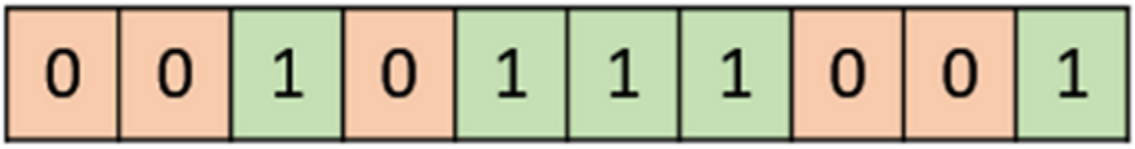
One of the most important decisions to make while implementing a genetic algorithm is deciding the representation that we will use to represent our solutions. It has been observed that improper representation can lead to poor performance of the GA.

Therefore, choosing a proper representation, having a proper definition of the mappings between the phenotype and genotype spaces is essential for the success of a GA.

Binary Representation

This is one of the simplest and most widely used representation in GAs. In this type of representation the genotype consists of bit strings.

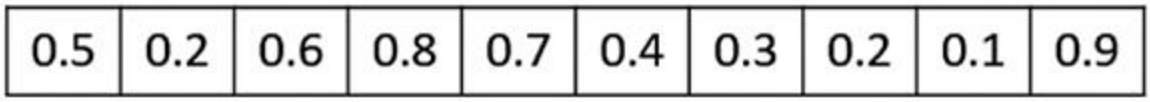
For some problems when the solution space consists of Boolean decision variables – yes or no, the binary representation is natural. Take for example the 0/1 Knapsack Problem. If there are n items, we can represent a solution by a binary string of n elements, where the x^{th} element tells whether the item x is picked (1) or not (0).



For other problems, specifically those dealing with numbers, we can represent the numbers with their binary representation. The problem with this kind of encoding is that different bits have different significance and therefore mutation and crossover operators can have undesired consequences. This can be resolved to some extent by using **Gray Coding,** as a change in one bit does not have a massive effect on the solution.

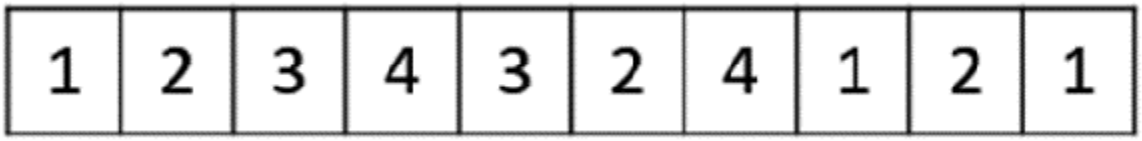
Real Valued Representation

For problems where we want to define the genes using continuous rather than discrete variables, the real valued representation is the most natural. The precision of these real valued or floating point numbers is however limited to the computer.



Integer Representation

For discrete valued genes, we cannot always limit the solution space to binary 'yes' or 'no'. For example, if we want to encode the four distances – North, South, East and West, we can encode them as **{0,1,2,3}**. In such cases, integer representation is desirable.



Permutation Representation

In many problems, the solution is represented by an order of elements. In such cases permutation representation is the most suited.

A classic example of this representation is the travelling salesman problem (TSP). In this the salesman has to take a tour of all the cities, visiting each city exactly once and come back to the starting city. The total distance of the tour has to be minimized. The solution to this TSP is naturally an ordering or permutation of all the cities and therefore using a permutation representation makes sense for this problem.

(b) Selection

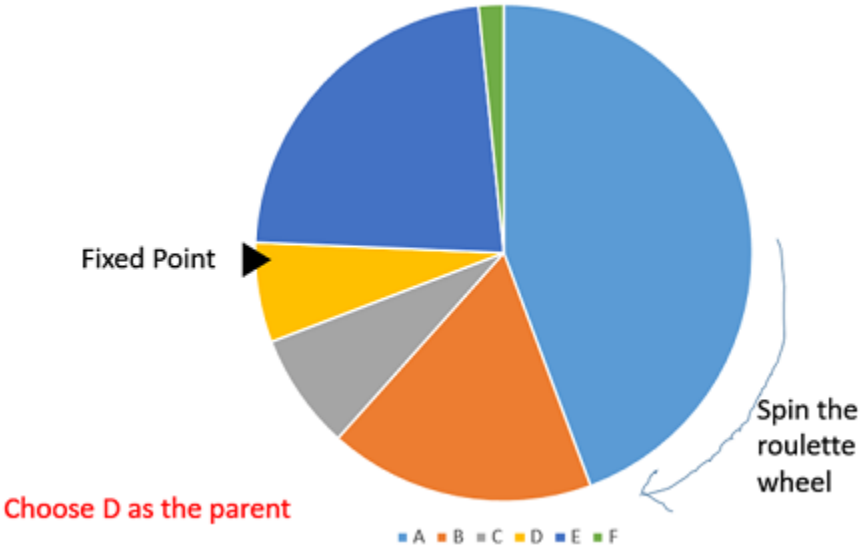
Parent Selection is the process of selecting parents which mate and recombine to create off-springs for the next generation. Parent selection is very crucial to the convergence rate of the GA as good parents drive individuals to a better and fitter solutions.

Fitness Proportionate Selection is one of the most popular ways of parent selection. In this every individual can become a parent with a probability which is proportional to its fitness. Therefore, fitter individuals have a higher chance of mating and propagating their features to the next generation. Therefore, such a selection strategy applies a selection pressure to the more fit individuals in the population, evolving better individuals over time.

Two implementations of fitness proportionate selection are possible –

Roulette Wheel Selection

In a roulette wheel selection, the circular wheel is divided as described before. A fixed point is chosen on the wheel circumference as shown and the wheel is rotated. The region of the wheel which comes in front of the fixed point is chosen as the parent. For the second parent, the same process is repeated.



Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

It is clear that a fitter individual has a greater pie on the wheel and therefore a greater chance of landing in front of the fixed point when the wheel is rotated. Therefore, the probability of choosing an individual depends directly on its fitness.

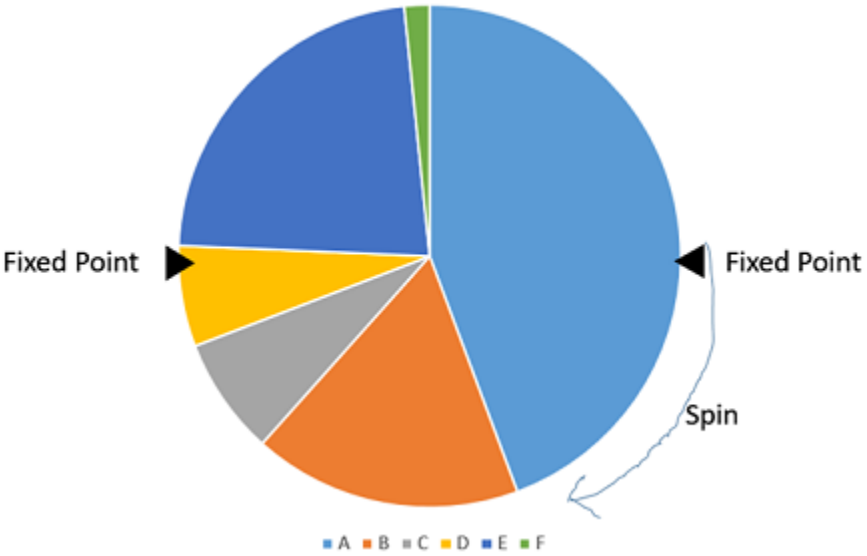
Implementation wise, we use the following steps −

- Calculate S = the sum of a finesses.
- Generate a random number between 0 and S.

- Starting from the top of the population, keep adding the finesses to the partial sum P, till P<S.
- The individual for which P exceeds S is the chosen individual.

Stochastic Universal Sampling (SUS)

Stochastic Universal Sampling is quite similar to Roulette wheel selection, however instead of having just one fixed point, we have multiple fixed points as shown in the following image. Therefore, all the parents are chosen in just one spin of the wheel. Also, such a setup encourages the highly fit individuals to be chosen at least once.

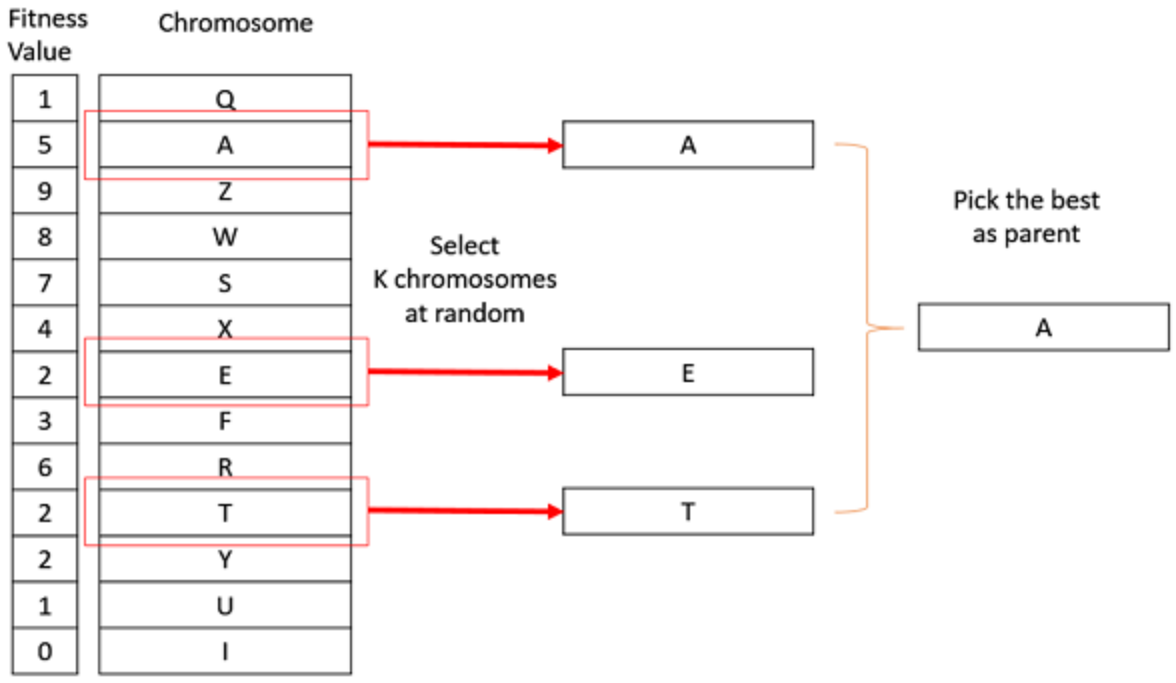


Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

It is to be noted that fitness proportionate selection methods don't work for cases where the fitness can take a negative value.

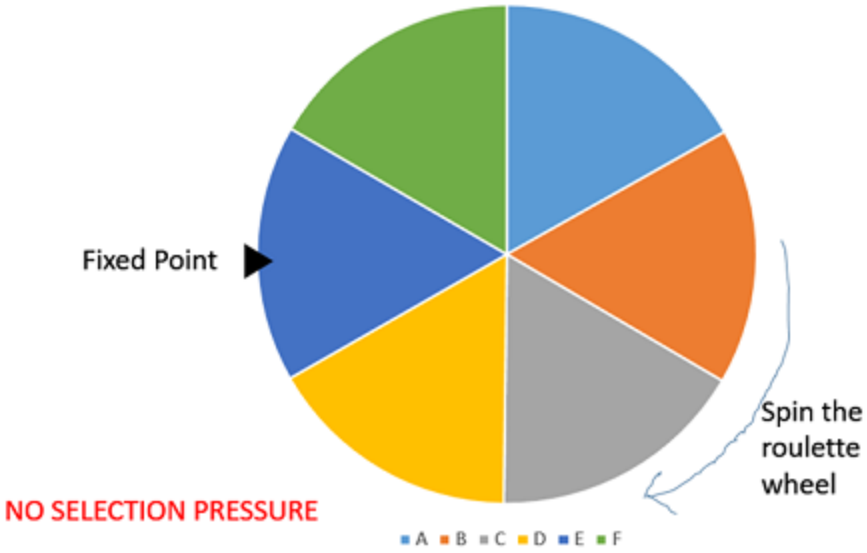
Tournament Selection

In K-Way tournament selection, we select K individuals from the population at random and select the best out of these to become a parent. The same process is repeated for selecting the next parent. Tournament Selection is also extremely popular in literature as it can even work with negative fitness values.



Rank Selection

Rank Selection also works with negative fitness values and is mostly used when the individuals in the population have very close fitness values (this happens usually at the end of the run). This leads to each individual having an almost equal share of the pie (like in case of fitness proportionate selection) as shown in the following image and hence each individual no matter how fit relative to each other has an approximately same probability of getting selected as a parent. This in turn leads to a loss in the selection pressure towards fitter individuals, making the GA to make poor parent selections in such situations.



Chromosome	Fitness Value
A	8.1
B	8.0
C	8.05
D	7.95
E	8.02
F	7.99

In this, we remove the concept of a fitness value while selecting a parent. However, every individual in the population is ranked according to their fitness. The selection of the parents depends on the rank of each individual and not the fitness. The higher ranked individuals are preferred more than the lower ranked ones.

Chromosome	Fitness Value	Rank
A	8.1	1
B	8.0	4
C	8.05	2
D	7.95	6
E	8.02	3
F	7.99	5

Random Selection

In this strategy we randomly select parents from the existing population. There is no selection pressure towards fitter individuals and therefore this strategy is usually avoided.

(c) Crossover

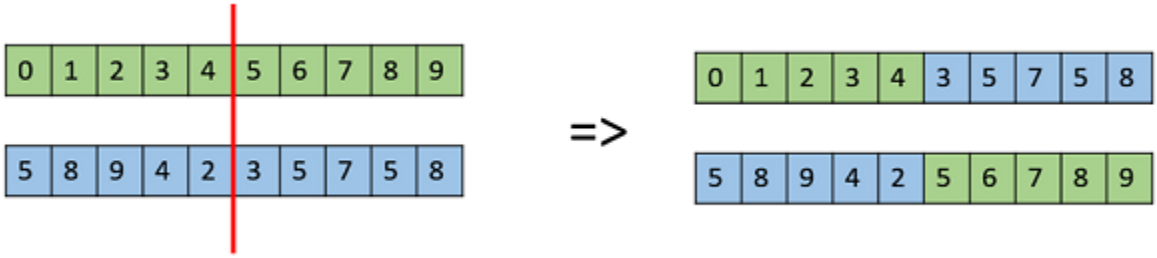
The crossover operator is analogous to reproduction and biological crossover. In this more than one parent is selected and one or more off-springs are produced using the genetic material of the parents. Crossover is usually applied in a GA with a high probability – p_c .

Crossover Operators

The crossover operators are very generic and the GA Designer might choose to implement a problem-specific crossover operator as well.

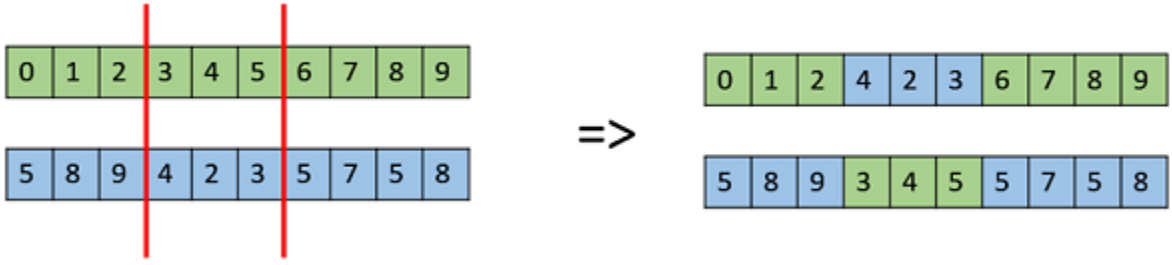
(i) One Point Crossover

In this one-point crossover, a random crossover point is selected and the tails of its two parents are swapped to get new off-springs.



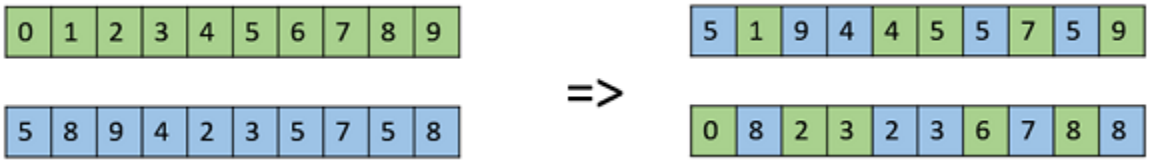
(ii) Multi Point Crossover

Multi point crossover is a generalization of the one-point crossover wherein alternating segments are swapped to get new off-springs.



(iii) Uniform Crossover

In a uniform crossover, we don't divide the chromosome into segments, rather we treat each gene separately. In this, we essentially flip a coin for each chromosome to decide whether or not it'll be included in the off-spring. We can also bias the coin to one parent, to have more genetic material in the child from that parent.



(d) Mutation

Mutation may be defined as a small random tweak in the chromosome, to get a new solution. It is used to maintain and introduce diversity in the genetic population and is usually applied with a low probability – p_m. If the probability is very high, the GA gets reduced to a random search.

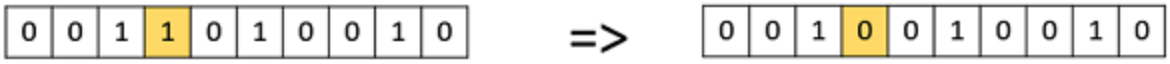
Mutation is the part of the GA which is related to the "exploration" of the search space. It has been observed that mutation is essential to the convergence of the GA while crossover is not.

Mutation Operators

Like the crossover operators, this is not an exhaustive list and the GA designer might find a combination of these approaches or a problem-specific mutation operator more useful.

(i) Bit Flip Mutation

In this bit flip mutation, we select one or more random bits and flip them. This is used for binary encoded GAs.

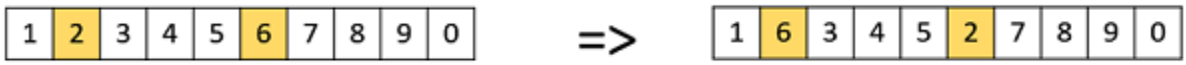


(ii) Random Resetting

Random Resetting is an extension of the bit flip for the integer representation. In this, a random value from the set of permissible values is assigned to a randomly chosen gene.

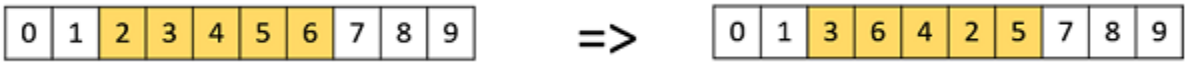
(iii) Swap Mutation

In swap mutation, we select two positions on the chromosome at random, and interchange the values. This is common in permutation based encodings.



(iv) Scramble Mutation

Scramble mutation is also popular with permutation representations. In this, from the entire chromosome, a subset of genes is chosen and their values are scrambled or shuffled randomly.



(v) Inversion Mutation

In inversion mutation, we select a subset of genes like in scramble mutation, but instead of shuffling the subset, we merely invert the entire string in the subset.

➢ STOPPING CONDITION FOR GENETIC ALGORITHM FLOW

1. *Maximum generations:*. The GA stops when the specified number of generations has evolved.

2. *Elapsed time:* The generic process will end when a specified rime has elapsed. If the maximum number of generation has been reached before the specified rime has elapsed, the process will end.

3. *No change in fitness:* The genetic process will end if there is no change to the population's best fitness for a specified number of generations. If the maximum number of generation has been reached before the specified number of generation with no changes has been reached, the process will end.

4. *Stall generations:* The algorithm stops if there is no improvement in the objective function for a sequence of consecutive generations of length "Stall generations."

5. *Stall time limit.* The algorithm stops if there is no improvement in the objective function during an interval of time in seconds equal to "Stall time limit."·

MODULE-6

Expert Systems

An Expert System is a system that employs human knowledge captured in acomputer

to solve problems that ordinarily require human expertise

ES imitate the expert's reasoning processes to solve specific problems

Mid-1960s: Special-purpose ES programs

– DENDRAL

–MYCIN

The power of an ES is derived from the specific knowledge it possesses, not from the

particular formalisms and inference schemes it employs and ES Technology Starts to go Commercial-

– XCON

– XSEL

– CATS-1

Programming Tools and Shells Appear

– EMYCIN

– EXPERT

– META-DENDRAL

– EURISKO

An Expert system is attempt to Imitate Expert Reasoning Processes and Knowledge

in Solving Specific Problems

Knowledge as Rules:Rule based ES

MYCIN rule example:

IF the infection is meningitis

AND patient has evidence of serious skin or soft tissue infection

AND organisms were not seen on the stain of the culture

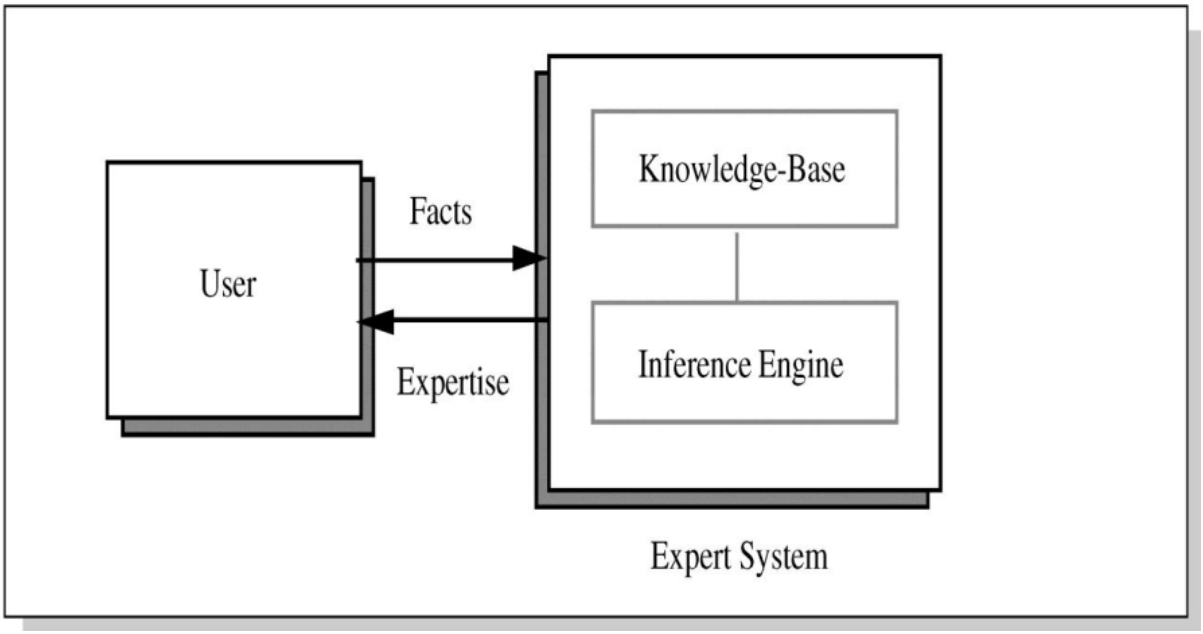
AND type of infection is bacterial

THEN There is evidence that the organism (other than those seen on cultures or smears) causing the infection is Staphylococus coagpus.

Structure of Expert Systems

- Development Environment
- Consultation (Runtime) Environment

Three Components of ES

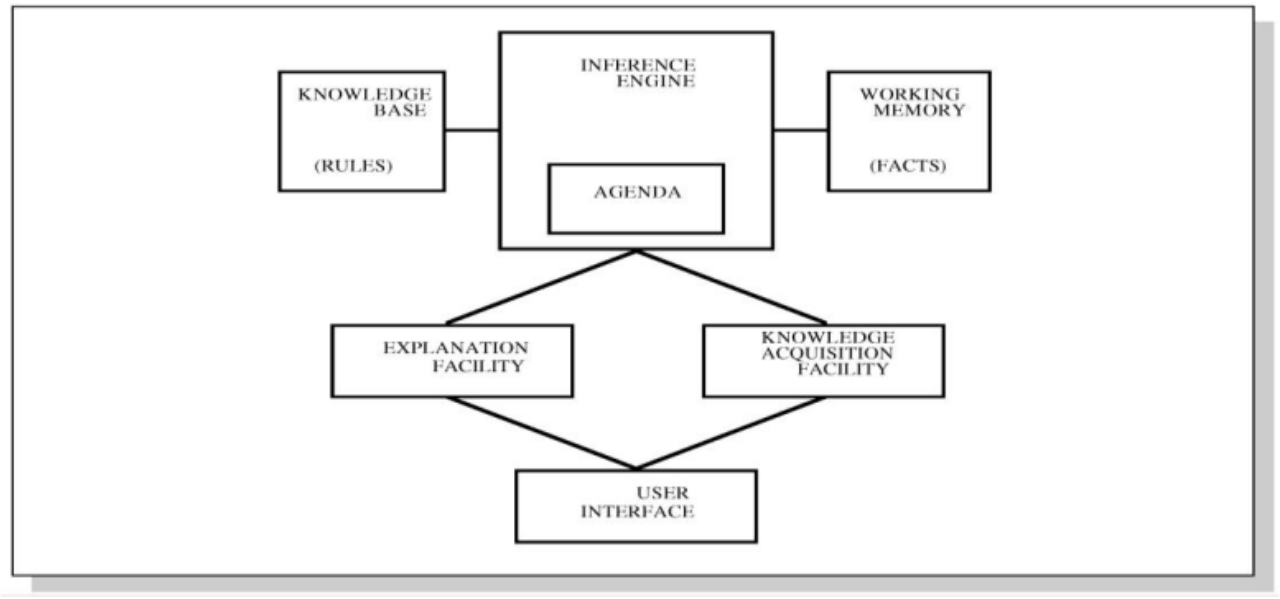


Basic Functions of Expert Systems

The process of building an expert system:

- The knowledge engineer establishes a dialog with the human expert to elicit knowledge.
- The knowledge engineer codes the knowledge explicitly in the knowledge base.
- The expert evaluates the expert system and gives a critique to the knowledge engineer.

Structure of a Rule-Based Expert System



RULE BASED EXPERT SYSTEM

- Rule-based systems (also known as production systems or expert systems) are the simplest form of artificial intelligence. A rule based system uses rules as the knowledge representation for knowledge coded into the system.
- The definitions of rule-based system depend almost entirely on expert systems, which are system that mimic the reasoning of human expert in solving a knowledge intensive problem.
- Instead of representing knowledge in a declarative, static way as a set of things which are true, rule-based system represent knowledge in terms of a set of rules that tells what to do or what to conclude in different situations.
- A rule-based system is a way of encoding a human expert's knowledge in a fairly narrow area into an automated system.
- A rule-based system can be simply created by using a set of assertions and a set of rules that specify how to act on the assertion set.
- Rules are expressed as a set of if-then statements (called IF-THEN rules or production rules):

IF P THEN Q which is also equivalent to: P⇒Q.

- A rule-based system consists of a set of IF-THEN rules, a set of facts and some interpreter controlling the application of the rules, given the facts.
- The idea of an expert system is to use the knowledge from an expert system and to encode it into a set of rules.
- When exposed to the same data, the expert system will perform (or is expected to perform) in a similar manner to the expert.
- Rule-based systems are very simple models and can be adapted and applied for a large kind of problems.
- The requirement is that the knowledge on the problem area can be expressed in the form of if-then rules. The area should also not be that large because a high number of rules can make the problem solver (the expert system) inefficient.

➢ **Elements of a Rule-Based System**

Any rule-based system consists of a few basic and simple elements as follows:

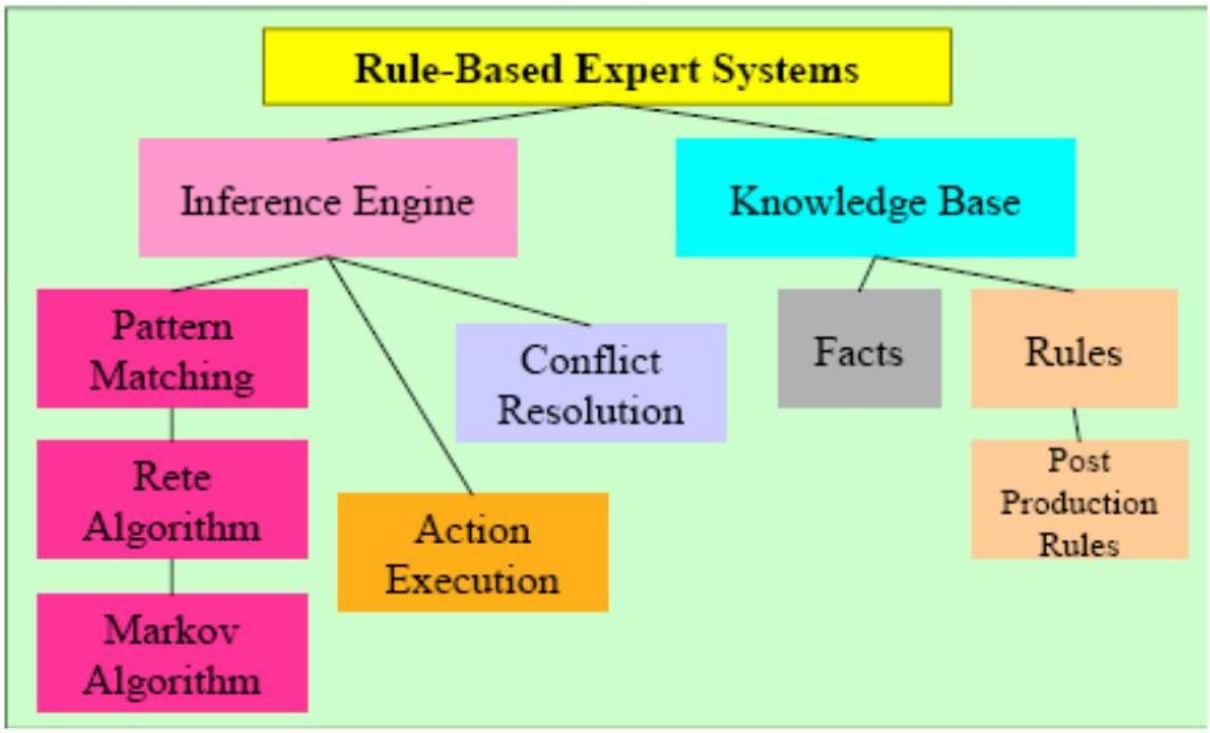
1. A set of facts. These facts are actually the assertions and should be anything relevant to the beginning state of the system.

2. A set of rules. This contains all actions that should be taken within the scope of a problem specify how to act on the assertion set. A rule relates the facts in the IF part to some action in the THEN part. The system should contain only relevant rules and avoid the irrelevant ones because the number of rules in the system will affect its performance.

3. A termination criterion. This is a condition that determines that a solution has been found or that none exists. This is necessary to terminate some rule-based systems that find themselves in infinite loops otherwise.

- Facts can be seen as a collection of data and conditions.
- Data associates the value of characteristics with a thing and conditions perform tests of the values of characteristics to determine if something is of interest, perhaps the correct classification of something or whether an event has taken place.

Foundation of Expert Systems



NATURAL LANGUAGE PROCESSING

- Natural language processing (NLP) is the ability of a computer program to understand human language as it is spoken. NLP is a component of artificial intelligence (AI).
- The development of NLP applications is challenging because computers traditionally require humans to "speak" to them in a programming language that is precise, unambiguous and highly structured, or through a limited number of clearly enunciated voice commands.
- Human speech, however, is not always precise -- it is often ambiguous and the linguistic structure can depend on many complex variables, including slang, regional dialects and social context.

➢ How natural language processing works: techniques and tools

- Syntax and semantic analysis are two main techniques used with natural language processing.

- Syntax is the arrangement of words in a sentence to make grammatical sense. NLP uses syntax to assess meaning from a language based on grammatical rules.
- Syntax techniques used include parsing (grammatical analysis for a sentence), word segmentation (which divides a large piece of text to units), sentence breaking (which places sentence boundaries in large texts), morphological segmentation (which divides words into groups) and stemming (which divides words with inflection in them to root forms).
- Semantics involves the use and meaning behind words. NLP applies algorithms to understand the meaning and structure of sentences.
- Techniques that NLP uses with semantics include word sense disambiguation (which derives meaning of a word based on context), named entity recognition (which determines words that can be categorized into groups), and natural language generation (which will use a database to determine semantics behind words).
- Current approaches to NLP are based on deep learning, a type of AI that examines and uses patterns in data to improve a program's understanding.
- Deep learning models require massive amounts of labeled data to train on and identify relevant correlations, and assembling this kind of big data set is one of the main hurdles to NLP currently.
- Earlier approaches to NLP involved a more rules-based approach, where simpler machine learning algorithms were told what words and phrases to look for in text and given specific responses when those phrases appeared.
- But deep learning is a more flexible, intuitive approach in which algorithms learn to identify speakers' intent from many examples, almost like how a child would learn human language.
- Three tools used commonly for NLP include NLTK, Gensim, and Intel NLP Architect. NTLK, Natural Language Toolkit, is an open source python modules with data sets and tutorials.
- Gensim is a Python library for topic modeling and document indexing. Intel NLP Architect is also another Python library for deep learning topologies and techniques.

- Uses of natural language processing

- Research being done on natural language processing revolves around search, especially enterprise search.
- This involves allowing users to query data sets in the form of a question that they might pose to another person.
- The machine interprets the important elements of the human language sentence, such as those that might correspond to specific features in a data set, and returns an answer.
- NLP can be used to interpret free text and make it analyzable. There is a tremendous amount of information stored in free text files, like patients' medical records, for example.
- Before deep learning-based NLP models, this information was inaccessible to computer-assisted analysis and could not be analyzed in any systematic way. But NLP allows analysts to sift through massive troves of free text to find relevant information in the files.
- Sentiment analysis is another primary use case for NLP. Using sentiment analysis, data scientists can assess comments on social media to see how their business's brand is performing, for example, or review notes from customer service teams to identify areas where people want the business to perform better.
- Google and other search engines base their machine translation technology on NLP deep learning models. This allows algorithms to read text on a webpage, interpret its meaning and translate it to another language.

Importance of NLP

- The advantage of natural language processing can be seen when considering the following two statements: "Cloud computing insurance should be part of every service level agreement" and "A good SLA ensures an easier night's sleep -- even in the cloud."
- If you use natural language processing for search, the program will recognize that *cloud computing* is an entity, that *cloud* is an abbreviated form of cloud computing and that *SLA* is an industry acronym for service level agreement.
- These are the types of vague elements that frequently appear in human language and that machine learning algorithms have historically been bad at interpreting. Now, with improvements in deep learning and artificial intelligence, algorithms can effectively interpret them.

- This has implications for the types of data that can be analyzed. More and more information is being created online every day, and a lot of it is natural human language. Until recently, businesses have been unable to analyze this data. But advances in NLP make it possible to analyze and learn from a greater range of data sources.

➢ Benefits of NLP

NLP hosts benefits such as:

- Improved accuracy and efficiency of documentation.
- The ability to automatically make a readable summary text.
- Useful for personal assistants such as Alexa.
- Allows an organization to use chatbots for customer support.
- Easier to perform sentiment analysis.

➢ Natural Language Understanding

Natural language understanding (NLU) is a branch of artificial intelligence (AI) that uses computer software to understand input made in the form of sentences in text or speech format.NLU directly enables human-computer interaction (HCI). NLU understanding of natural human languages enables computers to understand commands without the formalized syntax of computer languages and for computers to communicate back to humans in their own languages.The field of NLU is an important and challenging subset of natural language processing (NLP). While both understand human language, NLU is tasked with communicating with untrained individuals and understanding their intent, meaning that NLU goes beyond understanding words and interprets meaning. NLU is even programmed with the ability to understand meaning in spite of common human errors like mispronunciations or transposed letters or words.NLU uses algorithms to reduce human speech into a structured ontology. AI fishes out such things as intent, timing, locations and sentiments. For example, a request for an island camping trip on Vancouver Island on the 18th of august might break down something like this: Ferry tickets [intent] / need: camping lot reservation [intent] / Vancouver Island [location] / August 18th [date].

Natural-language understanding is considered an AI-hard problem.

There is considerable commercial interest in the field because of its application to automated reasoning,machine translation, question answering, news-gathering, text categorization, voice-activation, archiving, and large-scale content analysis.

Regardless of the approach used, most natural-language-understanding systems share some common components. The system needs a lexicon of the language and a parser and grammar rules to break sentences into an internal representation. The construction of a rich lexicon with a suitable ontology requires significant effort, *e.g.*, the Wordnet lexicon required many person-years of effort.

The system also needs theory from *semantics* to guide the comprehension. The interpretation capabilities of a language-understanding system depend on the semantic theory it uses. Competing semantic theories of language have specific trade-offs in their suitability as the basis of computer-automated semantic interpretation. These range from *naive semantics* or *stochastic semantic analysis* to the use of *pragmatics* to derive meaning from context. Semantic parsers convert natural-language texts into formal meaning representations.

Advanced applications of natural-language understanding also attempt to incorporate logical inference within their framework. This is generally achieved by mapping the derived meaning into a set of assertions in predicate logic, then using logical deduction to arrive at conclusions. Therefore, systems based on functional languages such as Lisp need to include a subsystem to represent logical assertions, while logic-oriented systems such as those using the language Prolog generally rely on an extension of the built-in logical representation framework.

The management of context in natural-language understanding can present special challenges. A large variety of examples and counter examples have resulted in multiple approaches to the formal modeling of context, each with specific strengths and weaknesses

- NLU or Natural Language Understanding tries to understand the meaning of given text. The nature and structure of each word inside text must be understood for NLU.
- For understanding structure, NLU tries to resolve following ambiguity present in natural language:

a) **Lexical Ambiguity** – Words have multiple meanings

b) **Syntactic Ambiguity** – Sentence having multiple parse trees.

c) **Semantic Ambiguity** – Sentence having multiple meanings

d) **Anaphoric Ambiguity** – Phrase or word which is previously mentioned but has a different meaning.

- Next, the meaning of each word is understood by using lexicons (vocabulary) and set of grammatical rules. However, there are certain different words having similar meaning (synonyms) and words having more than one meaning (polysemy).

Natural Language Generation

- It is the process of automatically producing text from structured data in a readable format with meaningful phrases and sentences.
- The problem of natural language generation is hard to deal with. It is subset of NLP.
- Natural language generation divided into three proposed stages:

1. **Text Planning** – Ordering of the basic content in structured data is done.
2. **Sentence Planning** – The sentences are combined from structured data to represent the flow of information.
3. **Realization** – Grammatically correct sentences are produced finally to represent text.

Applications of NLP:

1. Text Classification and Categorization

Text classification is an essential part in many applications, such as web searching, information filtering, language identification, readability assessment, and sentiment analysis. Neural networks are actively used for these tasks.

2. Named Entity Recognition (NER)

The main task of named entity recognition (NER) is to classify named entities, such as Guido van Rossum, Microsoft, London, etc., into predefined categories like persons, organizations, locations, time, dates, and so on. Many NER systems were already created, and the best of them use neural networks.

3. Part-of-Speech Tagging

Part-of-speech (POS) tagging has many applications including parsing, text-to-speech conversion, information extraction, and so on.

4. Semantic Parsing and Question Answering

Question Answering systems automatically answer different types of questions asked in natural languages including definition questions, biographical questions, multilingual questions, and so on. Neural networks usage makes it possible to develop high performing question answering systems.

5. Paraphrase Detection

Paraphrase detection determines whether two sentences have the same meaning. This task is especially important for question answering systems since there are many ways to ask the same question.

6. Language Generation and Multi-document Summarization

Natural language generation has many applications such as automated writing of reports, generating texts based on analysis of retail sales data, summarizing electronic medical records, producing textual weather forecasts from weather data, and even producing jokes.

7. Machine Translation

Machine translation software is used around the world despite its limitations. In some domains, the quality of translation is not good. To improve the results researchers try different techniques and models, including the neural network approach.

8. Speech Recognition

Speech recognition has many applications, such as home automation, mobile telephony, virtual assistance, hands-free computing, video games, and so on. Neutral networks are widely used in this area.

9. Character Recognition

Character Recognition systems also have numerous applications like receipt character recognition, invoice character recognition, check character recognition, legal billing document character recognition, and so on. The article Character Recognition Using Neural Network presents a method for the recognition of handwritten characters with 85% accuracy

10. Spell Checking

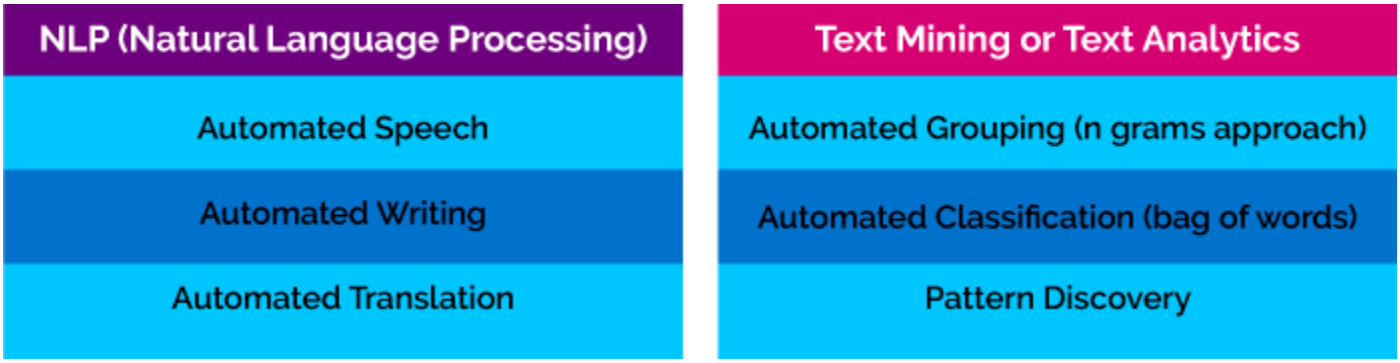
Most text editors let users check if their text contains spelling mistakes. Neural networks are now incorporated into many spell-checking tools.

Challenges associated with NLP

- NLP has not yet been wholly perfected. For example, semantic analysis can still be a challenge for NLP. Other difficulties include the fact that abstract use of language is typically tricky for programs to understand.
- For instance, NLP does not pick up sarcasm easily. These topics usually require the understanding of the words being used and the context in which the way they are being used.
- As another example, a sentence can change meaning depending on which word the speaker puts stress on. NLP is also challenged by the fact that language, and the way people use it, is continually changing.

➢ difference between NLP And Text Mining Or Text Analytics

- Natural language processing is responsible for understanding meaning and structure of given text. Text Mining or Text Analytics is a process of extracting hidden information inside text data through pattern recognition.



- Natural language processing is used to understand the meaning (semantics) of given text data, while text mining is used to understand structure (syntax) of given text data.

- As an example – I found my wallet near the bank. The task of NLP is to understand in the end that 'bank' refers to financial institute or 'river bank'

Deconstructing Language

Language is a complicated phenomenon,involving processes as varied as the recognition of sounds or printed letters,syntactic parsing high level semantic inferences and even the communication of emotional content through rhythm and reflection,

To manage this complexity, linguists have defined different levels of analysis for natural language:

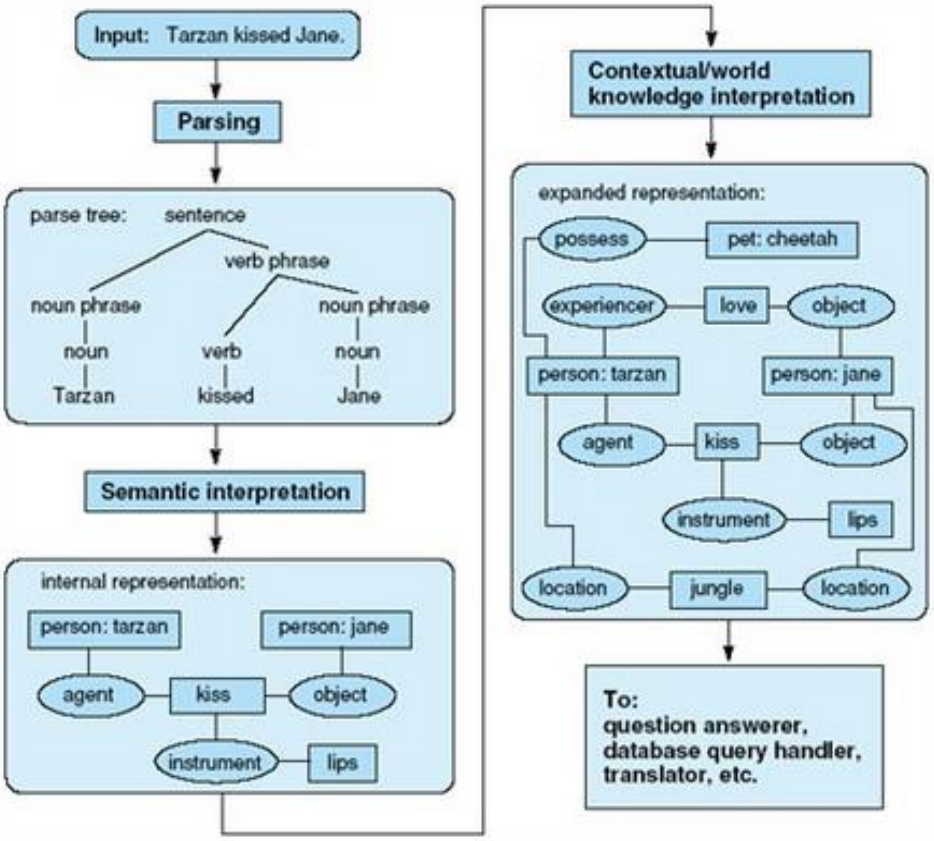
1. *Prosody* deals with the rhythm and intonation of language. This level of analysis is difficult to formalize and often neglected; however, its importance is evident in the powerful effect of poetry or religious chants, as well as the role played by rhythm in children's wordplay and the babbling of infants.
2. *Phonology* examines the sounds that are combined to form language. This branch of linguistics is important for computerized speech recognition and generation.
3. *Morphology* is concerned with the components (morphemes) that make up words. These include the rules governing the formation of words, such as the effect of prefixes (un-, non-, anti-, etc.) and suffixes (-ing, -ly, etc.) that modify the meaning of root words. Morphological analysis is important in determining the role of a word in a sentence, including its tense, number, and part of speech.
4. *Syntax* studies the rules for combining words into legal phrases and sentences, and the use of those rules to parse and generate sentences. This is the best formalized and thus the most successfully automated component of linguistic analysis.
5. *Semantics* considers the meaning of words, phrases, and sentences and the ways in which meaning is conveyed in natural language expressions.
6. *Pragmatics* is the study of the ways in which language is used and its effects on the listener. For example, pragmatics would address the reason why "Yes" is *usually* an inappropriate answer to the question "Do you know what time it is?"
7. *World knowledge* includes knowledge of the physical world, the world of human social interaction, and the role of goals and intentions in communication. This general background knowledge is essential to understand the full meaning of a text or conversation.

Stages of language analysis.

Although the specific organization of natural language understanding programs varies with different philosophies and applications.- eg. a front end for a database,an automatic translation

system , a story understanding program- all of them must translate the original sentence into an internal representation of its meaning . Generally , natural language understanding follows the stages in the below figure.

Fig 15.2 Stages in producing an internal representation of a sentence.



The **first stage** is **parsing** which analyzes the syntactic structure of sentences. Parsing both verifies that sentences are syntactically well formed and also determines a linguistic structure. By identifying the major linguistic relations such as subject-verb,verb-object and noun modifier, the parser provides a framework for semantic interpretation. This is often represented as a parse tree.

The **second stage** is **semantic representation** , which produces a representation of the meaning of the text. Other representations commonly used include conceptual dependencies , frames , and

logic -based representations. Semantic interpretations use knowledge about the meaning of words and linguistic structure,such as case roles of nouns or the transitivity of verbs.

In the **third stage**, structures from the knowledge base are added to the internal representation of the sentence to produce an expanded representation of the sentence,s meaning. This adds the required world knowledge for complete understanding. These stages exist in all systems,although they may or may not correspond to distinct software modules.

Stochastic tools for language analysis

Introduction

Statistical language techniques are methods which arise when we view natural language as a random process. In everyday parlance,randomness suggests lack of structure, definition or understanding. However viewing natural language as a random process generalizes the deterministic view point. That is ,statistical or stochastic techniques can accurately model both those parts of language which are well defined as well as those parts which indeed do have some degree of randomness.

Viewing language as a random process allows us to redefine many of the basic problems within natural language understanding in a rigorous , mathematical manner. As an example of the use of stochastic tools, we consider the problem of part-of-speech tagging.

Most people are familiar with this problem from grammar class. We want to label each word in a sentence as a noun,verb,preposition,adjective and so on. In addition ,if the word is a verb,we may want to know if it is active ,passive,transitive or intransitive. If the word is a noun,whether it is singular or plural and so on. Example:

Art	**is**	**a**	**lie**	**that**	**lets**	**us**	**see**	**the**	**truth**
Noun	Verb	Article	Noun	Pronoun	Verb	Pronoun	Verb	Article	Noun

To begin our analysis, we first define the problem formally. We have a set of words in our language **S_w={w_1,w_2…...w_n}**

For example {a,hardwork,…..,zygote} and a set of parts of speech or tags **S_t ={t_1,t_2,….t_m}**

A sentence with n words is a sequence of n random variables **W_1,W_2,…..W_N.**

These are called random variables because they can take on any of the values in **S_w** with some probability. The tags **T_1,T_2,...T_N** are also a sequence of random variables. The value of T_i takes on will be denoted ti and the value W_i is w_i.We want to find the sequence of values for these tags which is most likely , given the words in the sentence. Formally , we want to pick **t_1,t_2,….t_n** to maximize.

P(T_1=t_1,….T_n=t_n|W_1=w_1,…. W_n=w_n)

P(X|Y) stands for the probability of x given that Y has occurred so wew can write the above eq with random variables as..

P(t_1,...t_n| w1,...wn)……. Equation 1

we can rewrite equation 1 in a more useful manner.. as

$$P(t_1, ..., t_n \mid w_1, ..., w_n) = P(t_1, ..., t_n, w_1, ..., w_n) / P(w_1, ..., w_n)$$

and since we maximize this by choosing $t_1, ..., t_n$, we can simplify equation 1 to:

$$P(t_1, ..., t_n, w_1, ..., w_n) =$$

$$P(t_1)P(w_1 \mid t_1)P(t_2 \mid t_1, w_1) ... P(t_n \mid w_1, ..., w_n, t_1, ..., t_{n-1}) =$$

$$\prod_{i=1}^{n} P(t_i \mid t_1, ..., t_{i-1}, w_1, ..., w_{i-1}) P(w_i \mid t_1, ..., t_{i-1}, w_1, ..., w_{i-1}) \quad \text{equation 2}$$

Notice that equation 2 is equivalent to equation 1.

1. A Markov Model Approach

It is usually a complex task to maximize equations with probabilities conditioned on many other random variables such as we find in equation 2.There are three reasons for this : first. It is

difficult to store the probability of a random variable conditioned on many other random variables because the number of possible probabilities increases exponentially with the number of conditioning variables. Secondly even if we could store all of the probability values ,it is often difficult to estimate their values. Estimation is usually done by counting the number of occurrences of an event in a hand tagged training set and thus ,if an event occurs only a few times in the training set ,we will not get a good estimate of its probability. That is ,it is easier to estimate P(cat| the) than P(cat| the dog chased the)since there will be fewer occurrences of the latter in the training set. Finally, finding the chain of tags that maximizes structures like equation 2 would take too long. First we need to make some useful approximations of equation 2. The first rough attempt is:

$P(t_i \mid t_1, \ldots, t_{i-1}, w_1, \ldots, w_{i-1})$ approaches $P(t_i \mid t_{i-1})$

and

$P(w_i \mid t_1, \ldots, t_{i-1}, w_1, \ldots, w_{i-1})$ approaches $P(w_i \mid t_i)$.

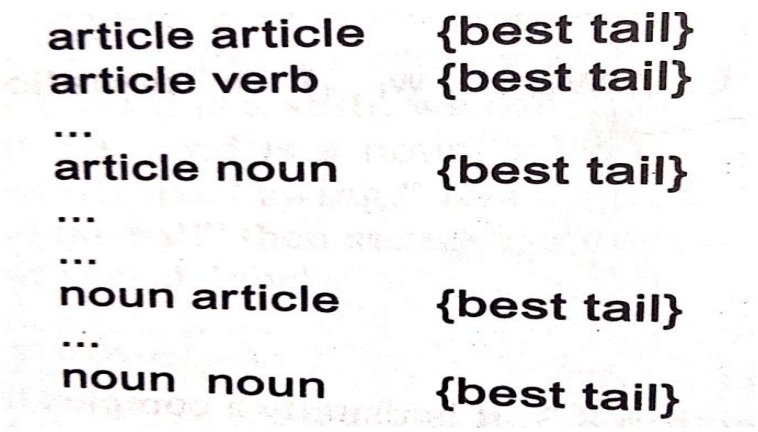
These are called *Markov assumptions* because they assume that the present thing under consideration is independent of things in the far past.

Plugging these approximations back into **equation 2**, we get

$$\prod_{i=1}^{n} P(t_i \mid t_{i-1}) P(w_i \mid t_i)$$

equation3

equation 3 is straight forward to build with because its probabilities can be easily estimated and stored. Recall that equation 3 id just an estimate of $P(t_1,...t_n|w_1,...w_n)$ and we still need to maximize it by choosing the tags , I.et, $t_1,...t_n$. Fortunately there is a dynamic programming algorithm called the viterbi algorithm which allows us to do this. This Viterbi algorithm calculates the probability of t^2 tag sequences for each word in the sentence where t is the number of possible tags. For a particular step,the tag sequences under consideration are of the following form:



where {best tail} is the most likely sequence of tags found dynamically for the n-2 words for the given n-1 tag.

There is an entry in the table for every possible values for tag number n-1 and tag number n .At each step , the algorithm finds the maximal probabilities and adds one tag to each best tail sequence. This algorithm is guaranteed to find the tag sequence which maximizes equation 3 .

Contend beyond syllabus

Reinforcement learning is the training of machine learning models to make a sequence of decisions. The agent learns to achieve a goal in an uncertain, potentially complex environment. In reinforcement learning, an artificial intelligence faces a game-like situation. The computer employs trial and error to come up with a solution to the problem. To get the machine to do what the programmer wants, the artificial intelligence gets either rewards or penalties for the actions it performs. Its goal is to maximize the total reward.
Although the designer sets the reward policy–that is, the rules of the game–he gives the model no hints or suggestions for how to solve the game. It's up to the model to figure out how to perform the task to maximize the reward, starting from totally random trials and finishing with sophisticated tactics and superhuman skills. By leveraging the power of search and many trials, reinforcement learning is currently the most effective way to hint machine's creativity.
In contrast to human beings, artificial intelligence can gather experience from thousands of parallel gameplays if a reinforcement learning algorithm is run on a sufficiently powerful computer infrastructure.

Examples of reinforcement learning

Applications of reinforcement learning were in the past limited by weak computer infrastructure. However, as Gerard Tesauro's backgamon AI superplayer developed in 1990's shows, progress did happen. That early progress is now rapidly changing with powerful new computational technologies opening the way to completely new inspiring applications.
Training the models that control autonomous cars is an excellent example of a potential application of reinforcement learning. In an ideal situation, the computer should get no instructions on driving the car. The programmer would avoid hard-wiring anything connected

with the task and allow the machine to learn from its own errors. In a perfect situation, the only hard-wired element would be the reward function.

- **For example**, in usual circumstances we would require an autonomous vehicle to put safety first, minimize ride time, reduce pollution, offer passengers comfort and obey the rules of law. With an autonomous race car, on the other hand, we would emphasize speed much more than the driver's comfort. The programmer cannot predict everything that could happen on the road. Instead of building lengthy "if-then" instructions, the programmer prepares the reinforcement learning agent to be capable of learning from the system of rewards and penalties. The agent (another name for reinforcement learning algorithms performing the task) gets rewards for reaching specific goals.
- **Another example:** deepsense.ai took part in the "Learning to run" project, which aimed to train a virtual runner from scratch. The runner is an advanced and precise musculoskeletal model designed by the Stanford Neuromuscular Biomechanics Laboratory. Learning the agent how to run is a first step in building a new generation of prosthetic legs, ones that automatically recognize people's walking patterns and tweak themselves to make moving easier and more effective. While it is possible and has been done in Stanford's labs, hard-wiring all the commands and predicting all possible patterns of walking requires a lot of work from highly skilled programmers.

Challenges with reinforcement learning

The main challenge in reinforcement learning lays in preparing the simulation environment, which is highly dependant on the task to be performed. When the model has to go superhuman in Chess, Go or Atari games, preparing the simulation environment is relatively simple. When it comes to building a model capable of driving an autonomous car, building a realistic simulator is crucial before letting the car ride on the street. The model has to figure out how to brake or avoid a collision in a safe environment, where sacrificing even a thousand cars comes at a minimal cost. Transferring the model out of the training environment and into to the real world is where things get tricky.

Scaling and tweaking the neural network controlling the agent is another challenge. There is no way to communicate with the network other than through the system of rewards and penalties.

This in particular may lead to *catastrophic forgetting*, where acquiring new knowledge causes some of the old to be erased from the network.

What distinguishes reinforcement learning from deep learning and machine learning?

In fact, there should be no clear divide between machine learning, deep learning and reinforcement learning. It is like a parallelogram – rectangle – square relation, where machine learning is the broadest category and the deep reinforcement learning the most narrow one. In the same way, reinforcement learning is a specialized application of machine and deep learning techniques, designed to solve problems in a particular way.