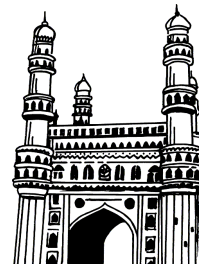


Rahul's 4
Topper's Voice

**NEW
SYLLABUS**



M.Sc.




(COMPUTER SCIENCE)

II Year IV Sem

(Osmania University)

**LATEST EDITION
2024**

ARTIFICIAL INTELLIGENCE

-  **Study Manual**
-  **Important Questions**
-  **Solved Model Papers**

- by -

WELL EXPERIENCED LECTURER



Rahul Publications TM
Hyderabad. Cell : 9391018098, 9505799122.

All disputes are subjects to Hyderabad Jurisdiction only

M.Sc.

II Year IV Sem

ARTIFICIAL INTELLIGENCE

Inspite of many efforts taken to present this book without errors, some errors might have crept in. Therefore we do not take any legal responsibility for such errors and omissions. However, if they are brought to our notice, they will be corrected in the next edition.

© No part of this publications should be reporduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording and/or otherwise without the prior written permission of the publisher



Sole Distributors :

Cell : 9391018098, 9505799122

VASU BOOK CENTRE

Shop No. 2, Beside Gokul Chat, Koti, Hyderabad.

Maternity Hospital Opp. Lane, Narayan Naik Complex, Koti, Hyderabad.

Near Andhra Bank, Subway, Sultan Bazar, Koti, Hyderabad -195.

ARTIFICIAL INTELLIGENCE

CONTENTS

Important Questions	V - VIII
Model Paper - I	IX - IX
Model Paper - II	X - X
Model Paper - III	XI - XI
Unit - I	1 - 24
Unit - II	25 - 54
Unit - III	55 - 90
Unit - IV	91 - 146

SYLLABUS

UNIT - I

Introduction to Artificial Intelligence: introduction, AI techniques, problem solving with AI, AI models, data acquisition and learning aspects in AI. Problem Solving: problem-solving process, formulating problem, problem types and characteristics, problem analysis and representation, problem space and search, toy problems, real-world problems, problem reduction methods. Uniformed Search: general search algorithm, uniformed search methods – BFS, uniform cost search, DFS, DLS, IS, bi-directional search, comparison of the uniformed techniques.

UNIT - II

Informed Search: generate and test, best first search, greedy search, A* search, memory bounded heuristic search, heuristic function, AO* search, local search algorithms and optimization problems, adversarial search methods (game theory), online search algorithms. What is an intelligent agent? Types of agent, what is constraint satisfaction problem (CSP), CSP as search problem, local search for CSP, formulating problem structure. Knowledge and Reasoning: knowledge representation, knowledge-based agents, the wumpus world, logic, propositional logic, predicate logic, unification and lifting: inference in FOL, representing knowledge using rules, semantic networks, frame systems, inference, types of reasoning.

UNIT - III

Uncertain Knowledge and Reasoning: uncertainty and methods, Bayesian probability and belief network, probabilistic reasoning, probabilistic reasoning over time, forward and backward reasoning, perception, making simple decisions, making complex decisions, other techniques in uncertainty and reasoning process. Planning problem, simple planning agent, planning languages, blocks world, goal stack planning, means-ends analysis, planning as a state-space search. Learning: what is machine learning? Learning paradigms, learning concepts, methods and models, statistical learning methods, artificial neural networks-based learning, support vector machines, reinforcement learning.

UNIT - IV

Expert Systems: architecture of expert system, confidence factors, existing expert systems, knowledge acquisition, shell and explanations, self-explaining system, rule-based expert systems, forward and backward chaining, frame-based expert systems, uncertainty management in expert systems, expert system and DSS, pros and cons of expert systems, case study. Pattern Recognition: machine perception and pattern recognition, feature extraction, classification, object recognition, speech recognition, pattern mining. Game Playing: important concepts of game theory, game playing and knowledge structure, game as search problem, alpha-beta pruning, game theory problems, robotics. Concepts and terminology of ANN, feed-forward NN, feedback networks, pattern associative networks, competitive learning, fuzzy sets, fuzzy inference process, neuro-fuzzy systems, range of AI applications, AI applications and examples, case study: agricultural domain – farmer's intelligent assistant.

Contents

Topic	Page No.
UNIT - I	
1.1 Introduction To Artificial Intelligence	1
1.1.1 Introduction	1
1.1.2 AI Techniques	3
1.1.3 Problem Solving with AI	3
1.1.4 AI Models	4
1.1.5 Data Acquisition and Learning Aspects in AI	5
1.2 Problem Solving	6
1.2.1 Problem-solving Process	6
1.2.2 Formulating Problem	6
1.2.3 Problem Types and Characteristics	7
1.2.4 Problem Analysis and Representation	10
1.2.5 Problem Space and Search	11
1.2.6 Toy Problems	12
1.2.7 Real-World Problems	16
1.2.8 Problem Reduction Methods	16
1.3 Uniformed Search	17
1.3.1 General Search Algorithm	17
1.3.2 Uniformed Search Methods - BFS	18
1.3.3 Uniform Cost Search	19
1.3.4 DFS	20
1.3.5 DLS	21
1.3.6 IS	22
1.3.7 Bi-Directional Search	23
1.3.8 Comparison of the Uniformed Techniques	24
UNIT - II	
2.1 Informed Search	25
2.1.1 General and Test	25
2.1.2 Best First Search	25
2.1.3 Greedy Search	26
2.1.4 A* Search	26

Topic	Page No.
2.1.5 Memory Bounded Heuristic Search.	27
2.1.6 Heuristic Function	27
2.1.7 AO* Search	28
2.1.8 Local Search Algorithms and Optimization Problems	29
2.1.9 Adversarial Search Methods (Game Theory)	31
2.1.10 Online Search Algorithms	33
2.2 What is an Intelligent Agent	33
2.2.1 Types of Agent	33
2.2.2 What is Constraint Satisfaction Problem (CSP)	35
2.2.3 CSP as Search Problem	37
2.2.4 Local Search for CSP	38
2.2.5 Formulating Problem Structure	40
2.3 Knowledge And Reasoning	41
2.3.1 Knowledge Representation	41
2.3.2 Knowledge-Based Agents	42
2.3.3 The Wumpus World	43
2.3.4 Logic	45
2.3.5 Propositional Logic	46
2.3.6 Predicate Logic	48
2.4 Unification and Lifting	49
2.4.1 Inference in FOL	49
2.4.2 Representing Knowledge Using Rules	52
2.4.3 Semantic Networks	52
2.4.4 Frame Systems	53
2.4.5 Inference	53
2.4.6 Types of Reasoning	54
UNIT - III	
3.1 Uncertain Knowledge and Reasoning	55
3.1.1 Uncertainty and Methods	55
3.1.2 Bayesian Probability and Belief Network	56
3.1.3 Probabilistic Reasoning	58
3.1.4 Probabilistic Reasoning Over Time	59

Topic	Page No.
3.1.5 Forward and Backward Reasoning	61
3.1.6 Perception Making Simple Decisions.....	63
3.1.7 Making Complex Decisions	64
3.1.8 Other Techniques in Uncertainty and Reasoning Process.....	66
3.2 Planning Problem.....	67
3.2.1 Simple Planning Agent	69
3.2.2 Planning Languages	69
3.2.3 Blocks World	70
3.2.4 Goal Stack Planning Means-ends Analysis	73
3.2.5 Planning as a State-space Search	76
3.3 Learning.....	76
3.3.1 What Is Machine Learning?	76
3.3.2 Learning Paradigms	77
3.3.3 Learning Concepts	78
3.3.4 Methods and Models.....	80
3.3.5 Statistical Learning Methods	82
3.3.6 Artificial Neural Networks–Based Learning	84
3.3.7 Support Vector Machines	88
3.3.8 Reinforcement Learning	89
UNIT - IV	
4.1 Expert Systems	91
4.1.1 Architecture of Expert System	91
4.1.2 Confidence Factors.....	92
4.1.3 Existing Expert Systems Expert Systems - Dendral, Mycin	93
4.1.4 Knowledge Acquisition	94
4.1.5 Shell and Explanations	96
4.1.6 Self-Explaining System	97
4.1.7 Rule-based Expert Systems	97
4.1.8 Forward and Backward Chaining	98
j 4.1.9 Frame-based Expert Systems	101
4.1.10 Uncertainty Management in Expert Systems	105
4.1.11 Expert System and DSS	107
4.1.12 Pros and Cons of Expert Systems	108
4.1.13 Case Study	109

Topic	Page No.
4.2 Pattern Recognition	113
4.2.1 Machine Perception And Pattern Recognition	113
4.2.2 Feature Extraction	115
4.2.3 Classification	115
4.2.4 Object Recognition	115
4.2.5 Speech Recognition.....	116
4.2.6 Pattern Mining	117
4.3 Game Playing.....	118
4.3.1 Important Concepts of Game Theory	118
4.3.2 Game Playing and Knowledge Structure	119
4.3.3 Game As Search Problem	119
4.3.4 Alpha-beta Pruning	121
4.3.5 Game Theory Problems	122
4.3.6 Robotics	124
4.4 Concepts And Terminology Of Ann	125
4.4.1 Feed-Forward NN	125
4.4.2 Feedback Networks	126
4.4.3 Pattern Associative Networks	127
4.4.4 Competitive Learning	132
4.4.5 Fuzzy Sets	133
4.4.6 Fuzzy Inference Process	136
4.4.7 Neuro-Fuzzy Systems	138
4.4.8 Range of AI Applications	141
4.4.9 AI Applications And Examples	141
4.5 Case Study	145
4.5.1 Agricultural Domain – Farmer’s Intelligent Assistant	145

Important Questions

UNIT - I

SHORT QUESTIONS

1. What is Artificial Intelligence?
2. Write any two applications of AI.
3. What is AI Technique?
4. What is Data acquisition?
5. What is learning? What are various learning aspects in AI ?
6. What is state space search ? write about it.
7. Write a note on the real world problems that AI can solve
8. Write about depth limit search
9. Write about bi- directional search
10. Compare between the search strategies

LONG QUESTIONS

1. What is Artificial Intelligence? Write about various approaches used to define AI.
2. Write about problem solving techniques in AI.
3. Explain the steps needed to build a system to solve a particular problem.
4. Explain about problem formulation.
5. What are various types of problems.
6. Write about problem characteristics.
7. Write the solution for missionaries and cannibals problem.
8. Explain various problem reduction methods.
9. Explain BFS method.
10. Explain Iterative deepening depth-first search (IDS).

UNIT - II**SHORT QUESTIONS**

1. Write about generate and test strategy.
2. What is greedy search.
3. What is memory bounded heuristic search
4. What is Simulated Annealing?
5. What is online algorithm?
6. What is an intelligent Agent ?
7. What is constraint satisfaction problem?
8. Write a note on Knowledge representation.
9. Write a note on Semantic Nets.
10. Write a note Frame systems.

LONG QUESTIONS

1. Explain A* Algorithm
2. What is heuristic function? Explain with an example.
3. Explain AO* ALGORITHM.
4. Explain Hill Climbing Search algorithm for optimization problems
5. Explain Mini-Max Algorithm with Alpha beta pruning.
6. What is alpha-beta pruning? Explain.
7. What are the various types of agents? Write about them.
8. What is Problem Formulation? Explain How to formulate the problems in AI.
9. Explain the rules of Inference for FOL.
10. Write about Knowledge representation rules.

UNIT - III**SHORT QUESTIONS**

1. What is probabilistic reasoning?
2. Explain about Markov chains.
3. Write about backward reasoning.
4. Write about forward reasoning.
5. How perception is used in making simple decisions.
6. Write a note on simple planning agent.
7. Write about Artificial intelligence planning language STRIPS
8. What is machine learning?
9. What are Support Vector Machines ?
10. What does *Reinforcement Learning* mean?

LONG QUESTIONS

1. What is Uncertainty? How uncertainty can be handled in AI.
2. Explain in detail about belief networks.
3. What is probabilistic reasoning? Explain.
4. Explain hidden Markov models.
5. What is planning system Explain about it with the help of block world puzzle example.
6. Write about planning as state-space search.
7. Explain about learning paradigms.
8. Explain about various models in machine learning.
9. Explain about statistical learning methods.
10. Write about Artificial Neural network based learning.

UNIT - IV**SHORT QUESTIONS**

1. What is Expert System?
2. What are confidence factors in expert systems
3. What is rule based expert systems
4. Distinguish between expert systems and DSS
5. What is Pattern Recognition
6. What Is Object Recognition?
7. What is speech recognition?
8. What is pattern mining ? write about it
9. What is game theory?
10. What is Artificial Neural Network?

LONG QUESTIONS

1. Explain Expert system shells.
2. Explain about rule based expert systems.
3. Explain frame-based expert systems.
4. Explain uncertainty management in expert systems.
5. Write about the advantages and disadvantages of expert systems.
6. What is Machine Perception?
7. What is Pattern Recognition and Explain about the approaches of pattern recognition.
8. What is Object Recognition? Explain object recognition techniques.
9. Explain about Feed Forward and feed back Neural Networks.
10. What are Fuzzy Sets ? Write about how to implement fuzzy sets.

FACULTIES OF COMPUTER SCIENCE

M.Sc. IV Semester Examination

Model Paper - I

ARTIFICIAL INTELLIGENCE

Time : 3 Hours]

[Max. Marks : 80

PART- A (8 × 4 = 32 M)

Answer all Questions

Each question carries equal marks

ANSWERS

- | | |
|---------------------------------------|----------------------|
| 1. What is Artificial Intelligence? | (Unit-I, Q.No. 1) |
| 2. Write about depth limit search. | (Unit-I, Q.No. 22) |
| 3. What is an intelligent Agent ? | (Unit-II, Q.No. 13) |
| 4. Write a note on Semantic Nets. | (Unit-II, Q.No. 27) |
| 5. Write about backward reasoning. | (Unit-III, Q.No. 8) |
| 6. What are Support Vector Machines? | (Unit-III, Q.No. 24) |
| 7. What is Expert System? | (Unit-IV, Q.No. 1) |
| 8. What is Artificial Neural Network? | (Unit-IV, Q.No. 31) |

PART- B (4 × 12 = 48 M)

Answer any four from the following

- | | |
|--|----------------------|
| 9 a) Explain about problem formulation. | (Unit-I, Q.No. 9) |
| b) Explain Iterative deepening depth-first search (IDS). | (Unit-I, Q.No. 23) |
| 10a). Explain AO* ALGORITHM. | (Unit-II, Q.No. 7) |
| b) Explain the rules of Inference for FOL. | (Unit-II, Q.No. 25) |
| 11a) Explain in detail about belief networks. | (Unit-III, Q.No. 3) |
| b). What is planning system Explain about it with the help of block world puzzle example. | (Unit-III, Q.No. 12) |
| 12a). What is Pattern Recognition and Explain about the approaches of pattern recognition. | (Unit-IV, Q.No. 17) |
| b) What are Fuzzy Sets ? Write about how to implement fuzzy sets. | (Unit-IV, Q.No. 37) |

FACULTIES OF COMPUTER SCIENCE**M.Sc. IV Semester Examination****Model Paper - II****ARTIFICIAL INTELLIGENCE**

Time : 3 Hours]

[Max. Marks : 80

PART- A (8 × 4 = 32 M)*Answer all Questions**Each question carries equal marks***ANSWERS**

- | | |
|--|----------------------|
| 1. What is AI Technique? | (Unit-I, Q.No. 3) |
| 2. What is learning? What are various learning aspects in AI ? | (Unit-I, Q.No. 7) |
| 3. What is Simulated Annealing? | (Unit-II, Q.No. 9) |
| 4. Write a note on Knowledge representation. | (Unit-II, Q.No. 19) |
| 5. Write about forward reasoning. | (Unit-III, Q.No. 8) |
| 6. What is machine learning? | (Unit-III, Q.No. 18) |
| 7. What is Pattern Recognition ? | (Unit-IV, Q.No. 17) |
| 8. What is speech recognition? | (Unit-IV, Q.No. 21) |

PART- B (4 × 12 = 48 M)*Answer any four from the following*

- | | |
|---|----------------------|
| 9. a) Write about problem solving techniques in AI. | (Unit-I, Q.No. 4) |
| b) Explain how to analyse and represent the problem. | (Unit-I, Q.No. 12) |
| 10a). Explain Hill Climbing Search algorithm for optimization problems. | (Unit-II, Q.No. 8) |
| b) Write about Knowledge representation rules. | (Unit-II, Q.No. 26) |
| 11a) What is Uncertainty? How uncertainty can be handled in AI. | (Unit-III, Q.No. 1) |
| b) Explain about statistical learning methods. | (Unit-III, Q.No. 22) |
| 12a) Compare Forward and Backward chaining with an example. | (Unit-IV, Q.No. 10) |
| b) Write about feed back networks. | (Unit-IV, Q.No. 33) |

FACULTIES OF COMPUTER SCIENCE**M.Sc. IV Semester Examination*****Model Paper - III*****ARTIFICIAL INTELLIGENCE**

Time : 3 Hours]

[Max. Marks : 80

PART- A (8 × 4 = 32 M)*Answer all Questions**Each question carries equal marks***ANSWERS**

- | | |
|--|----------------------|
| 1. What is Data acquisition? | (Unit-I, Q.No. 6) |
| 2. What is state space search ? write about it. | (Unit-I, Q.No. 13) |
| 3. What is online algorithm? | (Unit-II, Q.No. 12) |
| 4. What is an intelligent Agent ? | (Unit-II, Q.No. 13) |
| 5. Write a note on simple planning agent. | (Unit-III, Q.No. 13) |
| 6. What does <i>Reinforcement Learning</i> mean? | (Unit-III, Q.No. 25) |
| 7. What are known as classifiers in expert system? | (Unit-IV, Q.No. 19) |
| 8. What is Object Recognition? | (Unit-IV, Q.No. 20) |

PART- B (4 × 12 = 48 M)*Answer any four from the following*

- | | |
|--|--------------------------|
| 9a) Write about AI learning Models. | (Unit-I, Q.No. 5) |
| b) Explain various problem reduction methods. | (Unit-I, Q.No. 17) |
| 10a) What is alpha-beta pruning? Explain. | (Unit-II, Q.No. 11) |
| b) What are known as Knowledge based agents? Explain them. | (Unit-II, Q.No. 20) |
| 11a) Write Planning-Goal Stack Algorithm. | (Unit-III, Q.No. 16) |
| b) Write about Artificial Neural network based learning. | (Unit-III, Q.No. 23) |
| 12a) Explain uncertainty management in expert systems. | (Unit-IV, Q.No. 12) |
| b) Explain about Feed Forward and feed back Neural Networks. | (Unit-IV, Q.No. 32 & 33) |

UNIT I

Introduction to Artificial Intelligence: introduction, AI techniques, problem solving with AI, AI models, data acquisition and learning aspects in AI. Problem Solving: problem-solving process, formulating problem, problem types and characteristics, problem analysis and representation, problem space and search, toy problems, real-world problems, problem reduction methods.

Uniformed Search: general search algorithm, uniformed search methods – BFS, uniform cost search, DFS, DLS, IS, bi-directional search, comparison of the uniformed techniques.

1.1 INTRODUCTION TO ARTIFICIAL INTELLIGENCE

1.1.1 Introduction

Q1. What is Artificial Intelligence? Write about various approaches used to define AI.

Ans :

Artificial Intelligence is the branch of computer science concerned with making computers behave like humans.

The definitions of AI according to some text books are categorized into four approaches and are summarized in the table below :

Systems that think like humans "The exciting new effort to make computers think ... machines with minds in the full and literal sense."	Systems that think rationally "The study of mental faculties through the use of computer models."
Systems that act like humans The art of creating machines that perform functions that require intelligence when performed by people."	Systems that act rationally "Computational intelligence is the study of the design of intelligent agents."

The four approaches in more detail are as follows :

a) Acting humanly : The Turing Test approach

- ▶ Test proposed by Alan Turing in 1950
- ▶ The computer is asked questions by a human interrogator.

The computer passes the test if a human interrogator, after posing some written questions, cannot tell whether the written responses come from a person or not. Programming a computer to pass, the computer need to possess the following capabilities :

- ▶ **Natural language processing** to enable it to communicate successfully in English.
- ▶ **Knowledge representation** to store what it knows or hears
- ▶ **Automated reasoning** to use the stored information to answer questions and to draw new conclusions.

- ▶ **Machine learning** to adapt to new circumstances and to detect and extrapolate patterns

To pass the complete Turing Test, the computer will need :

- ▶ **Computer vision** to perceive the objects, and
- ▶ **Robotics** to manipulate objects and move about.

b) **Thinking humanly : The cognitive modelling approach**

We need to get inside actual working of the human mind :

- through introspection – trying to capture our own thoughts as they go by;
- through psychological experiments

Allen Newell and Herbert Simon, who developed **GPS**, the “**General Problem Solver**” tried to trace the reasoning steps to traces of human subjects solving the same problems.

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

c) **Thinking rationally : The “laws of thought approach”**

The Greek philosopher Aristotle was one of the first to attempt to codify “right thinking”, that is irrefutable reasoning processes. His **sylogism** provided patterns for argument structures that always yielded correct conclusions when given correct premises—for example, “Socrates is a man; all men are mortal; therefore Socrates is mortal.”.

These laws of thought were supposed to govern the operation of the mind; their study initiated a field called **logic**.

d) **Acting rationally : The rational agent approach**

An **agent** is something that acts. Computer agents are not mere programs, but they are expected to have the following attributes also : (a)

operating under autonomous control, (b) perceiving their environment, (c) persisting over a prolonged time period, (e) adapting to change.

A **rational agent** is one that acts so as to achieve the best outcome.

Q2. Explain the applications of AI.

Ans :

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.

1.1.2 AI Techniques

Q3. What is AI Technique ?

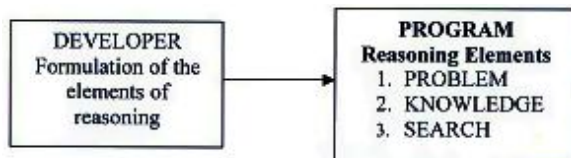
Ans :

AI problems are of many varieties and they appear to have very little in common. But there are techniques appropriate for the selection of variety of those problems, AI researches have show that "Intelligence requires knowledge", and knowledge itself posses some less desirable properties.

- ▶ It voluminous
- ▶ It is hard characterize accurately
- ▶ It is constantly changing
- ▶ It differs from data
- ▶ It is organized data

AI techniques EXPLOIT knowledge and for this knowledge must be represented as follows.

AI techniques must be designed keeping in mind the above constraints imposed by AI problems. AI techniques EXPLOIT knowledge and for this knowledge must be represented as follows.



1. Knowledge captures generalizations: Instead of representing individual situations separately, situation that share important properties grouped together, this avoids wastage of, memory and unnecessary updation.

2. In many AI domains, most of the knowledge a program has, must be provided by people in terms of they understand.
3. It can be easily modified to correct errors.
4. It can be used in several situations even if it is not totally accurate or complete.
5. It can be used to help overcome its own sheer bulk, by helping to narrow the range of possibilities that must be considered.

AI techniques must be designed keeping in mind the above constraints imposed by AI problems.

1.1.3 Problem Solving with AI

Q4. Write about problem solving techniques in AI.

Ans :

The steps that are required to build a system to solve a particular problem are :

1. Problem Definition that must include precise specifications of what the initial situation will be as well as what final situations constitute acceptable solutions to the problem.
2. Problem Analysis, this can have immense impact on the appropriateness of varies possible techniques for solving the problem.
3. Selection of the best technique(s) for solving the particular problem.

Define the problem as a state Space Search

Consider the problem of "Playing Chess" .to build a program that could play chess, we have to specify the starting position of the chess board, the rules that define legal moves. And the board position that represent a win. The goal of the winning the game, if possible, must be made explicit.

The starting position can be described by an 8 X 8 array square in which each element square (x,y), (x varying from 1to 8 & y varying from1 to 8) describes the board position of an appropriate chess coin, the goal is any board position in which the opponent does not have a legal move and his or her "king" is under attack. The legal moves provide the way of getting from initial state of final state.

The legal moves can be described as a set of rules consisting of two parts: A left side that gives the current position and the right side that describes the change to be made to the board position. An example is shown in the following figure.

Current Position	Changing Board Position
While pawn at square (5, 2) AND Square (5, 3) is empty AND Square (5, 4) is empty	Move pawn from Square (5, 2) to Square (5, 4)

The current position of a coin on the board is its STATE and the set of all possible STATES is STATE SPACE. One or more states where the problem terminates is FINAL STATE or GOAL STATE . The state space representation forms the basis of most of the AI methods. It allows for a formal definition of the problem as the need to convert some given situation into some desired situation using a set of permissible operations. It permits the problem to be solved with the help of known techniques and control strategies to move through the problem space until goal state is found.

1.1.4 AI Models

Q5. Write about AI learning Models.

Ans :

AI Learning Models

AI learning models are classified into two types :

- ▶ Knowledge-Based Classification
- ▶ Feedback-Based Classification

1. Knowledge-Based Classification

Factoring its representation of knowledge, AI learning models can be classified in two main types: inductive and deductive.

- ▶ **Inductive Learning:** This type of AI learning model is based on inferring a general rule from datasets of input-output pairs.. Algorithms such as knowledge based inductive learning(KBIL) are a great example of this type of AI learning technique. KBIL focused on finding inductive hypotheses on a dataset with the help of background information.
- ▶ **Deductive Learning:** This type of AI learning technique starts with te series of rules and infers new rules that are more efficient in the context of a specific AI algorithm. Explanation-Based Learning(EBL) and Relevance-OBased Learning(RBL) are examples examples of deductive techniques. EBL extracts general rules from examples by “generalizing” the explanation. RBL focuses on identifying attributes and deductive generalizations from simple example.

2. Feedback-Based Classification

Based on the feedback characteristics, AI learning models can be classified as supervised, unsupervised, semi-supervised or reinforced.

- ▶ **Unsupervised Learning:** Unsupervised models focus on learning a pattern in the input data without any external feedback. Clustering is a classic example of unsupervised learning models.

- ▶ **Supervised Learning:** Supervised learning models use external feedback to learning functions that map inputs to output observations. In those models the external environment acts as a “teacher” of the AI algorithms.
- ▶ **Semi-supervised Learning :** Semi-Supervised learning uses a set of curated, labeled data and tries to infer new labels/ attributes on new data data sets. Semi-Supervised learning models are a solid middle ground between supervised and unsupervised models.
- ▶ **Reinforcement Learning :** Reinforcement learning models use opposite dynamics such as rewards and punishment to “reinforce” different types of knowledge. This type of learning technique is becoming really popular in modern AI solutions.

1.1.5 Data Acquisition and Learning Aspects in AI

Q6. What is Data acquisition ?

Ans :

Data acquisition is the process of sampling signals that measure real world physical conditions and converting the resulting samples into digital numeric values that can be manipulated by a computer. Data acquisition systems, abbreviated by the acronyms DAS or DAQ, typically convert analog waveforms into digital values for processing. The components of data acquisition systems include :

- ▶ Sensors, to convert physical parameters to electrical signals.
- ▶ Signal conditioning circuitry, to convert sensor signals into a form that can be converted to digital values.
- ▶ Analog-to-digital converters, to convert conditioned sensor signals to digital values.

Q7. What is learning? What are various learning aspects in AI ?

Ans :

- ▶ Most often heard criticisms of AI is that machines cannot be called intelligent until they are able to learn to do new things and adapt to new situations, rather than simply doing as they are told to do.
- ▶ Some critics of AI have been saying that computers cannot learn!
- ▶ Definitions of Learning: changes in the system that are adaptive in the sense that they enable the system to do the same task or tasks drawn from the same population more efficiently and more effectively the next time.
- ▶ Learning covers a wide range of phenomenon:
 - Skill refinement: Practice makes skills improve. More you play tennis, better you get.
 - Knowledge acquisition: Knowledge is generally acquired through experience.

Various learning Mechanisms

- ▶ Simple storing of computed information or rote learning, is the most basic learning activity.
- ▶ Many computer programs i e., database systems can be said to learn in this sense although most people would not call such simple storage learning.
- ▶ Another way we learn if through taking advice from others. Advice taking is similar to rote learning, but high-level advice may not be in a form simple enough for a program to use directly in problem solving.
- ▶ People also learn through their own problem-solving experience.
- ▶ Learning from examples: we often learn to classify things in the world without being given explicit rules.
- ▶ Learning from examples usually involves a teacher who helps us classify things by correcting us when we are wrong.

1.2 PROBLEM SOLVING

1.2.1 Problem-solving Process

Q8. Explain the steps needed to build a system to solve a particular problem.

Ans :

The steps that are required to build a system to solve a particular problem are :

1. Problem Definition that must include precise specifications of what the initial situation will be as well as what final situations constitute acceptable solutions to the problem.
2. Problem Analysis , this can have immense impact on the appropriateness of varies possible techniques for solving the problem.
3. Selection of the best technique(s) for solving the particular problem.

Define the Problem as State Space Search

Consider the problem of "Playing Chess" .to build a program that could play chess, we have to specify the starting position of the chess board, the rules that define legal moves. And the board position that represent a win. The goal of the winning the game, if possible, must be made explicit.

The starting position can be described by an 8 X 8 array square in which each element square (x,y),(x varying from 1 to 8 & y varying from 1 to 8) describes the board position of an appropriate chess coin, the goal is any board position in which the opponent does not have a legal move and his or her "king" is under attack. The legal moves provide the way of getting from initial state of final state.

The legal moves can be described as a set of rules consisting of two parts: A left side that gives the current position and the right side that describes the change to be made to the board position. An example is shown in the following figure.

Current Position

While pawn at square (5, 2), AND Square (5, 3) is empty, AND Square (5, 4) is empty.

Changing Board Position

Move pawn from Square (5 , 2) to Square (5, 4).

The current position of a coin on the board is its STATE and the set of all possible STATES is STATE SPACE. One or more states where the problem terminates is FINAL STATE or GOAL STATE . The state space representation forms the basis of most of the AI methods. It allows for a formal definition of the problem as the need to convert some given situation into some desired situation using a set of permissible operations. It permits the problem to be solved with the help of known techniques and control strategies to move through the problem space until goal state is found.

Some of the problems that fall within the scope of AI and the kinds of techniques will be useful to solve these problems.

1.2.2 Formulating Problem

9. Explain about problem formulation.

Ans :

Problem Formulation

- A problem formulation is about deciding what actions and states to consider, we will come to this point it shortly.

- We will describe our states as "in(CITYNAME)" where CITYNAME is the name of the city in which we are currently in.

Now suppose that our agent will consider actions of the form "Travel from city A to City B". and is standing in city 'A' and wants to travel to city 'E', which means that our current state is in(A) and we want to reach the state in(E).

There are 3 roads out of A, one toward B, one toward C and one toward D, none of these achieves the goal and will bring our agent to state in(E), given that our agent is not familiar with the geography of our alien map then it doesn't know which road is the best to take, so our agent will pick any road in random.

Now suppose that our agent is updated with the above map in its memory, the point of a map that our agent now knows what action bring it to what city, so our agent will start to study the map and consider a hypothetical journey through the map until it reaches E from A.

Once our agent has found the sequence of cities it should pass by to reach its goal it should start following this sequence.

The process of finding such sequence is called **search**, a search algorithm is like a black box which takes **problem** as input returns a **solution**, and once the solution is found the sequence of actions it recommends is carried out and this is what is called the **execution phase**.

We now have a simple (formulate, search, execute) design for our problem solving agent, so let's find out precisely how to formulate a problem.

Formulating Problems

A problem can be defined formally by 4 components :

1. Initial State :

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

2. State Description :

- a description of the possible **actions** available to the agent, it is common to describe it by means of a **successor**

function, given state x then SUCCESSOR-FN(x) returns a set of ordered pairs <action, successor> where action is a legal action from state x and successor is the state in which we can be by applying action.

- The initial state and the successor function together defined what is called **state space** which is the set of all possible states reachable from the initial state {e.i: in(A), in(B), in(C), in(D), in(E)}.

3. Goal Test :

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

4. Path cost :

- a function that assigns a numeric value to each path, each step we take in solving the problem should be somehow weighted, so if I travel from A to E our agent will pass by many cities, the cost to travel between two consecutive cities should have some cost measure, {e.i: Traveling from 'A' to 'B' costs 20 km or it can be typed as c(A, 20, B)}.

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

1.2.3 Problem Types and Characteristics

Q10. What are various types of problems ?

Ans :

Problem Types

Not all problems are created equal. There are different types of problem.

- single-state problem
- multiple-state problem
- contingency problem
- exploration problem

Single-state problem

exact prediction is possible **state** - is known exactly after any sequence of actions **accessibility** of the world all essential information can be obtained through sensors **consequences** of actions are known to the agent **goal** - for each known initial state, there is a unique goal state that is guaranteed to be reachable via an action sequence. It is simplest case, but severely restricted.

Example :

Vacuum world,

Limitations :

- ▶ Can't deal with incomplete accessibility
- ▶ incomplete knowledge about consequences changes in the world.
- ▶ indeterminism in the world, in action

Multiple-state problem

semi-exact prediction is possible **state** is not known exactly, but limited to a set of possible states after each action **accessibility** of the world not all essential information can be obtained through sensors reasoning can be used to determine the set of possible states **consequences** of actions are not always or completely known to the agent; actions or the environment might exhibit randomness **goal** due to ignorance, there may be no fixed action sequence that leads to the goal less restricted, but more complex.

Example :

Vacuum world, but the agent has no sensors.

The action sequence right, suck, left, suck is guaranteed to reach the goal state from any initial state.

Limitations :

- ▶ Can't deal with changes in the world during execution ("contingencies")

Contingency Problem

exact prediction is impossible **state** unknown in advance, may depend on the outcome of actions and changes in the environment **accessibility** of the world some essential information may be obtained through sensors only at execution time **consequences**

of action may not be known at planning time **goal** instead of single action sequences, there are trees of actions **contingency** branching point in the tree of actions **agent design** different from the previous two cases: the agent must act on incomplete plans. search and execution phases are interleaved.

Example : Vacuum world, The effect of a suck action is random.

There is no action sequence that can be calculated at planning time and is guaranteed to reach the goal state.

Limitations : Can't deal with situations in which the environment or effects of action are unknown

Exploration Problem

effects of actions are unknown **state** the set of possible states may be unknown **accessibility** of the world some essential information may be obtained through sensors only at execution time **consequences** of actions may not be known at planning time **goal** can't be completely formulated in advance because states and consequences may not be known at planning time **discovery** what states exist **experimentation** what are the outcomes of actions **learning** remember and evaluate experiments **agent design** different from the previous cases : the agent must experiment search requires search in the real world, not in an abstract model **realistic problems**, very hard.

Problem Characteristics

Q11. Write about problem characteristics.

Ans :

Heuristic search is a very general method applicable to a large class of problem . It includes a variety of techniques. In order to choose an appropriate method, it is necessary to analyze the problem with respect to the following considerations.

Is the problem decomposable ?

A very large and composite problem can be easily solved if it can be broken into smaller problems and recursion could be used. Suppose we want to solve.

$$\text{Ex : } - + \int x^2 + 3x + \sin 2x \cos 2x \, dx$$

This can be done by breaking it into three smaller problems and solving each by applying specific rules. Adding the results the complete solution is obtained.

2. Can solution steps be ignored or undone?

Problem fall under three classes ignorable , recoverable and irrecoverable. This classification is with reference to the steps of the solution to a problem. Consider thermo proving. We may later find that it is of no help. We can still proceed further, since nothing is lost by this redundant step. This is an example of ignorable solutions steps.

Now consider the 8 puzzle problem tray and arranged in specified order. While moving from the start state towards goal state, we may make some stupid move and consider theorem proving. We may proceed by first proving lemma. But we may backtrack and undo the unwanted move. This only involves additional steps and the solution steps are recoverable.

Lastly consider the game of chess. If a wrong move is made, it can neither be ignored nor be recovered. The thing to do is to make the best use of current situation and proceed. This is an example of an irrecoverable solution steps.

1. Ignorable problems Ex : theorem proving
In which solution steps can be ignored.
2. Recoverable problems Ex : 8 puzzle
In which solution steps can be undone
3. Irrecoverable problems Ex : Chess.

In which solution steps can't be undone

A knowledge of these will help in determining the control structure.

3. Is the Universal Predictable?

Problems can be classified into those with certain outcome (eight puzzle and water jug problems) and those with uncertain outcome (playing cards). in certain – outcome problems, planning could be done to generate a sequence of operators that guarantees to a lead to a solution. Planning helps to avoid unwanted solution steps.

For uncertain out come problems, planning can at best generate a sequence of operators that has a good probability of leading to a solution.

The uncertain outcome problems do not guarantee a solution and it is often very expensive since the number of solution and it is often very expensive since the number of solution paths to be explored increases exponentially with the number of points at which the outcome can not be predicted. Thus one of the hardest types of problems to solve is the irrecoverable, uncertain – outcome problems (Ex : Playing cards).

4. Is good solution absolute or relative ?

(Is the solution a state or a path ?)

There are two categories of problems. In one, like the water jug and 8 puzzle problems, we are satisfied with the solution, unmindful of the solution path taken, whereas in the other category not just any solution is acceptable. We want the best, like that of traveling sales man problem, where it is the shortest path. In any – path problems, by heuristic methods we obtain a solution and we do not explore alternatives. For the best-path problems all possible paths are explored using an exhaustive search until the best path is obtained.

5. The knowledge base consistent ?

In some problems the knowledge base is consistent and in some it is not. For example consider the case when a Boolean expression is evaluated. The knowledge base now contains theorems and laws of Boolean Algebra which are always true. On the contrary consider a knowledge base that contains facts about production and cost. These keep varying with time. Hence many reasoning schemes that work well in consistent domains are not appropriate in inconsistent domains.

Ex. Boolean expression evaluation.

6. What is the role of Knowledge?

Though one could have unlimited computing power, the size of the knowledge base available for solving the problem does matter in arriving at a good solution. Take for example the game of playing chess, just the rules for determining legal moves and

some simple control mechanism is sufficient to arrive at a solution. But additional knowledge about good strategy and tactics could help to constrain the search and speed up the execution of the program. The solution would then be realistic.

Consider the case of predicting the political trend. This would require an enormous amount of knowledge even to be able to recognize a solution, leave alone the best.

Example :

1. Playing chess
2. News paper understanding.

7. Does the task requires interaction with the person.

The problems can again be categorized under two heads.

1. Solitary in which the computer will be given a problem description and will produce an answer, with no intermediate communication and with the demand for an explanation of the reasoning process. Simple theorem proving falls under this category. Given the basic rules and laws, the theorem could be proved, if one exists.

Ex : theorem proving (give basic rules & laws to computer)

2. Conversational, in which there will be intermediate communication between a person and the computer, wither to provide additional assistance to the computer or to provide additional informed information to the user, or both problems such as medical diagnosis fall under this category, where people will be unwilling to accept the verdict of the program, if they can not follow its reasoning.

Ex : Problems such as medical diagnosis.

8. Problem Classification

Actual problems are examined from the point of view, the task here is examine an input and decide which of a set of known classes.

Ex : Problems such as medical diagnosis, engineering design.

1.2.4 Problem Analysis and Representation

Q12. Explain how to analyse and represent the problem.

Ans :

This problem can be abstracted to the mathematical problem of finding a path from a start node to a goal node in a directed graph. Many other problems can also be mapped to this abstraction, so it is worthwhile to consider this level of abstraction. Most of this chapter explores various algorithms for finding such paths.

This notion of search is computation inside the agent. It is different from searching in the world, when it may have to act in the world, for example, an agent searching for its keys, lifting up cushions, and so on. It is also different from searching the web, which involves searching for information. Searching in this chapter means searching in an internal representation for a path to a goal.

The idea of search is straightforward: the agent constructs a set of potential partial solutions to a problem that can be checked to see if they truly are solutions or if they could lead to solutions. Search proceeds by repeatedly selecting a partial solution, stopping if it is a path to a goal, and otherwise extending it by one more arc in all possible ways.

Search underlies much of artificial intelligence. When an agent is given a problem, it is usually given only a description that lets it recognize a solution, not an algorithm to solve it. It has to search for a solution. The existence of NP-complete problems, with efficient means to recognize answers but no efficient methods for finding them, indicates that searching is, in many cases, a necessary part of solving problems.

It is often believed that humans are able to use intuition to jump to solutions to difficult problems. However, humans do not tend to solve general problems; instead they solve specific instances about which they may know much more than the underlying search space. Problems in which little structure exists or in which the structure cannot be related to the physical world are very difficult for humans to solve. The existence of public key encryption codes, where the search space is clear

and the test for a solution is given - for which humans nevertheless have no hope of solving and computers cannot solve in a realistic time frame - demonstrates the difficulty of search.

The difficulty of search and the fact that humans are able to solve some search problems efficiently suggests that computer agents should exploit knowledge about special cases to guide them to a solution. This extra knowledge beyond the search space is **heuristic knowledge**. This chapter considers one kind of heuristic knowledge in the form of an estimate of the cost from a node to a goal.

1.2.5 Problem Space and Search

Q13. What is state space search ? write about it.

Ans :

State Spaces

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

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

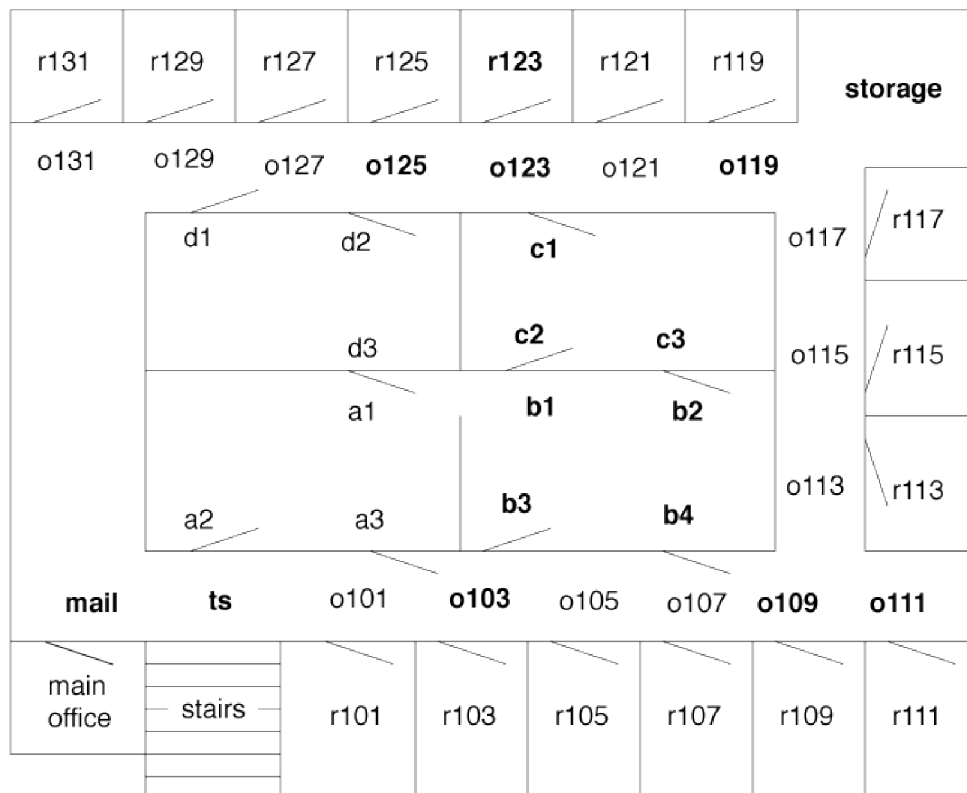


Fig. : The delivery robot domain with interesting locations labeled

Example : Consider the robot delivery domain and the task of finding a path from one location to another in Figure 3.1. This can be modeled as a state-space search problem, where the states are locations. Assume that the agent can use a lower-level controller to carry out the high-level action of getting from one location to a neighboring location. Thus, at this level of abstraction, the actions can involve deterministic traveling between neighboring locations.

An example problem is where the robot is outside room r103, at position o103, and the goal is to get to room r123. A solution is a sequence of actions that will get the robot to room r123.

1.2.6 Toy Problems

Q14. Explain tic – tac – toe problem with all solution states.

Ans :

Introductory Problem-Tic Tac Toe

The game **Tic Tac Toe** is also known as **Noughts** and **Crosses** or **Xs** and **Os**, the player needs to take turns marking the spaces in a 3x3 grid with their own marks, if 3 consecutive marks (**Horizontal, Vertical, Diagonal**) are formed then the player who owns these moves get won.

Assume,

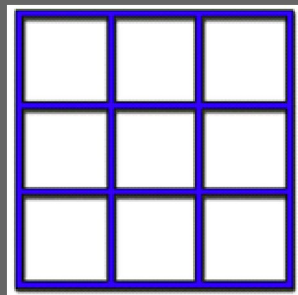
Player 1 - X

Player 2 - O

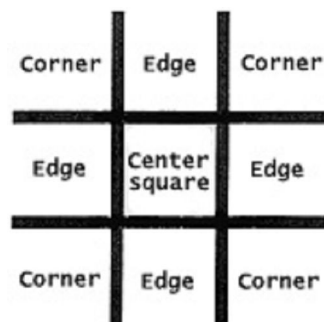
So, a player who gets 3 consecutive marks first, they will win the game .

Let's have a discussion about how a board's data structure looks and how the Tic Tac Toe algorithm works.

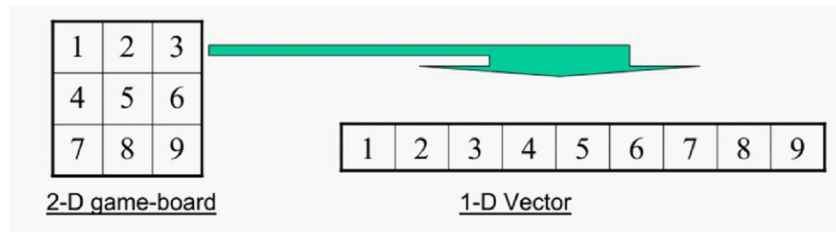
Board's Data Structure:



The cells could be represent as Centersquare, Corner, Edge as like below.



Number each square starts from 1 to 9 like following image



Consider a Board having nine elements vector. Each element will contain

- ▶ 0 for blank
- ▶ 1 indicating X player move
- ▶ 2 indicating O player move

Computer may play as X or O player. First player is always X.

Move Table

It is a vector of 3^9 elements, each element of which is a nine element vector representing board position.

Total of 3^9 (19683) elements in move table

Move Table

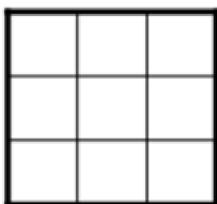
Index	Current Board position	New Board position
0	000000000	000010000
1	1000000001	020000001
2	000000002	000100002
3	000000010	002000010
.		
.		

Algorithm

To make a move, do the following :

1. View the vector (board) as a ternary number and convert it to its corresponding decimal number.
2. Use the computed number as an index into the move table and access the vector stored there.
3. The vector selected in step 2 represents the way the board will look after the move that should be made. So set board equal to that vector.

Let's start with empty board.



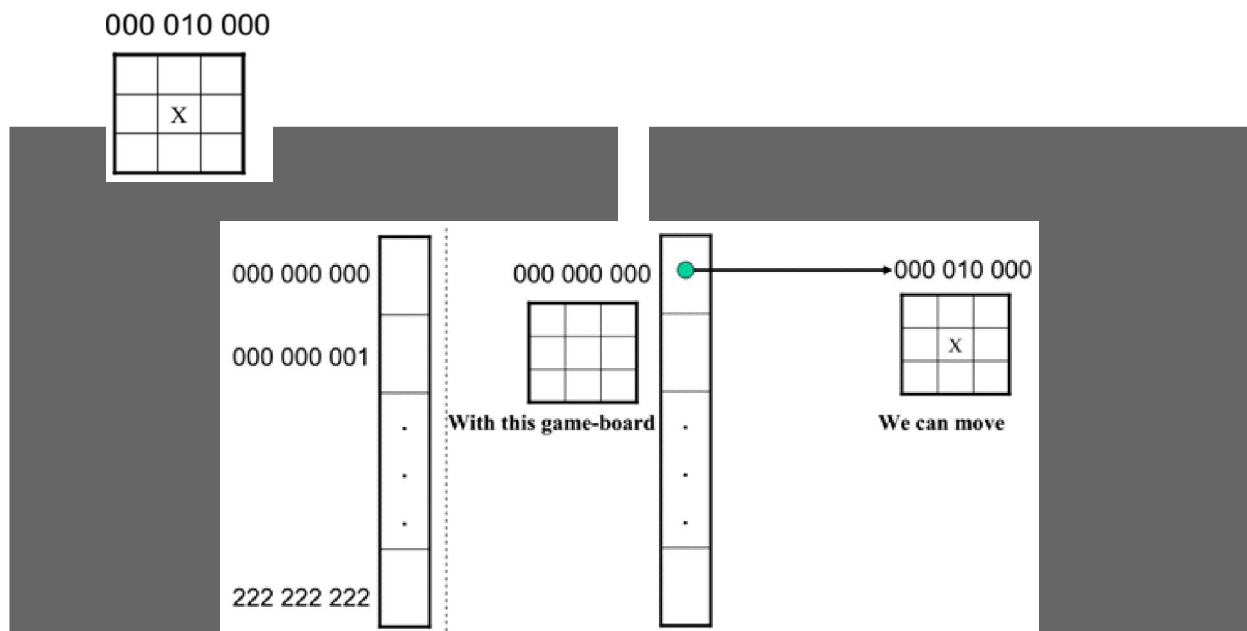
Step 1: Now our board looks like **000 000 000** (ternary number) convert it into decimal no. The decimal no is **0**

Step 2: Use the computed number ie **0** as an index into the move table and access the vector stored in New Board Position.

The new board position is **000 010 000**

Step 3: The vector selected in step 2 (**000 010 000**) represents the way the board will look after the move that should be made. So set board equal to that vector.

After complete the 3rd step your board looks like :



This process continues until the player get win or tie.

Q15. Write the solution for missionaries and cannibals problem.

Ans :

The Missionaries and Cannibals Problem Statement

Three missionaries and three cannibals find themselves on one side of a river. They have would like to get to the other side. But the missionaries are not sure what else the cannibals agreed to. So the missionaries managed the trip across the river in such a way that the number of missionaries on either side of the river is never less than the number of cannibals who are on the same side. The only boat available holds only two at a time. How can everyone get across the river without the missionaries risking being eaten?

Solution :

The state for this problem can be defined as

$\{(i, j) / i=0, 1, 2, 3, : j=0, 1, 2, 3\}$ where i represents the number missionaries in one side of a river . j represents the number of cannibals in the same side of river. The initial state is (3,3), that is three missionaries and three cannibals one side of a river, (Bank 1) and (0,0) on another side of the river (bank 2) . the goal state is to get (3,3) at bank 2 and (0,0) at bank 1.

To solve this problem we will make the following assumptions :

1. Number of cannibals should lesser than the missionaries on either side.
2. Only one boat is available to travel.
3. Only one or maximum of two people can go in the boat at a time.
4. All the six have to cross the river from bank.
5. There is no restriction on the number of trips that can be made to reach of the goal.
6. Both the missionaries and cannibals can row the boat.

The objective of the solution is to find the sequence of their transfer from one bank of river to other using the boat sailing through the river satisfying these constraints.

We can form various production rules as presented in water-jug problem. Let Missionary is denoted by 'M' and Cannibal, by 'C'. These rules are described below:

Rule 1 : (0, M) : One missionary sailing the boat from bank-1 to bank-2

Rule 2 : (M, 0) : One missionary sailing the boat from bank-2 to bank-1

Rule 3 : (M, M) : Two missionaries sailing the boat from bank-1 to bank-2

Rule 4 : (M, M) : Two missionaries sailing the boat from bank-2 to bank-1

Rule 5 : (M, C) : One missionary and one Cannibal sailing the boat from bank-1 to bank-2

Rule 6 : (C, M) : One missionary and one Cannibal sailing the boat from bank-2 to bank-1

Rule 7 : (C, C) : Two Cannibals sailing the boat from bank-1 to bank-2

Rule 8 : (C, C) : Two Cannibals sailing the boat from bank-2 to bank-1

Rule 9 : (0, C) : One Cannibal sailing the boat from bank-1 to bank-2

Rule 10 : (C, 0) : One Cannibal sailing the boat from bank-2 to bank-1

All or some of these production rules will have to be used in a particular sequence to find the solution of the problem. The rules applied and their sequence is presented in the following Table

Table: Rules applied and their sequence in Missionaries and Cannibals problem

or

After application of rule	persons in the river bank-1	persons in the river bank-2	boat position
Start state	M, M, M, C, C, C	0	bank-1
5	M, M, C, C	M, C	bank-2
2	M, M, C, C, M	C	bank-1
7	M, M, M	C, C, C	bank-2
10	M, M, M, C	C, C	bank-1
3	M, C	C, C, M, M	bank-2
6	M, C, C, M	C, M	bank-1
3	C, C	C, M, M, M	bank-2
10	C, C, C	M, M, M	bank-1
7	C	M, M, M, C, C	bank-2
10	C, C	M, M, M, C	bank-1
7	0	M, M, M, C, C, C	bank-2

1.2.7 Real-World Problems

Q16. Write a note on the real world problems that AI can solve.

Ans :

With real-time problem solving skills the only thing you have to worry about are your goals as you can leave the assistance to a computer that can think on it's own but for your benefit. Many intelligent brains working in Artificial Intelligence to make our life comfortable.

As a smart technology entrepreneur with a machine intelligence research background and passionate about advancing the state-of-the-art in machine or artificial intelligence (AI) to help solve real-world problems, Machine intelligence is not only changing the way we use our computers and smartphones but the way we interact with the real world. It is also one of the key exponential technologies

Here is the list of some real world examples can be solved by machine intelligence :

- ▶ Virtual Personal Assistants
- ▶ Video Games
- ▶ Smart Cars
- ▶ Purchase Prediction
- ▶ Fraud Detection
- ▶ Chat Bots
- ▶ Social Networking
- ▶ Real Estate
- ▶ Flying Drones
- ▶ Analytics.

1.2.8 Problem Reduction Methods

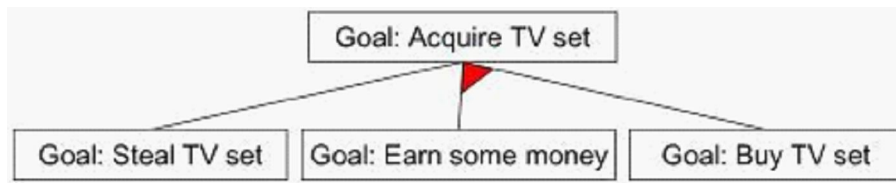
Q17. Explain various problem reduction methods.

Ans :

AND-OR Graphs

- ▶ AND-OR graph (or tree) is useful for representing the solution of problems that can be solved by decomposing them into a set of smaller problems, all of which must then be solved.
- ▶ This decomposition or reduction generates arcs that we call AND arcs.
- ▶ One AND arc may point to any numbers of successor nodes. All of which must then solved in order for the arc to point solution
- ▶ In order to find the solution in an AND-OR graph, we need an algorithm similar to best –first search but with the ability to handle the AND arcs appropriately.
- ▶ We define FUTILITY, if the estimated cost of a solution becomes greater than the value of FUTILITY then we abandon the search, FUTILITY should be chosen to correspond to a threshold.

Following figure shows AND arcs are indicated with a line connecting all the components.



The AO* Algorithm

- ▶ Rather than the two lists, OPEN and CLOSED, that used in the A* algorithm, the AO* algorithm will use a single structure GRAPH, representing the part of the search graph that has explicitly generated so far.
- ▶ Each node in the graph will point both down to its immediate successors and up to its immediate predecessors.
- ▶ Each node in the graph will also have associated with it an h' value, an estimate of the cost of a path from itself to a set of solution nodes.
- ▶ We will not store g (the cost of getting from the start node to the current node) as we did in the A* algorithm.
- ▶ And such a value is not necessary because of the top-down traversing of the edge which guarantees that only nodes that are on the best path will ever consider for expansion.

1.3 UNIFORMED SEARCH

1.3.1 General Search Algorithm

Q18. What is search problem ? Write generic search algorithm.

Ans :

Defining a Search Problem

- ▶ State space S : all possible configurations of the domain of interest
- ▶ An initial (start) state $s_0 \in S$
- ▶ Goal states $G \subset S$: the set of end states
 - Often defined by a goal test rather than enumerating a set of states
- ▶ Operators A : the actions available
 - Often defined in terms of a mapping from a state to its successor
- ▶ Path : a sequence of states and operators
- ▶ Path cost : a number associated with any path
 - Measures the quality of the path
 - Usually the smaller, the better
- ▶ Solution of a search problem is a path from s_0 to some $sg \in G$
- ▶ Optimal solution: any path with minimum cost.

Generic Search Algorithm

1. Initialize the search tree using the initial state of the problem
2. Repeat
 - a) If no candidate nodes can be expanded, return failure
 - b) Choose a leaf node for expansion, according to some search strategy
 - c) If the node contains a goal state, return the corresponding path
 - d) Otherwise expand the node by:
 - ▶ Applying each operator
 - ▶ Generating the successor state
 - ▶ Adding the resulting nodes to the tree.

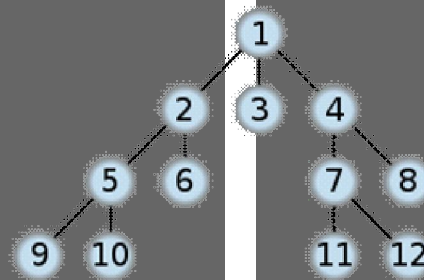
1.3.2 Uniformed Search Methods - BFS

Q19. Explain BFS method.

Ans :

Description

- ▶ A simple strategy in which the root is expanded first then all the root successors are expanded next, then their successors.
- ▶ We visit the search tree level by level that all nodes are expanded at a given depth before any nodes at the next level are expanded.
- ▶ Order in which nodes are expanded.



Performance Measure

Completeness

- ▶ it is easy to see that breadth-first search is complete that it visit all levels given that d factor is finite, so in some d it will find a solution.

Optimality:

- ▶ breadth-first search is not optimal until all actions have the same cost.

Space complexity and Time complexity:

- ▶ Consider a state space where each node as a branching factor b , the root of the tree generates b nodes, each of which generates b nodes yielding b^2 each of these generates b^3 and so on.

- ▶ In the worst case, suppose that our solution is at depth d , and we expand all nodes but the last node at level d , then the total number of generated nodes is: $b + b^2 + b^3 + b^4 + b^{d+1} - b = O(b^{d+1})$, which is the time complexity of BFS.
- ▶ As all the nodes must retain in memory while we expand our search, then the space complexity is like the time complexity plus the root node = $O(b^{d+1})$.

Conclusion:

- ▶ We see that space complexity is the biggest problem for BFS than its exponential execution time.
- ▶ Time complexity is still a major problem, to convince your-self look at the table below.

Depth	Nodes	Time	Memory
2	1100	.11 seconds	1 megabyte
4	111.100	11 seconds	106 megabytes
6	10^7	19 minutes	10 gigabytes
8	10^9	31 hours	1 terabytes
10	10^{11}	129 days	101 terabytes
12	10^{13}	35 years	10 petabytes
14	10^{15}	3,523 years	1 exabyte

Figure: Time and memory requirements for breadth - first search. The numbers shown assume branching factor $b = 10$; 10,000 nodes/second; 1000 bytes/node.

1.3.3 Uniform Cost Search

Q20. Explain Uniform Cost search.

Ans :

Description

- ▶ Uniform-cost is guided by path cost rather than path length like in BFS, the algorithms starts by expanding the root, then expanding the node with the lowest cost from the root, the search continues in this manner for all nodes.
- ▶ Hints about UCS implementation can be found [here](#).
- ▶ You should not be surprised that Dijkstra's algorithm, which is perhaps better-known, can be regarded as a variant of uniform-cost search, where there is no goal state and processing continues until the shortest path to all nodes has been determined.

Performance Measure

Completeness

It is obvious that UCS is complete if the cost of each step exceeds some small positive integer, this to prevent infinite loops.

Optimality

UCS is always optimal in the sense that the node that it always expands is the node with the least path cost.

Time Complexity

UCS is guided by path cost rather than path length so it is hard to determine its complexity in terms of b and d , so if we consider C to be the cost of the optimal solution, and every action costs at least e , then the algorithm worst case is $O(b^{C/e})$.

Space Complexity

The space complexity is $O(b^{C/e})$ as the time complexity of UCS.

Conclusion

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

1.3.4 DFS

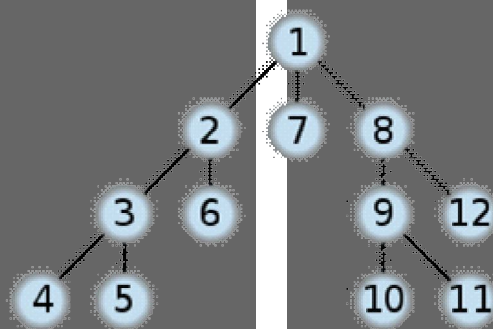
Q21. Explain DFS Search Method.

Ans :

Description

DFS progresses by expanding the first child node of the search tree that appears and thus going deeper and deeper until a goal node is found, or until it hits a node that has no children. Then the search backtracks, returning to the most recent node it hasn't finished exploring.

Order in which nodes are expanded.



Performance Measure :

Completeness

DFS is not complete, to convince yourself consider that our search start expanding the left sub tree of the root for so long path (may be infinite) when different choice near the root could lead to a solution, now suppose that the left sub tree of the root has no solution, and it is unbounded, then the search will continue going deep infinitely, in this case we say that DFS is not complete.

Optimality

Consider the scenario that there is more than one goal node, and our search decided to first expand the left sub tree of the root where there is a solution at a very deep level of this left sub tree, in the same time the right sub tree of the root has a solution near the root, here comes the non-optimality of DFS that it is not guaranteed that the first goal to find is the optimal one, so we conclude that DFS is not optimal.

Time Complexity

Consider a state space that is identical to that of BFS, with branching factor b , and we start the search from the root.

In the worst case that goal will be in the shallowest level in the search tree resulting in generating all tree nodes which are $O(b^m)$.

Space Complexity

Unlike BFS, our DFS has a very modest memory requirements, it needs to store only the path from the root to the leaf node, beside the siblings of each node on the path, remember that BFS needs to store all the explored nodes in memory.

DFS removes a node from memory once all of its descendants have been expanded.

With branching factor b and maximum depth m , DFS requires storage of only $bm + 1$ nodes which are $O(bm)$ compared to the $O(b^{d+1})$ of the BFS.

Conclusion

DFS may suffer from non-termination when the length of a path in the search tree is infinite, so we perform DFS to a limited depth which is called Depth-limited Search.

1.3.5 DLS

Q22. Write about depth limit search.

Ans :

Description

The unbounded tree problem appeared in DFS can be fixed by imposing a limit on the depth that DFS can reach, this limit we will call depth limit l , this solves the infinite path problem.

Performance Measure

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

Optimality : One can view DFS as a special case of the depth DLS, that DFS is DLS with $l = \text{infinity}$.

DLS is not optimal even if $l > d$.

Time Complexity : $O(b^l)$

Space Complexity : $O(b^l)$

Conclusion : DLS can be used when there is a prior knowledge to the problem, which is always not the case, Typically, we will not know the depth of the shallowest goal of a problem unless we solved this problem before.

1.3.6 IS

Q23. Explain Iterative deepening depth-first search (IDS).*Ans :*

Iterative deepening depth-first search (IDS).

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

IDS works by looking for the best search depth d , thus starting with depth limit 0 and make a BFS and if the search failed it increase the depth limit by 1 and try a BFS again with depth 1 and so on – first $d = 0$, then 1 then 2 and so on – until a depth d is reached where a goal is found.

Performance Measure :

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

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

Time Complexity : One may find that it is wasteful to generate nodes multiple times, but actually it is not that costly compared to BFS, that is because most of the generated nodes are always in the deepest level reached, consider that we are searching a binary tree and our depth limit reached 4, the nodes generated in last level = $2^4 = 16$, the nodes generated in all nodes before last level = $2^0 + 2^1 + 2^2 + 2^3 = 15$

Imagine this scenario, we are performing IDS and the depth limit reached depth d , now if you remember the way IDS expands nodes, you can see that nodes at depth d are generated once, nodes at depth $d-1$ are generated 2 times, nodes at depth $d-2$ are generated 3 times and so on, until you reach depth 1 which is generated d times, we can view the total number of generated nodes in the worst case as :

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

If this search were to be done with BFS, the total number of generated nodes in the worst case will be like :

$$N(\text{BFS}) = b + b^2 + b^3 + b^4 + \dots b^d + (b^{d+1} - b) = O(b^{d+1})$$

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

- ▶ $N(\text{IDS}) = 50 + 400 + 3000 + 20000 + 100000 = 123450$
- ▶ $N(\text{BFS}) = 10 + 100 + 1000 + 10000 + 100000 + 999990 = 1111100$

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

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

Conclusion : We can conclude that IDS is a hybrid search strategy between BFS and DFS inheriting their advantages.

IDS is faster than BFS and DFS

It is said that "IDS is the preferred uniformed search method when there is a large search space and the depth of the solution is not known".

1.3.7 Bi-Directional Search

Q24. Explain about bi- directional search.

Ans :

Bidirectional Search

Description :

As the name suggests, bidirectional search suggests to run 2 simultaneous searches, one from the initial state and the other from the goal state, those 2 searches stop when they meet each other at some point in the middle of the graph.

The following pictures illustrates a bidirectional search :

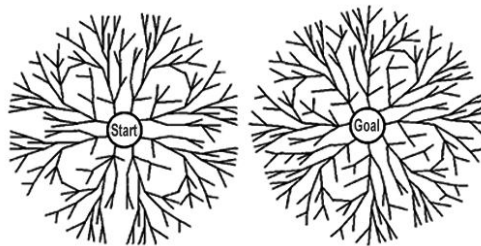


Fig.: Aschematic view of a bidirectional search that is about to succeed, when a branch from the start node meets a branch from the goal node.

Performance Measure

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

Optimality : Like the completeness, bidirectional search is optimal when BFS is used and paths are of a uniform cost – all steps of the same cost.

Other search strategies can be used like DFS, but this will sacrifice the optimality and completeness, any other combination than BFS may lead to a sacrifice in optimality or completeness or may be both of them.

Time and Space Complexity : May be the most attractive thing in bidirectional search is its performance, because both searches will run the same amount of time meeting in the middle of the graph, thus each search expands $O(b^{d/2})$ node, in total both searches expand $O(b^{d/2} + b^{d/2})$ node which is too far better than the $O(b^d + 1)$ of BFS.

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

We can say that the time and space complexity of bidirectional search is $O(b^{d/2})$.

Conclusion : Bidirectional search seems attractive for its $O(b^{d/2})$ performance, but things are not that easy, especially the implementation part.

It is not that easy to formulate a problem such that each state can be reversed, that is going from the head to the tail is like going from the tail to the head.

It should be efficient to compute the predecessor of any state so that we can run the search from the goal.

1.3.8 Comparison of the Uniformed Techniques

Q25. Compare between the search strategies.

Ans :

Search Strategies' Comparison:

Here is a table that compares the performance measures of each search strategy.

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes ^a	Yes ^{a,b}	No	No	Yes ^a	Yes ^{a,d}
Time	$O(b^{d+1})$	$O(b^{1+\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^{d+1})$	$O(b^{1+\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes ^c	Yes	No	No	Yes ^c	Yes ^{c,d}

Figure: Evaluation of search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: ^a complete if b is finite; ^b complete if step costs $\geq \epsilon$ for positive ϵ ; ^c optimal if step costs are all identical; ^d if both directions use breadth-first search.

UNIT II

Informed Search: generate and test, best first search, greedy search, A* search, memory bounded heuristic search, heuristic function, AO* search, local search algorithms and optimization problems, adversarial search methods (game theory), online search algorithms.

What is an intelligent agent? Types of agent, what is constraint satisfaction problem (CSP), CSP as search problem, local search for CSP, formulating problem structure. Knowledge and Reasoning: knowledge representation, knowledge-based agents, the wumpus world, logic, propositional logic, predicate logic, unification and lifting: inference in FOL, representing knowledge using rules, semantic networks, frame systems, inference, types of reasoning.

2.1 INFORMED SEARCH

2.1.1 General and Test

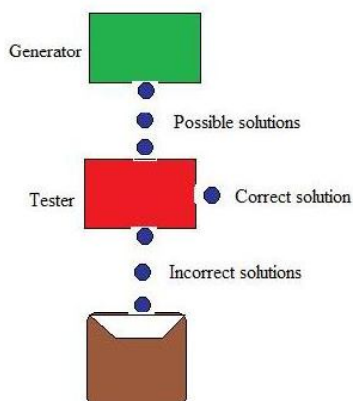
Q1. Write about generate and test strategy.

Ans :

This is the simplest search strategy. It consists of the following steps;

1. Generating a possible solution for some problems; this means generating a particular point in the problem space. For others it may be generating a path from a start state.
2. Test to see if this is actually a solution by comparing the chosen point at the end point of the chosen path to the set of acceptable goal states.
3. If a solution has been found, quit otherwise return to step 1.

The generate - and - Test algorithm is a depth first search procedure because complete possible solutions are generated before test. This can be implemented states are likely to appear often in a tree; it can be implemented on a search graph rather than a tree.



2.1.2 Best First Search

Q2. Write a short note on Best first research?

Ans :

Best-first search, rather than plunging as deep as possible into the tree (as indepth-first search), or traversing each level of the tree in succession (as inbreadth-first search), uses a heuristic to decide at each stage which is the best place to continue the search.

Best-first search in its most basic form consists of the following algorithm:

The first step is to define the OPEN list with a single node, the starting node. The second step is to check whether or not OPEN is empty. If it is empty, then the algorithm returns failure and exits. The third step is to remove the node with the best score, n , from OPEN and place it in CLOSED.

The fourth step "expands" the node n , where expansion is the identification of successor nodes of n . The fifth step then checks each of the successor nodes to see whether of not one of them is the goal node. If any successor is the goal node, the algorithm returns success and the solution, which consists of a path traced backwards from the goal to the start node. Otherwise, the algorithm proceeds to the sixth step.

For every successor node, the algorithm applies the evaluation function, f , to it, then checks to see if the node has been in either OPEN or CLOSED. If the node has not been in either, it gets added to OPEN. Finally, the seventh step establishes a looping structure by sending the algorithm back to the second step. This loop will only be broken if the

algorithm returns success in step five or failure in step two.

The algorithm is represented here in pseudo-code :

1. Define a list, OPEN, consisting solely of a single node, the start node, s.
2. IF the list is empty, return failure.
3. Remove from the list the node n with the best score (the node where f is the minimum), and move it to a list, CLOSED.
4. Expand node n.
5. IF any successor to n is the goal node, return success and the solution (by tracing the path from the goal node to s).
6. FOR each successor node:
 - a) apply the evaluation function, f, to the node.
 - b) IF the node has not been in either list, add it to OPEN.
7. GOTO 2.

Applications

Best-first search and its more advanced variants have been used in such applications as games and web crawlers. In a web crawler, each web page is treated as a node, and all the hyperlinks on the page are treated as unvisited successor nodes. A crawler that uses best-first search generally uses an evaluation function that assigns priority to links based on how closely the contents of their parent page resemble the search query. In games, best-first search may be used as a path-finding algorithm for game characters.

For example, it could be used by an enemy agent to find the location of the player in the game world. Some games divide up the terrain into "tiles" which can either be blocked or unblocked. In such cases, the search algorithm treats each tile as a node, with the neighbouring unblocked tiles being successor nodes, and the goal node being the destination tile.

2.1.3 Greedy Search

Q3. What is greedy search ?

Ans :

A greedy algorithm, as the name suggests, **always makes the choice that seems to be the best at that moment**. This means that it makes a locally-optimal choice in the hope that this choice will lead to a globally-optimal solution.

How do you decide which choice is optimal?

Assume that you have an **objective function** that needs to be optimized (either maximized or minimized) at a given point. A Greedy algorithm makes greedy choices at each step to ensure that the objective function is optimized. The Greedy algorithm has only one shot to compute the optimal solution so that **it never goes back and reverses the decision**.

Greedy algorithms have some advantages and disadvantages :

1. It is quite easy to **come up with a greedy algorithm** (or even multiple greedy algorithms) for a problem.
2. **Analyzing the run time for greedy algorithms will generally be much easier** than for other techniques (like Divide and conquer). For the Divide and conquer technique, it is not clear whether the technique is fast or slow. This is because at each level of recursion the size of gets smaller and the number of sub-problems increases.
3. The difficult part is that for greedy algorithms **you have to work much harder to understand correctness issues**. Even with the correct algorithm, it is hard to prove why it is correct. Proving that a greedy algorithm is correct is more of an art than a science. It involves a lot of creativity.

2.1.4 A* Search

Q4. Explain A* Algorithm.

Ans :

THE A* ALGORITHM:-

1. Start with OPEN containing the initial node. Its $g=0$ and $f = h$

Set CLOSED to empty list.

2. Repeat

If OPEN is empty, stop and return failure

Else pick the BESTNODE on OPEN with lowest f' value and place it on CLOSED

If BESTNODE is goal state return success and stop

Else

Generate the successors of BESTNODE.

For each SUCCESSOR do the following:

1. Set SUCCESSOR to point back to BESTNODE. (back links will help to recover the path)
2. compute $g(\text{SUCCESSOR}) = g(\text{BESTNODE})$ cost of getting from BESTNODE to SUCCESSOR.
3. If SUCCESSOR is the same as any node on OPEN, call that node OLD and add OLD to BESTNODE's successors. Check $g(\text{OLD})$ and $g(\text{SUCCESSOR})$. If $g(\text{SUCCESSOR})$ is cheaper then reset OLD's parent link to point to BESTNODE. Update $g(\text{OLD})$ and $f'(\text{OLD})$.
4. If SUCCESSOR was not on OPEN, see if it is on CLOSED. If so call the node CLOSED OLD, and better as earlier and set the parent link and g and f' values appropriately.
5. If SUCCESSOR was not already on earlier OPEN or CLOSED, then put it on OPEN and add it to the list of BESTNODE's successors.

Compute $f'(\text{SUCCESSOR}) = g(\text{SUCCESSOR}) + h'(\text{SUCCESSOR})$

Best first searches will always find good paths to a goal after exploring the entire state space. All that is required is that a good measure of goal distance be used.

2.1.5 Memory Bounded Heuristic Search

Q5. What is memory bounded heuristic search?

Ans :

The Simplified Memory-Bounded Algorithm (SMA*) is a variant of A* search which is memory-bounded.

Complete & optimal if enough memory is available to store shallowest solution path.

To enqueue new expand, dequeue the unpromising node with highest f -cost.

Retains ancestor node information about the quality of the best path in the forgotten sub tree.

- To reduce memory- Iterative deepening to the heuristic search.
- 2 memory bounded algorithm:
 - 1) RBFS (recursive best-first search).
 - 2) MA* (Memory-bounded A*) and SMA*(simplified memory MA*)

RBFS :

- It attempts to mimic the operation of BFS.
- It replaces the f -value of each node along the path with the best f -value of its children.
- Suffers from **using too little memory**.
- Even if more memory were available, RBFS has no way to make use of it.

SMA*

- Proceeds like A*, expands best leaf until memory is full.
- Cannot add new node without dropping an old one. (always drops worst one)
- **Expands the best leaf and deletes the worst leaf.**
- If all have same f -value-selects same node for expansion and deletion.
- SMA* is complete if any reachable solution.

2.1.6 Heuristic Function

Q6. What is heuristic function? Explain with an example.

Ans :

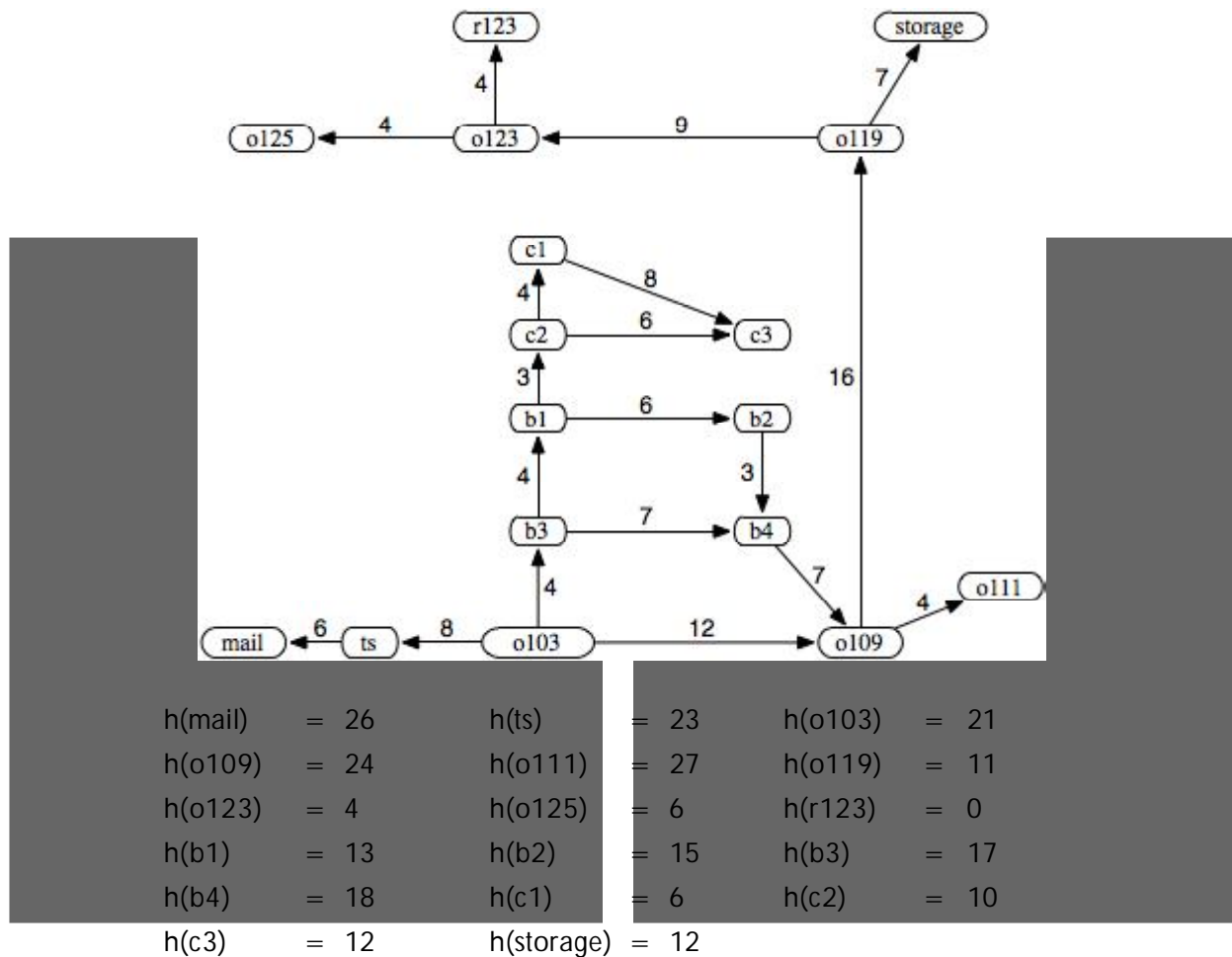
The heuristic function is a way to inform the search about the direction to a goal. It provides an informed way to guess which neighbor of a node will lead to a goal.

There is nothing magical about a heuristic function. It must use only information that can be

readily obtained about a node. Typically a trade-off exists between the amount of work it takes to derive a heuristic value for a node and how accurately the heuristic value of a node measures the actual path cost from the node to a goal.

A standard way to derive a heuristic function is to solve a simpler problem and to use the actual cost in the simplified problem as the heuristic function of the original problem.

Example : For the graph of given figure the straight-line distance in the world between the node and the goal position can be used as the heuristic function. The examples that follow assume the following heuristic function :



This h function is an underestimate because the h value is less than or equal to the exact cost of a lowest-cost path from the node to a goal. It is the exact cost for node o123 . It is very much an underestimate for node b1 , which seems to be close, but there is only a long route to the goal. It is very misleading for c1 , which also seems close to the goal, but no path exists from that node to the goal.

2.1.7 AO* Search

Q7. Explain AO* Algorithm.

Ans :

AO* ALGORITHM

1. Let G consists only to the node representing the initial state call this node INTT. Compute h' (INIT).

2. Until INIT is labeled SOLVED or h_i (INIT) becomes greater than FUTILITY, repeat the following procedure.
 - I) Trace the marked arcs from INIT and select an unbounded node NODE.
 - II) Generate the successors of NODE. if there are no successors then assign FUTILITY as h' (NODE). This means that NODE is not solvable. If there are successors then for each one called SUCCESSOR, that is not also an ancestor of NODE do the following :
 - a) add SUCCESSOR to graph G
 - b) if successor is not a terminal node, mark it solved and assign zero to its h' value.
 - c) If successor is not a terminal node, compute its h' value.
 - III) propagate the newly discovered information up the graph by doing the following. Let S be a set of nodes that have been marked SOLVED. Initialize S to NODE. Until S is empty repeat the following procedure;
 - a) select a node from S call it CURRENT and remove it from S.
 - b) compute h' of each of the arcs emerging from CURRENT, Assign minimum h' to CURRENT.
 - c) Mark the minimum cost path as the best out of CURRENT.
 - d) Mark CURRENT SOLVED if all of the nodes connected to it through the new marked arcs have been labelled SOLVED.
 - e) If CURRENT has been marked SOLVED or its h' has just changed, its new status must be propagated backwards up the graph. Hence all the ancestors of CURRENT are added to S.

AO* Search Procedure

1. Place the start node on open.
2. Using the search tree, compute the most promising solution tree TP.
3. Select node n that is both on open and a part of tp, remove n from open and place it in closed.

4. If n is a goal node, label n as solved. If the start node is solved, exit with success where tp is the solution tree, remove all nodes from open with a solved ancestor.
5. If n is not solvable node, label n as unsolvable. If the start node is labeled as unsolvable, exit with failure. Remove all nodes from open, with unsolvable ancestors.
6. Otherwise, expand node n generating all of its successors compute the cost of for each newly generated node and place all such nodes on open.
7. Go back to step(2)

Note: AO* will always find minimum cost solution.

2.1.8 Local Search Algorithms and Optimization Problems

Q8. Explain Hill Climbing Search algorithm for optimization problems.

Ans :

Hill Climbing Search

This is a variety of depth-first (generate - and - test) search. A feedback is used here to decide on the direction of motion in the search space. In the depth-first search, the test function will merely accept or reject a solution. But in hill climbing the test function is provided with a heuristic function which provides an estimate of how close a given state is to goal state. The hill climbing test procedure is as follows :

1. Generate the first proposed solution as done in depth-first procedure. See if it is a solution. If so quit, else continue.
2. From this solution generate new set of solutions use, some application rules
3. For each element of this set
 - i) Apply test function. It is a solution quit.
 - ii) Else see whether it is closer to the goal state than the solution already generated.

If yes, remember it else discard it.
4. Take the best element so far generated and use it as the next proposed solution.

This step corresponds to move through the problem space in the direction towards the goal state.

5. Go back to step 2.

Sometimes this procedure may lead to a position, which is not a solution, but from which there is no move that improves things. This will happen if we have reached one of the following three states.

- a) A "local maximum " which is a state better than all its neighbors , but is not better than some other states farther away. Local maxim sometimes occur with in sight of a solution. In such cases they are called " Foothills".
- b) A "plateau" which is a flat area of the search space, in which neighboring states have the same value. On a plateau, it is not possible to determine the best direction in which to move by making local comparisons.
- c) A "ridge" which is an area in the search that is higher than the surrounding areas, but can not be searched in a simple move.

To overcome theses problems we can

- a) Back track to some earlier nodes and try a different direction. This is a good way of dealing with local maxim.
- b) Make a big jump an some direction to a new area in the search. This can be done by applying two more rules of the same rule several times, before testing. This is a good strategy is dealing with plate and ridges.

Hill climbing becomes inefficient in large problem spaces, and when combinatorial explosion occurs. But it is a useful when combined with other methods.

Q9. What is Simulated Annealing ?

Ans :

- **Idea :** escape local maxima by allowing some "bad" moves but gradually decrease their frequency.
function SIMULATED-ANNEALING (problem, schedule) returns a solution state
 inputs : problem, a problem
 schedule, a mapping from time to "temperature"
 local variables : current, a node
 next, a node

T, a "temperature" controlling prob. of downward steps

currents \leftarrow MAKE-NODE(LNITIAL-STATE[problem])

for $t \leftarrow 1$ to ∞ do

$T \leftarrow \text{schedxde}[t]$

 if $T = 0$ then return current

 next \leftarrow a randomly selected successor of current

$\Delta E \leftarrow \text{VALUE}[\text{next}] - \text{VALUE}[\text{current}]$

 if $\Delta E > 0$ then current \leftarrow next

 else current \leftarrow next only with probability $e^{\Delta E/T}$

Properties of simulated annealing Search

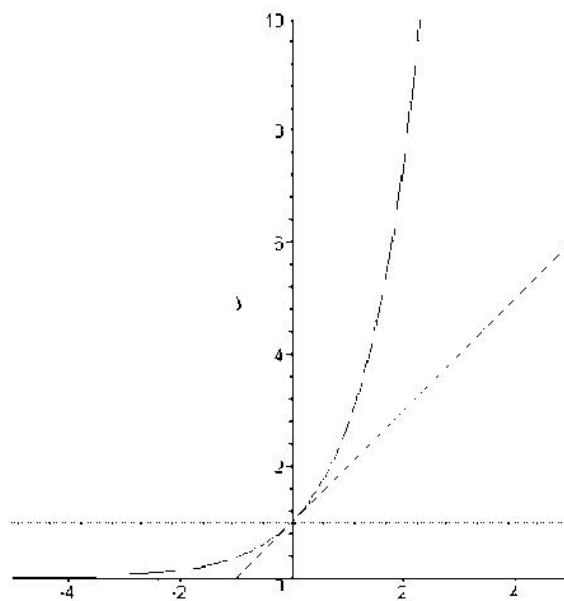
The parameter $e^{\frac{\Delta E}{T}}$ is the key idea.

The graph for $x = \frac{\Delta E}{T}$ is given by the exponential function (see graph)

$$x < 0 \Rightarrow f(x) < 1 \wedge f(x) > 0$$

One can prove : If T decreases slowly enough, then simulated annealing search will find a global optimum with probability approaching 1

Widely used in VLSI layout, airline scheduling, etc.,



2.1.9 Adversarial Search Methods (Game Theory)

Q10. Explain Mini-Max Algorithm with Alpha beta pruning.

Ans :

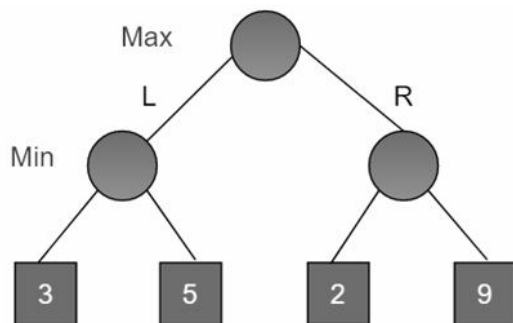
Minimax Algorithm in Game Theory

Minimax is a kind of backtracking algorithm that is used in decision making and game theory to find the optimal move for a player, assuming that your opponent also plays optimally. It is widely used in two player turn based games such as Tic-Tac-Toe, Backgamon, Mancala, Chess, etc.

In Minimax the two players are called maximizer and minimizer. The **maximizer** tries to get the highest score possible while the **minimizer** tries to get the lowest score possible while minimizer tries to do opposite.

Example :

Consider a game which has 4 final states and paths to reach final state are from root to 4 leaves of a perfect binary tree as shown below. Assume you are the maximizing player and you get the first chance to move, i.e., you are at root, and your opponent at next level.



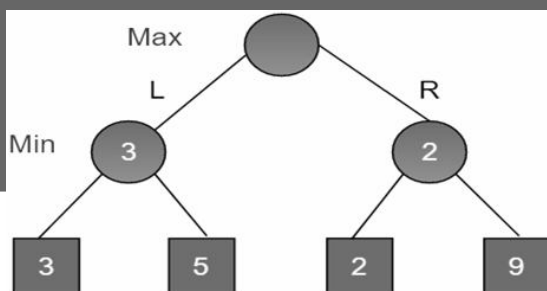
Since this is a backtracking based algorithm, it tries all possible moves, then backtracks and makes a decision.

► **Maximizer goes LEFT :** It is now the minimizers turn. The minimizer now has a choice between 3 and 5. Being the minimizer it will definitely choose the least among both, that is 3.

► **Maximizer goes RIGHT :** It is now the minimizers turn. The minimizer now has a choice between 2 and 9. He will choose 2 as it is the least among the two values.

Being the maximizer you would choose the larger value that is 3. Hence the optimal move for the maximizer is to go LEFT and the optimal value is 3.

Now the game tree looks like below.



The above tree shows two possible scores when maximizer makes left and right moves.

Q11. What is alpha-beta pruning? Explain.

Ans :

ALPHA-BETA pruning

ALPHA-BETA pruning is a method that reduces the number of nodes explored in Minimax strategy.

It reduces the time required for the search and it must be restricted so that no time is to be wasted searching moves that are obviously bad for the current player.

The exact implementation of alpha-beta keeps track of the best move for each side as it moves throughout the tree.

We proceed in the same (preorder) way as for the minimax algorithm. For the MIN nodes, the score computed starts with infinity and decreases with time. For MAX nodes, scores computed starts with -infinity and increase with time.

The efficiency of the Alpha-Beta procedure depends on the order in which successors of a node are examined. If we were lucky, at a MIN node we would always consider the nodes in order from low to high score and at a MAX node the nodes in order from high to low score. In general it can be shown that in the most favorable circumstances the alpha-beta search opens as many leaves as minimax on a game tree with double its depth.

Here is an example of Alpha-Beta search

Alpha-Beta algorithm : The algorithm maintains two values, alpha and beta, which represent the minimum score that the maximizing player is assured of and the maximum score that the minimizing player is assured of respectively. Initially alpha is negative infinity and beta is positive infinity. As the recursion progresses the "window" becomes smaller.

When beta becomes less than alpha, it means that the current position cannot be the result of best play by both players and hence need not be explored further.

Pseudo code for the alpha-beta algorithm is given below.

```

evaluate (node, alpha, beta)
if node is a leaf
return the heuristic value of node
if node is a minimizing node
for each child of node
beta = min (beta, evaluate (child, alpha, beta))
if beta <= alpha
return beta
return beta
  
```

```

if node is a maximizing node
for each child of node
alpha = max (alpha, evaluate (child, alpha, beta))
if beta <= alpha
return alpha
return alpha.

```

2.1.10 Online Search Algorithms

Q12. What is online algorithm? Write in detail about it.

Ans :

- ▶ Suppose, a robot does not have the complete knowledge of the geometry of R apriori.
- ▶ The robot also does not know the location of the target t, but the target can be recognized by the robot.
- ▶ In such a situation, the robot is asked to reach t from its starting position is using its sensory input provided by acoustic, visual, or tactile sensors of its on-board sensor system.
- ▶ The problem here is to design an efficient online algorithm which a robot can use to search for the target t.
- ▶ Observe that any such algorithm is 'online' in the sense that decisions must be made based only on what the robot has received input so far from its sensor system.

Efficiency of Online Algorithms

One of the difficulties in working with incomplete information is that the path cannot be pre-planned and therefore, its global optimality can hardly be achieved.

Instead, one can judge the online algorithm performance based on how it stands with respect to other existing or theoretically feasible algorithms.

The efficiency of online algorithms for searching and exploration algorithms is generally measured using their competitive ratios.

Competitive ratio = Cost of the online algorithm

Cost of an optimal offline algorithm.

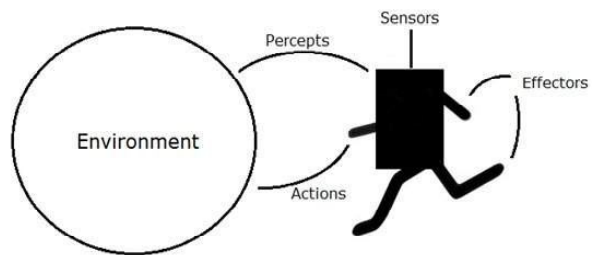
2.2 WHAT IS AN INTELLIGENT AGENT

Q13. What is an intelligent Agent ?

Ans :

An **agent** is anything that can perceive its environment through **sensors** and acts upon that environment through **effectors**.

- ▶ A **human agent** has sensory organs such as eyes, ears, nose, tongue and skin parallel to the sensors, and other organs such as hands, legs, mouth, for effectors.
- ▶ A **robotic agent** replaces cameras and infrared range finders for the sensors, and various motors and actuators for effectors.
- ▶ A **software agent** has encoded bit strings as its programs and actions.



Agents are designed to make computing easier. Currently they are used as Web browsers, news retrieval mechanisms, and shopping assistants. By specifying certain parameters, agents will "search" the Internet and return the results directly back to your PC.

2.2.1 Types of Agent

Q14. What are the various types of agents? Write about them ?

Ans :

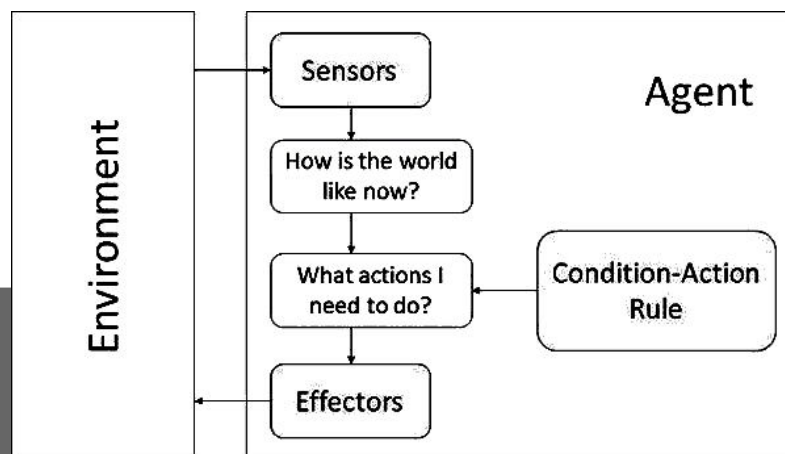
Agent's structure can be viewed as :

- ▶ Agent = Architecture + Agent Program
- ▶ Architecture = the machinery that an agent executes on.
- ▶ Agent Program = an implementation of an agent function.

Simple Reflex Agents

- ▶ They choose actions only based on the current percept.
- ▶ They are rational only if a correct decision is made only on the basis of current percept.
- ▶ Their environment is completely observable.

Condition-Action Rule - It is a rule that maps a state (condition) to an action.



Model Based Reflex Agents

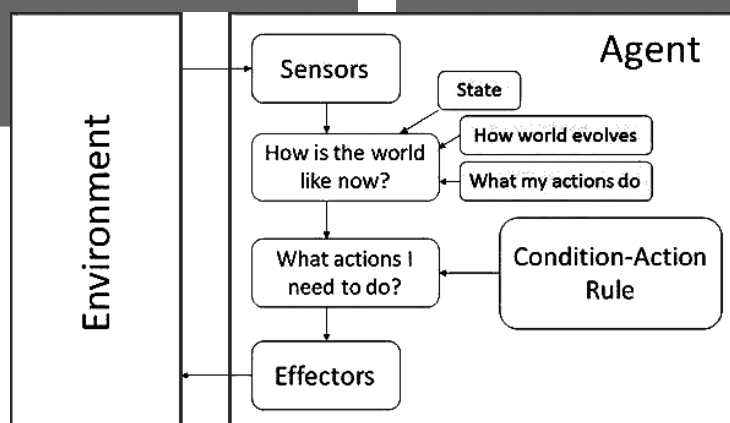
They use a model of the world to choose their actions. They maintain an internal state.

Model - The knowledge about "how the things happen in the world".

Internal State - It is a representation of unobserved aspects of current state depending on percept history.

Updating the state requires the information about "

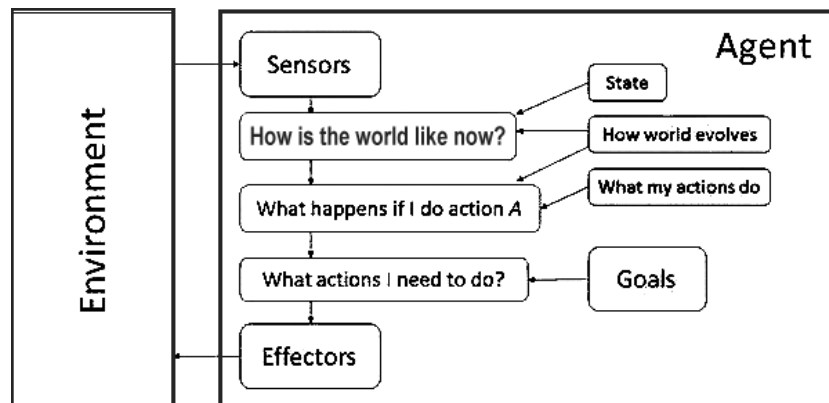
- ▶ How the world evolves.
- ▶ How the agent's actions affect the world.



Goal Based Agents

They choose their actions in order to achieve goals. Goal-based approach is more flexible than reflex agent since the knowledge supporting a decision is explicitly modeled, thereby allowing for modifications.

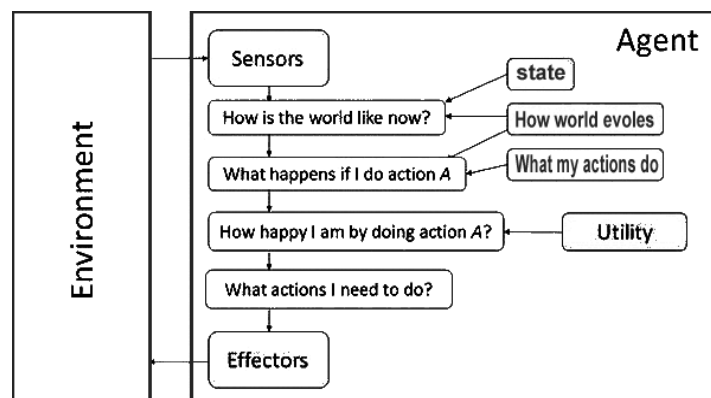
Goal - It is the description of desirable situations.



Utility Based Agents

They choose actions based on a preference (utility) for each state. Goals are inadequate when -

- ▶ There are conflicting goals, out of which only few can be achieved.
- ▶ Goals have some uncertainty of being achieved and you need to weigh likelihood of success against the importance of a goal.



2.2.2 What is Constraint Satisfaction Problem (CSP)

Q15. What is constraint satisfaction problem? Explain it with an example.

Or

Explain CSP with the example of $SEND + MORE = MONEY$?

Ans :

A **constraint satisfaction problem** (CSP) consists of

- ▶ a set of variables,
- ▶ a domain for each variable, and
- ▶ a set of constraints.

The aim is to choose a value for each variable so that the resulting possible world satisfies the constraints; we want a model of the constraints.

A finite CSP has a finite set of variables and a finite domain for each variable. Many of the methods considered here are only work for finite CSPs, although some are designed for infinite, even continuous, domains.

The multidimensional aspect of these problems, where each variable can be seen as a separate dimension, makes them difficult to solve but also provides structure that can be exploited.

Given a CSP, there are a number of tasks that can be performed :

- ▶ Determine whether or not there is a model.
- ▶ Find a model.
- ▶ Find all of the models or enumerate the models.
- ▶ Count the number of models.
- ▶ Find the best model, given a measure of how good models are;
- ▶ Determine whether some statement holds in all models.

Constraint Satisfaction Problem1

Many problems in AI can be considered as problems of constraint satisfaction, in which the goal state satisfies a given set of constraint. constraint satisfaction problems can be solved by using any of the search strategies. The general form of the constraint satisfaction procedure is as follows:

Until a complete solution is found or until all paths have led to lead ends, do

1. Select an unexpanded node of the search graph.
2. Apply the constraint inference rules to the selected node to generate all possible new constraints.
3. If the set of constraints contains a contradiction, then report that this path is a dead end.
4. If the set of constraints describes a complete solution then report success.
5. If neither a constraint nor a complete solution has been found then apply the rules to generate new partial solutions. Insert these partial solutions into the search graph.

Example: consider the crypt arithmetic problems.

```

SEND
+ MORE
-----
MONEY
-----

```

Assign decimal digit to each of the letters in such a way that the answer to the problem is correct to the same letter occurs more than once , it must be assign the same digit each time . no two different letters may be assigned the same digit. Consider the crypt arithmetic problem.

```

SEND
+ MORE
-----
MONEY
-----

```

Constraints :

1. no two digit can be assigned to same letter.
2. only single digit number can be assign to a letter.
3. no two letters can be assigned same digit.
4. Assumption can be made at various levels such that they do not contradict each other.
5. The problem can be decomposed into secured constraints. A constraint satisfaction approach may be used.
6. Any of search techniques may be used.
7. Backtracking may be performed as applicable us applied search techniques.
8. Rule of arithmetic may be followed.

Initial State of Problem

D=?
 E=?
 Y=?
 N=?
 R=?
 O=?
 S=?
 M=?

C1=?

C2=?

C1, C2, C3 stands for the carry variables respectively.

Goal State : The digits to the letters must be assigned in such a manner so that the sum is satisfied.

Solution Process :

We are following the depth-first method to solve the problem.

1. initial guess $m=1$ because the sum of two single digits can generate at most a carry '1'.
2. When $n=1$ $o=0$ or 1 because the largest single digit number added to $m=1$ can generate the sum of either 0 or 1 depend on the carry received from the carry sum. By this we conclude that $o=0$ because m is already 1 hence we cannot assign same digit another letter(rule no.)
3. We have $m=1$ and $o=0$ to get $o=0$ we have $s=8$ or 9 , again depending on the carry received from the earlier sum.

The same process can be repeated further. The problem has to be composed into various constraints.

Solution :

$$\begin{array}{r}
 9 \quad 5 \quad 6 \quad 7 \\
 + \quad 1 \quad 0 \quad 8 \quad 5 \\
 \hline
 1 \quad 0 \quad 6 \quad 5 \quad 2
 \end{array}$$

Values :

S = 9

E = 5

N = 6

D = 7

M = 1

O = 0

R = 8

Y = 2

2.2.3 CSP as Search Problem

Solving CSPs Using Search

Q16. Explain How to solve Constraint Satisfaction Problems using Search ?

Ans :

Generate-and-test algorithms assign values to all variables before checking the constraints. Because individual constraints only involve a subset of the variables, some constraints can be tested before all of the variables have been assigned values. If a partial assignment is inconsistent with a constraint, any complete assignment that extends the partial assignment will also be inconsistent.

Example

In the delivery scheduling problem the assignments $A=1$ and $B=1$ are inconsistent with the constraint $A=B$ regardless of the values of the other variables. If the variables A and B are assigned values first, this inconsistency can be discovered before any values are assigned to C , D , or E , thus saving a large amount of work.

An alternative to generate-and-test algorithms is to construct a search space. The search problem can be defined as follows :

- The nodes are assignments of values to some subset of the variables.
- The neighbors of a node N are obtained by selecting a variable V that is not assigned in node N and by having a neighbor for each assignment of a value to V that does not violate any constraint.

Suppose that node N represents the assignment $X_1=v_1, \dots, X_k=v_k$. To find the neighbors of N , select a variable Y that is not in the set $\{X_1, \dots, X_k\}$. For each value $y_i \in \text{dom}(Y)$, such that $X_1=v_1, \dots, X_k=v_k, Y=y_i$ is consistent with the constraints, $X_1=v_1, \dots, X_k=v_k, Y=y_i$ is a neighbor of N .

- The start node is the empty assignment that does not assign a value to any variables.
- A goal node is a node that assigns a value to every variable. Note that this only exists if the assignment is consistent with the constraints.

In this case, it is not the path that is of interest, but the goal nodes.

Example : Suppose you have a CSP with the variables A, B, and C, each with domain {1,2,3,4}. Suppose the constraints are $A < B$ and $B < C$. A possible search tree is shown in figure.

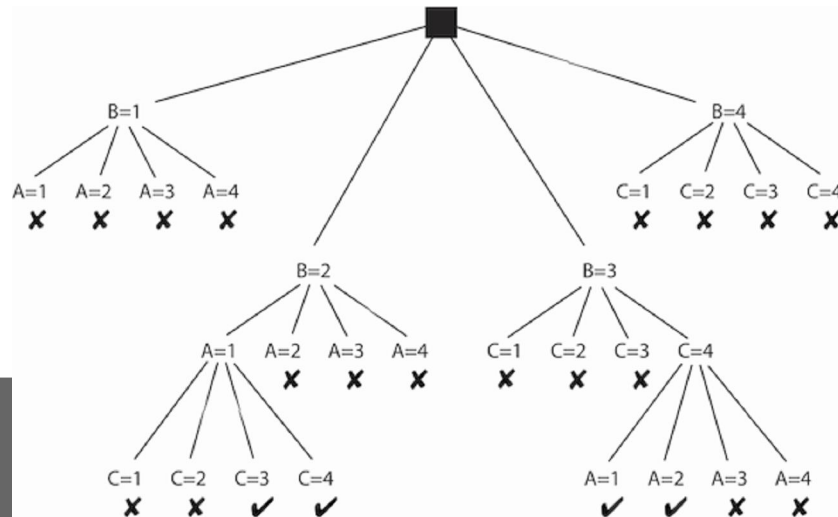


Fig. : Search tree for the CSP of Example 4.13

In this figure, a node corresponds to all of the assignments from the root to that node. The potential nodes that are pruned because they violate constraints are labeled with "X". The leftmost "X" corresponds to the assignment $A=1, B=1$. This violates the $A < B$ constraint, and so it is pruned.

This CSP has four solutions. The leftmost one is $A=1, B=2, C=3$. The size of the search tree, and thus the efficiency of the algorithm, depends on which variable is selected at each time. A static ordering, such as always splitting on A then B then C, is less efficient than the dynamic ordering used here. The set of answers is the same regardless of the variable ordering.

In the preceding example, there would be $4^3=64$ assignments tested in a generate-and-test algorithm. For the search method, there are 22 assignments generated.

Searching with a depth-first search, typically called **backtracking**, can be much more efficient than generate and test. Generate and test is equivalent to not checking constraints until reaching the leaves. Checking constraints higher in the tree can prune large subtrees that do not have to be searched.

2.2.4 Local Search for CSP

Q17. Explain Local Search Algorithm for CSP.

Ans :

Local search methods start with a complete assignment of a value to each variable and try to iteratively improve this assignment by improving steps, by taking random steps, or by restarting with another complete assignment. A wide variety of local search techniques has been proposed. Understanding when these techniques work for different problems forms the focus of a number of research communities, including those from both operations research and AI.

Procedure - Local-Search(V, dom, C)

Inputs

V : a set of variables

dom : a function such that $\text{dom}(X)$ is the domain of variable X

C : set of constraints to be satisfied

Output

complete assignment that satisfies the constraints

Local

$A[V]$ an array of values indexed by V

repeat

for each variable X **do**

$A[X] \leftarrow$ a random value in $\text{dom}(X)$;

while (stopping criterion not met & A is not a satisfying assignment)

 Select a variable Y and a value $V \in \text{dom}(Y)$

 Set $A[Y] \leftarrow V$

if (A is a satisfying assignment) **then**

return A

until termination

Fig. : Local search for finding a solution to a CSP

The generic local search algorithm for CSPs is given in Figure 4.6. A specifies an assignment of a value to each variable. The first **for each** loop assigns a random value to each variable. The first time it is executed is called a **random initialization**. Each iteration of the outer loop is called a **try**. A common way to implement a new try is to do a **random restart**. An alternative to random initialization is to use a construction heuristic that guesses a solution, which is then iteratively improved.

The while loop does a **local search**, or a **walk**, through the assignment space. It maintains a current assignment S , considers a set of **neighbors** of the current assignment, and selects one to be the next current assignment. Here the neighbors of a total assignment are those assignments that differ in the assignment of a single variable. Alternate sets of neighbors can also be used and will result in different search algorithms.

This walk through assignments continues until either a satisfying assignment is found and returned or some stopping criterion is satisfied. The stopping criterion is used to decide when to stop the current local search and do a random restart, starting again with a new assignment. A stopping criterion could be as simple as stopping after a certain number of steps.

This algorithm is not guaranteed to halt. In particular, it goes on forever if there is no solution, and it is possible to get trapped in some region of the search space. An algorithm is **complete** if it finds an answer whenever there is one. This algorithm is incomplete.

One instance of this algorithm is **random sampling**. In this algorithm, the stopping criterion is always true so that the while loop is never executed. Random sampling keeps picking random assignments until it finds one that satisfies the constraints, and otherwise it does not halt. Random sampling is complete in the sense that, given enough time, it guarantees that a solution will be found if one exists, but there is no upper bound on the time it may take. It is very slow. The efficiency depends only on the product of the domain sizes and how many solutions exist.

Another instance is a **random walk**. In this algorithm the while loop is only exited when it has found a satisfying assignment (i.e., the stopping criterion is always false and there are no random restarts). In the while loop it selects a variable and a value at random. Random walk is also complete in the same sense as random sampling

2.2.5 Formulating Problem Structure

Q18. What is Problem Formulation? Explain How to formulate the problems in AI.

Ans :

Problem Formulation

- ▶ A problem formulation is about deciding what actions and states to consider, we will come to this point it shortly.
- ▶ We will describe our states as "in(CITYNAME)" where CITYNAME is the name of the city in which we are currently in.

Now suppose that our agent will consider actions of the form "Travel from city A to City B". and is standing in city 'A' and wants to travel to city 'E', which means that our current state is in(A) and we want to reach the state in(E).

There are 3 roads out of A, one toward B, one toward C and one toward D, none of these achieves the goal and will bring our agent to state in(E), given that our agent is not familiar with the geography of our alien map then it doesn't know which road is the best to take, so our agent will pick any road in random.

Now suppose that our agent is updated with the above map in its memory, the point of a map that our agent now knows what action bring it to what city, so our agent will start to study the map and consider a hypothetical journey through the map until it reaches E from A.

Once our agent has found the sequence of cities it should pass by to reach its goal it should start following this sequence.

The process of finding such sequence is called **search**, a search algorithm is like a black box which takes **problem** as input returns a **solution**, and once the solution is found the sequence of actions

it recommends is carried out and this is what is called the **execution phase**.

We now have a simple (formulate, search, execute) design for our problem solving agent, so lets find out precisely how to formulate a problem.

Formulating Problems

A problem can be defined formally by 4 components :

1. Initial State :

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

2. State Description :

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

3. Goal Test :

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

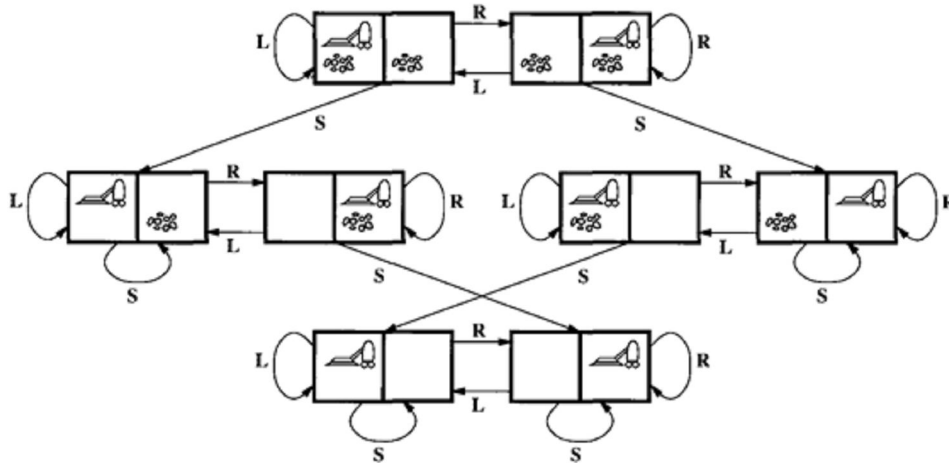
4. Path cost :

- ▶ a function that assigns a numeric value to each path, each step we take in solving the problem should be somehow weighted, so If I travel from A to E our agent will pass by many cities, the cost to travel between two consecutive cities should have some cost measure, {e.i: Traveling from 'A' to 'B' costs 20 km or it can be typed as c(A, 20, B)}.

A solution to a problem is path from the initial state to a goal state, and **solution quality** is

measured by the path cost, and the **optimal solution** has the lowest path cost among all possible solutions.

Example Problems



Vacuum World

1. Initial state:

- Our vacuum can be in any state of the 8 states shown in the picture.

2. State description:

- Successor function generates legal states resulting from applying the three actions {Left, Right, and Suck}.
- The states space is shown in the picture, there are 8 world states.

3. Goal test:

- Checks whether all squares are clean.

4. Path cost:

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

2.3 KNOWLEDGE AND REASONING

2.3.1 Knowledge Representation

Q19. Write a note on Knowledge representation.

Ans :

Knowledge Representation

For the purpose of solving complex problems encountered in AI, we need both a large amount of knowledge and some mechanism for manipulating that knowledge to create solutions to new problems. A variety of ways of representing knowledge (facts) have been exploited in AI programs. In all variety of knowledge representations, we deal with two kinds of entities.

- Facts :** Truths in some relevant world. These are the things we want to represent.

- b. Representations of facts in some chosen formalism these are things we will actually be able to manipulate.

One way to think of structuring these entities is at two levels : (a) the knowledge level, at which facts are described, and (b) the symbol level, at which representations of objects at the knowledge level are defined in terms of symbols that can be manipulated by programs.

The facts and representations are linked with two-way mappings. This link is called representation mappings. The forward representation mapping maps from facts to representations. The backward representation mapping goes the other way, from representations to facts.

One common representation is natural language (particularly English) sentences. Regardless of the representation for facts we use in a program, we may also need to be concerned with an English representation of those facts in order to facilitate getting information into and out of the system. We need mapping functions from English sentences to the representation we actually use and from it back to sentences.

2.3.2 Knowledge-Based Agents

Q20. What are known as Knowledge based agents? Explain them.

Ans :

- A knowledge-based agent includes a knowledge base and an inference system.
- A knowledge base is a set of representations of facts of the world.
- Each individual representation is called a sentence.
- The sentences are expressed in a knowledge representation language.
- The agent operates as follows :
 1. It TELLS the knowledge base what it perceives.
 2. It ASKS the knowledge base what action it should perform.
 3. It performs the chosen action.

Architecture of a KB agent

➤ Knowledge Level

- The most abstract level: describe agent by saying what it knows
- **Ex:** A taxi agent might know that the Golden Gate Bridge connects San Francisco with the Marin County

➤ Logical Level

- The level at which the knowledge is encoded into sentences
- **Ex:** links(GoldenGateBridge, San Francisco, MarinCounty)

➤ Implementation Level

- The physical representation of the sentences in the logical level
- Ex: '(links golden gate bridges an francisco marin county)

A simple knowledge-based agent

function KB-AGENT(percept) returns an action

static; KB, a knowledge base t,

a counter, initially 0, indicating time

TELL [KB, MAKE-PERCEPT-SENTENCE (percept, t,)

action ← Ask(KB, MAKE-ACTION-QUERY(t))

TELL{KB, MAKE- ACTION-SENTENCE(action, t))

t ← t + 1

return action

- The agent must be able to:
 - Represent states, actions, etc.
 - Incorporate new percepts
 - Update internal representations of the world
 - Deduce hidden properties of the world

2.3.3 The Wumpus World

Q21. What is WUMPUS World Problem? Explain.

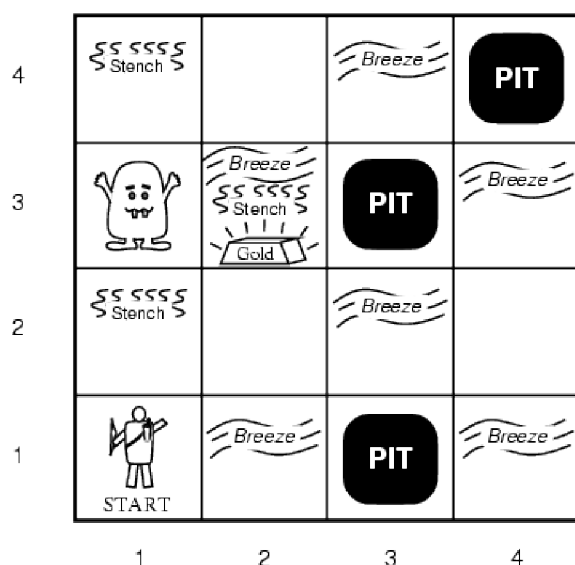
Ans :

Wumpus world problem is a computer game comes under the realm of problems known as Grid World problems. A Grid World problem is a very common problem in the Reinforcement Learning branch of A.I. So envision this:

1. The environment is a grid of dimensions, say 3 x 3 squares.
2. A robot/agent is placed randomly at a location in the grid which is its starting point.
3. The robot/agent is given a destination point.
4. The robot has a fixed set of actions i.e. moving right, moving left etc
5. The robot/agent is surrounded by walls and can only perceive the world after coming in contact with it: i.e. bumping into wall etc
6. There is a rewarding system based on the action taken : for example positive reward reinforcement for correct action taken, negative reward reinforcement for incorrect action taken etc.
7. The objective is to implement an RL algorithm with which the robot reaches its

Solution :

AIMA's Wumpus World



- The agent always starts in the field [1,1]
- Agent's AIMA's Wumpus World
- Task is to find the gold, return to the field [1,1] and climb out of the cave

Agent in a Wumpus world: Percepts

- The agent perceives
 - a stench in the square containing the Wumpus and in the adjacent squares (not diagonally)
 - a breeze in the squares adjacent to a pit
 - a glitter in the square where the gold is
 - a bump, if it walks into a wall
 - a woeful scream everywhere in the cave, if the Wumpus is killed
- The percepts are given as a five-symbol list. If there is a stench and a breeze, but no glitter, no bump, and no scream, the percept is [Stench, Breeze, None, None, None]

The agent cannot perceive its own location

Wumpus World Actions

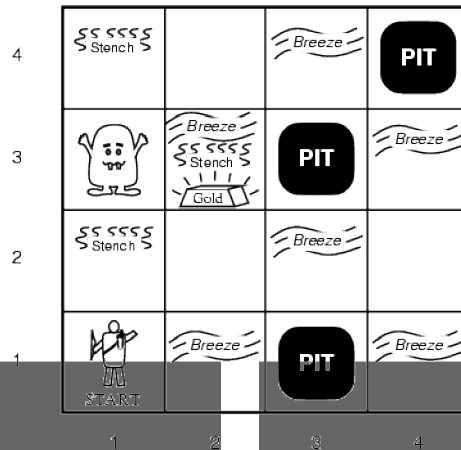
- go forward
- turn right 90 degrees
- turn left 90 degrees
- **grab:** Pick up an object that's in the same square as the agent
- **shoot:** Fire an arrow in a straight line in the direction the agent is facing. It continues until it hits and kills the Wumpus or hits the outer wall. The agent has only one arrow, so only the first shoot action has any effect
- climb is used to leave the cave and is only effective in the start square
- **die:** This action automatically and irretrievably happens if the agent enters a square with a pit or a live Wumpus

Wumpus World Goal

The agent's goal is to find the gold and bring it back to the start square as quickly as possible, without getting killed

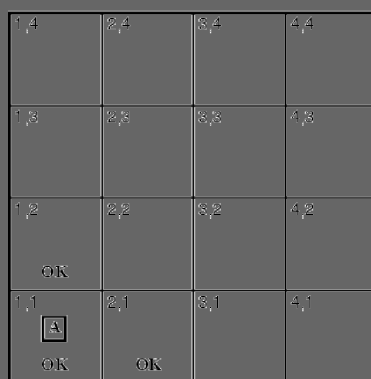
- 1,000 points reward for climbing out of the cave with the gold
- 1 point deducted for every action taken
- 10,000 points penalty for getting killed

AIMA's Wumpus World

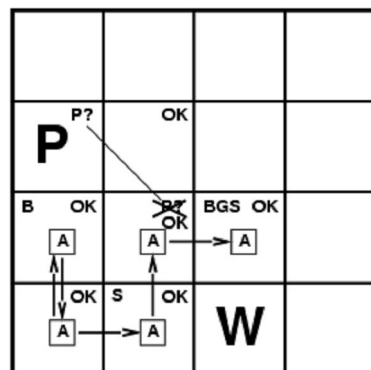
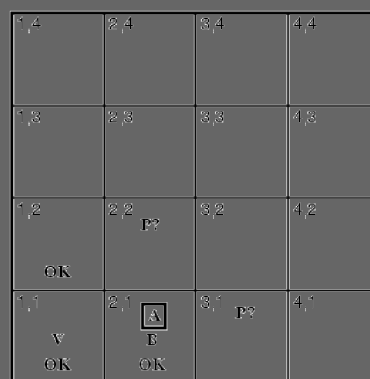


- The agent always starts in the field [1,1]
- Agent's AIMA's Wumpus World
- Task is to find the gold, return to the field [1,1] and climb out of the cave

The Hunter's first step



A – Agent
 B – Breeze
 G – Glitter, Gold
 OK – Safe square
 P – Pit
 S – Stench
 V – Visited
 W – Wumpus



A agent
 B breeze
 G glitter
 OK safe cell
 P pit
 S stench
 W wumpus

2.3.4 Logic

Q22. What is Logic? Explain briefly about symbolic logic in AI.

Ans :

Symbolic Logic was being used to represent knowledge even before the advent of digital computers. Today First Order Predicate Logic (FOPL) or simple Predicate Logic plays an important role in AI for the representation of knowledge. A familiarity with FOPL is important. It offers the only formal approach to reasoning that has a sound theoretical foundation, this is important in the attempts to automate the reasoning process, because inference should be correct and logically sound. The structure of FOPL is flexible and it permits accurate representation of natural language reasonably well.

Logic is a formal method for reasoning. Many concepts which can be verbalized can be translated into symbolic representations which closely approximate the meaning of these concepts. These symbolic structures can then be manipulated in programs to deduce various facts, to carry out a form of automated reasoning. In predicate logic statements from a natural language like English are translated into symbolic structures comprised of predicates, functions, variables, constants, quantifiers and logical connectives.

The symbols form the basic building blocks for the knowledge, and their combination into valid structures is accomplished using the syntax of FOPL. Once structures have been created to represent basic facts, inference rules may then be applied to compare, combine and transform these "assumed" structures into new "deduced" structures. This is how automated reasoning or inferencing is performed.

{ The following standard logic symbols are used in knowledge representation:

\longrightarrow	for "implies" or "then"
\sim	for "not" or "negation"
\wedge	for "and" or "Conjunction"
\vee	for "or" or "disjunction"
\longleftrightarrow	for "iff" or "Double implication"
\forall	for "forall"
\exists	for "there exists" }

as a simple example of use of logic, the statement

"All employees of the ABC company are programmers" might be written in predicate logic as

$(\forall x) (ABC_Co_Emp(x) \longrightarrow programmer(x)).$

$\forall x$ is read as "for all x". the predicates $ABC_Co_Emp(x)$ and $programmer(x)$ are read as "if x is an ABC_Co" and "x is a programmer", respectively. the symbol x is a variable and it can assume a person's name. if it is also known that Ram is employee of ABC company, one can draw the condition that Ram is a programmer.

This example shows how natural language sentences can be translated into predicate logic statements. Once translated, such statements can be put into a knowledge base and subsequently used in a program to perform inferencing.

2.3.5 Propositional Logic

Q23. Write a note on propositional logic?

Ans :

The syntax of propositional logic is constructed from propositions and connectives.

A **proposition** is a statement that is either true or false but not both

Examples of Propositions

- What is the time?
- $2 + 3 = 5$
- "Phone" has five letters.

Determine the propositions

- It is possible to determine whether any given statement is a proposition by prefixing it with
- It is true that . . .
- and seeing whether the result makes grammatical sense.

Propositions are often abbreviated using propositional variables eg.: p , q , r .

Thus we must associate the propositional variable with its meaning i.e.

Let p be Najib is Prime Minister.

Connectives

Propositions may be combined with other propositions to form **compound propositions**. These in turn may be combined into further propositions.

- The connectives that may be used are

and conjunction (& or \cdot)

or disjunction (\vee or $+$)

not negation (\sim)

if . . . then implication (\Rightarrow)

if and only if equivalence (\Leftrightarrow)

Propositional Logic: Syntax

- The proposition symbols P_1 , P_2 etc are sentences
- If S is a sentence, $\sim S$ is a sentence (negation)

- If S_1 and S_2 are sentences, $S_1 \cdot S_2$ is a sentence (conjunction)
- If S_1 and S_2 are sentences, $S_1 \vee S_2$ is a sentence (disjunction)
- If S_1 and S_2 are sentences, $S_1 \Rightarrow S_2$ is a sentence (implication)
- If S_1 and S_2 are sentences, $S_1 \Leftrightarrow S_2$ is a sentence (equivalence)

Propositional Logic: Semantic

Each model species true/false for each proposition symbol

E.g. P Q R

True true false

(With these symbols, 8 possible models, can be enumerated automatically.)

Rules for evaluating truth with respect to a model m are determined by truth tables

And (also called conjunction)

The conjunction ' p AND q ', written $p \wedge q$, of two propositions is true when both p and q are true, false otherwise.

We can summarise the operation of using a truth table. Rows in the table give all possible setting of the propositions to true (T) or false (F).

p	q	$p \wedge q$
T	T	T
T	F	F
F	T	F
F	F	F

Natural Language Meaning

P – Its Monday.

Q – Its raining.

$p \wedge q$ Its Monday and its raining.

Its Monday but its raining.

Its Monday. Its raining.

P - I took a shower

Q - I woke up

$p \wedge q$ "I took a shower and I woke up"

$q \wedge p$ "I woke up and I took a shower".

Logically the same! WE may see a difference.

The word both is often useful eg. I both took a shower and I woke up.

Or (\vee) Disjunction

The disjunction " p OR q " written $p \vee q$, of two propositions is true when p or q (or both) are true, false otherwise.

Sometimes called inclusive or.

p	q	$p \vee q$
T	T	T
T	F	T
F	T	T
F	F	F

Natural Language Meaning

P – It's Monday.

Q – It's raining.

$P \vee q$ == It's Monday or it is raining.

The word either is often useful eg. either it's Monday or it is raining.

It also includes the case of rain on a Monday!

Not \neg (Negation)

The negation "NOT p " of a proposition (or p) is true when p is false and is false otherwise. p may be read that it is false that p .

p	$\neg p$
T	F
F	T

Negation is a unary connective. It only takes one argument. Conjunction and disjunction were both binary connectives.

Natural Language Meaning

P – Logic is easy.

P – It is false that logic is easy.

It is not the case that logic is easy.

Logic is not easy.

If . . . then \Rightarrow Also known as **implication**

The implication " p IMPLIES q ", written $p \Rightarrow q$, of two propositions is true when either p is false or q is true, and false otherwise.

p	q	$p \Rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

Natural Language Meaning

P – I study hard.

Q – I get rich.

$p \Rightarrow q$ If I study hard then I get rich.

Whenever I study hard, I get rich.

That I study hard implies I get rich.

I get rich, if I study hard.

Biconditional \Leftrightarrow Also known as iff or the **biconditional**.

The bi-conditional, written as $p \Leftrightarrow q$, of two propositions is true when both p and q are true or when both p and q are false, and false otherwise.

p	q	$p \Leftrightarrow q$
T	T	T
T	F	F
F	T	F
F	F	T

WFF

The set of sentences or well-formed propositional formulae (WFF) is defined as:

Any propositional symbol is in WFF.

The nullary connectives, **true** and **false** are in WFF.

If A and B are in WFF then so is $\neg A$, $A \vee B$, $A \wedge B$, $A \supset B$ and $A \equiv B$.

If A is in WFF then so is (A) .

Example

$p(p) \quad p \vee q \quad ((p \supset q) \wedge \neg q) \supset \neg p$

2.3.6 Predicate Logic

Q24. Write about Predicate Logic.

Ans :

Propositional logic lacks the structure to express relations that exist among two or more entities. Predicate logic was developed by logicians to extend the expressiveness of propositional logic. It is a generalization of propositional logic that permits reasoning about world objects as relational entities as well as classes or subclasses of objects. This generalization is achieved from the introduction of predicates in the place of propositions and the use of variables together with variable quantifiers. The quantifier symbols used are (universal quantifier) and (existential quantifier) \exists stands for "for all" and \exists stands for "there exist". Thus $\forall x$ means "for all x" and $\exists x$ means "for some x there is an x".

For example, the sentence "All elephant are gray" might be represented by

$(\forall x) [\text{elephant}(x) \longrightarrow \text{Color}(x, \text{gray})]$

(Read as "For all x, if x is an elephant then the color of x is gray"). In the above quantified wff, 'Elephant' and 'color' are predicates. 'x' and 'gray' are terms. x is a variable term and 'gray' is a constant term.

Atomic formulas like $\text{color}(x, \text{gray})$ are merely elementary building blocks of predicate calculus language. We can combine atomic formulas (simple wffs) to form complex wffs by using connectives such as \wedge (and), \vee (or) and \longrightarrow (implies). Formulas built by connecting other formulas are called "conjunctions" each of the component formulas is called "conjunct". formulas built with \vee are called "disjunctions" and the components are called "disjuncts". Any conjunction or disjunction composed of wffs is also a wff.

For example "John likes apples and John likes oranges" is represented as a conjunction as

LIKES (JOHN, APPLES) \wedge LIKES (JOHN, ORANGES)

In predicate calculus it is often useful to have computable functions and computable predicates. For example the simple fact $(2+3) > 1$ might be represented as $\text{gt}(2+3+1)$.

To evaluate the truth of $\text{gt}(2+3,1)$, the value of the plus functions is evaluated first and then the arguments $2+3(=5)$ and 1 are sent to the computable predicate gt . The use of computable functions and predicates is illustrated below.

Consider the following set of facts and their representations :

- John was a man
 $\text{man}(\text{John})$
 - John was a graduate
 $\text{graduate}(\text{John})$
 - John liked apples and oranges
 $\text{liked}(\text{John}, \text{apples}) \wedge \text{liked}(\text{John}, \text{oranges})$
 - John was born in 1832
 $\text{born}(\text{John}, 1832)$
 - All men mortal
 $\forall x \text{ man}(x) \longrightarrow \text{mortal}(x)$
 - No man lives longer than 150 years
 $\forall x \forall t_1 \forall t_2 \text{ man}(x) \text{ born}(x, t_1) \wedge \text{gt}(t_2 - t_1, 150) \longrightarrow \text{dead}(x, t_2)$
 - If someone dies, then he is dead at all later times.
 $\forall x \forall t_1 \forall t_2 \text{ died}(x, t_1) \wedge \text{gt}(t_2, t_1) \longrightarrow \text{dead}(x, t_2)$
 - It is now 1995
 $\text{now} = 1995$
 - Alive means not dead
 $\forall x \forall t \text{ alive}(x, t) \longrightarrow \neg \text{dead}(x, t)$
- Now let us attempt to answer the question "is John alive?" by proving
 $\sim \text{alive}(\text{John}, \text{now})$

The proof is given in the following

\sim alive (John, now)

↓ (9, substitution)

dead (john , now)

↓ (6, substitution)

man (john) born (john, tl) gt (now-tl. 150)

↓ (4, substitution)

born (john, 1832) gt (now-tl , 150)

↓ (4)

gt (now-1832 , 150).

↓ (8, substitution)

gt (1995-1832,150)

↓ (compute gt)

nil.

Fig.: Proving that john is dead

2.4 UNIFICATION AND LIFTING

2.4.1 Inference in FOL

Q25. Explain the rules of Inference for FOL.

Ans :

Inference rules for FOL

- The propositional logic inference rules
- Remember the inference rules for propositional logic:

1. Modus Ponens or Implication Elimination

$A \Rightarrow B, A$

—————
B

2. And-Elimination

A1 and A2 and ... An

—————
Ai

3. And-Introduction

A1, A2, ... An

—————
A1 and A2 and ... An

4. Or-Introduction

Ai

—————
A1 or A2 or ... An

5. Double-Negation Elimination

NOT(NOT(A))

—————
A

6. Unit Resolution

A or B, NOT(B)

—————
A

7. Resolution

A or B, NOT(B) or C

—————
A or C

These rules are still valid for FOL.

A, B, and C are now atomic sentences (ie, a predicate or Term=Term, no negation).

Substitution

We need some way to deal with variables and the quantifiers "EXISTS" and "FORALL".

To describe inference rules involving variables and quantifiers, we need the notion of "substitution".

Let α be any sentence and let θ be a substitution list:

$\theta = \{x/fred, y/z, \dots\}$

The notation $SUBST(\theta, \alpha)$ will denote the result of applying the substitution θ to the sentence α . Intuitively $subst(x := g, \alpha)$ is α with every appearance of x replaced by g .

Elimination and Introduction

Many inference rules tell how to get rid of (eliminate) or introduce a connective or quantifier into a formula.

The "universal elimination" rule lets us use a universally quantified sentence to reach a specific conclusion, i.e. to obtain a concrete fact.

FORALL x α

—————
 $SUBST(\{x/g\} \alpha)$

Here g may be any term, including any term that is used elsewhere in the knowledge base.

The “existential elimination” rule lets us convert an existentially quantified sentence into a form without the quantifier.

$$\frac{\text{EXISTS } x \text{ } \alpha}{\text{SUBST}(\{x/k\} \text{ } \alpha)}$$

Here k must be a new constant symbol that does not appear anywhere else in the database. It is serving as a new name for something we know must exist, but whose name we do not yet know.

The “existential introduction” rule lets us use a specific fact to obtain an existentially quantified sentence:

$$\alpha$$

$$\text{EXISTS } y \text{ SUBST}(\{g/y\}, \alpha)$$

This rule can be viewed as “leaving out detail”: it omits the name of the actual entity that satisfies the sentence α , and just says that some such entity exists.

Skolemization

The “existential elimination” rule is also called the Skolemization rule, after a mathematician named Thoralf Skolem.

$$\frac{\text{EXISTS } x \text{ } \alpha}{\text{subst}(\{x/k\}, \alpha)}$$

where k is a FRESH constant

Saying that k is a “fresh” constant means that k is not used anywhere else in the knowledge base.

This means that k can be the name of a new entity that is not named and therefore not referred to anywhere else in the knowledge base.

Example: consider the fact

exists z brotherof(Harry, z)

It would not be correct to directly deduce

brotherof(Harry,William)

However it is correct to deduce

brotherof(Harry,personA)

and then later we might deduce that

personA = William.

Generalized Modus Ponens

This is a complex inference rule that captures a common pattern of reasoning.

$$p_1 \ p_2 \ \dots \ p_n \ (q_1 \ \text{AND} \ q_2 \ \text{AND} \ \dots \ \text{AND} \ q_n \rightarrow r)$$

$$\text{SUBST}(\text{theta}, r)$$

where for each i $\text{SUBST}(\text{theta}, p_i) = \text{SUBST}(\text{theta}, q_i)$.

Intuitively this is modus ponens with multiple antecedents on the left hand side of the \rightarrow connective.

Theta is a SET of individual substitutions of the form x/g where x is a variable and g is a ground term.

The equation $\text{SUBST}(\text{theta}, p_i) = \text{SUBST}(\text{theta}, q_i)$ says that applying theta makes p_i and q_i identical.

Terminology: theta UNIFIES p_i and q_i , theta is a unifying substitution.

The unification algorithm

There is a standard algorithm that given two sentences, finds their unique most general unifying substitution.

Here are some examples of unification.

Knows(John, x) = Likes(John, x)

Knows(John, Jane)

Knows(y , Leonid)

Knows(y , Mother(y))

The unification algorithm would give the following results:

$\text{UNIFY}(\text{Knows}(\text{John}, x), \text{Knows}(\text{John}, \text{Jane})) = \{x/\text{Jane}\}$

$\text{UNIFY}(\text{Knows}(\text{John}, x), \text{Knows}(y, \text{Leonid})) = \{x/\text{Leonid}, y/\text{John}\}$

The substitution makes the two sentences identical.

$\text{UNIFY}(\text{Knows}(\text{John}, x), \text{Knows}(y, \text{Mother}(y))) = \{x/\text{John}, x/\text{Mother}(\text{John})\}$

Notice that the substitutions always involve a **variable** and a **ground term**.

UNIFY(Knows(John, x), Knows(x, Elizabeth)) = fail

The variable x cannot have two values at the same time, so this last example fails.

Another example that would always fail is

UNIFY(x, F(x)) = fail

This fails because a the variable may never occur in the term it is being unified with.

Horn clauses

Generalized modus ponens requires sentences to be in a standard form, called Horn clauses after the mathematician Alfred Horn.

A Horn clause is a sentence of the form

$q_1 \text{ AND } q_2 \text{ AND } \dots \text{ AND } q_n \rightarrow r$

where each q_i and r is an **atomic sentence** and all variables are universally quantified.

Recall that an atomic sentence is a just single predicate and does not allow **negation**.

Normally the universal quantifiers are implicit (not written explicitly).

To refresh your memory, here is the syntax of FOL in Backus-Naur form.

Sentence \rightarrow AtomicSentence

| Sentence Connective Sentence

| Quantifier Variable,... Sentence

| NOT Sentence

| (Sentence)

Atomic Sentence \rightarrow Predicate(Term, ...) | Term = Term

Term \rightarrow Constant

| Variable

| Function(Term,...)

Connective \rightarrow AND | OR | \Rightarrow | \Leftarrow

Quantifier \rightarrow FORALL | EXISTS

Constant \rightarrow A | X₁ | John | ...

Variable \rightarrow a | x | s ...

Predicate \rightarrow Before | HasColor | Raining | ...

Function \rightarrow Smelly | LeftLegOf | Plus | ...

In order to obtain Horn clauses, existential quantifiers must be eliminated using Skolemization.

Example

EXISTS x Loves(John,x) AND Loves(x, John) becomes

Loves(John,personx)

Loves(personx,John)

2.4.2 Representing Knowledge Using Rules

Q26. Write about Knowledge representation rules.

Ans :

Knowledge Representation Rules

Some kind of knowledge are hard to represent in predicate logic. For example the degree of hotness in the statement " It is very hot today" can not be represented in predicate logic. A good deal of the reasoning people do involves manipulating beliefs. (eg: "I think it may rain today because it is cloudy "). The "belied system " is mostly incomplete and inconsistent (One may believe in something now and in something else later). consider the following situation.

A, B and C are suspects in a murder case. A has an alibi, in the register of a respectable hotel. B also has an alibi since his friend says that B was with him all the time. C pleads alibi too saying that he was watching a cricket match in the town. These make one believe that

1. A did not commit the crime
2. B did not commit the crime
3. A or B or C did fortunately for c, he was caught on the TV while watching the match. Now we have a new belief that
4. C did not commit the crime

All the above four beliefs are inconsistent, so we must reject the weaker one or add new beliefs.

The above example illustrates some of the problems posed by uncertain and fuzzy knowledge. A variety of techniques for handling such problems with in computer programs have been proposed. They include

Non monotonic logic

This allows for addition and deletion of statements in the database. This also allows the belief in one statement to rest on the lack of belief in another.

Probabilistic Reasoning

This makes it possible to represent likely but uncertain inferences.

Fuzzy logic

This provides a way of representing fuzzy or continuous properties of objects.

The concept of belief spaces, which allows for the representation of nested models of sets of beliefs.

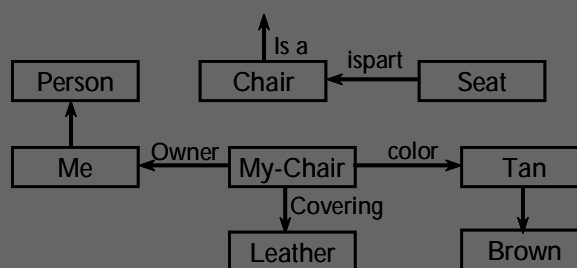
2.4.3 Semantic Networks

Q27. Write a note on Semantic Nets

Ans :

Semantic Nets

It is useful to think of semantic nets using graphical notation. In this information is represented as a set nodes connected to each other by a set of labeled arcs, which represent relationships between the nodes. A typical example (with ISA and ISPART relationships) is shown below.



The knowledge in the above semantic network is represented inside a program using some kind of attribute –value structure. The following gives a LISP representation of the above semantic net.

```

ATOM PROPERTY LIST
CHAIR ((IS A FURNITURE))
MY-CHAIR ((IS A CAHIR)
(COLOR TAN)
(COVERING LEATHER)
(OWNER ME ))
ME ((IS A PERSON))
TAN ((IS A BROWN))
SEAT ((ISPART CHAIR))
  
```

In predicate logic the above may be represented as

```

IS A (chair, furniture)
IS A ( me, person)
COVERING ( my-chair, leather)
COLOR (my-chair, tan)
  
```

2.4.4 Frame Systems

Q28. Write a note Frame systems.

Ans :

Frames

Semantic Networks and conceptual depednecy can be used to represent specific events or experiences. A frame structure is used to analyze new situations from scratch and then build new knowledge structures to describe those situations. Typically , a frame describes a class of objects, such as CHAIR or ROOM . it consists of a collection of "slots" that describe aspects of the objects. Associated with each slot may be a set of conditions that must be met by any filler for it. Each slot may also be filled with a default value, so that in the absence of specific information, things can be associated to be as they usually are. Procedural information may also be associated with particular slots. The AI systems exploit not one but many frames. Related frames can be grouped together to form a frame system.

Frames represent an object as a group of attributes. Each attributes in a particular frame is stored in a separate slot. For example, when a furniture salesman says " I have a nice chair, that I want you to see", the word 'chair' would immediately trigger in our minds a series of expectations. We would probably expect to see an object with four legs, a seat , a back and possibly (but not necessarily) two arms. We would expect it to have a particular size and serves a place to sit. In an AI system, a frame CHAIR might include knowledge organized as shown below:

Frame : CHAIR

Parts : seat, back, legs, arms

Number of legs : 4

Number of arms: 0 or 2

Default : 0

2.4.5 Inference

Q29. Write about backward and forward chaining.

Ans :

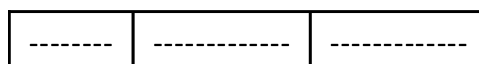
An **Inference Engine** is a tool from artificial intelligence. The first inference engines were components of expert systems. The typical expert system consisted of a knowledge base and an inference engine. The knowledge base stored facts about the world. The inference engine applied logical rules to the knowledge base and deduced new knowledge. This process would iterate as each new fact in the knowledge base could trigger additional rules in the inference engine. Inference engines work primarily in one of two modes either special rule or facts: forward chaining and backward chaining. Forward chaining starts with the known facts and asserts new facts. Backward chaining starts with goals, and works backward to determine what facts must be asserted so that the goals can be achieved.

Inference Rules

Deductive inference rule:

Forward Chaining: Conclude from "A" and "A implies B" to "B".

A
A -> B
B



Example

It is raining.
 If it is raining, the street is wet.
 The street is wet.

**Abductive inference rule**

Backward Chaining: Conclude from "B" and "A implies B" to "A".

B
 A \rightarrow B
 A

**Example**

The street is wet.
 If it is raining, the street is wet.
 It is raining.

2.4.6 Types of Reasoning**30. What is Reasoning? Explain different types of reasoning.**

Ans :

Reasoning is the capacity for a person to make sense of things to establish & verify facts, rationally work through data, information, facts, and beliefs. It is the process of forming conclusions and judgments from facts or premises. To put it plain and simple; it is the ability to coherently think from perceived premise to a logical conclusion.

Main Types of Reasoning**Deductive Reasoning**

Deductive reasoning is the form of reasoning in which a conclusion follows logically and coherently from the factual premises and proposition. These deductive arguments are based upon the concept of sound and consistent reasoning. If the premises are true, then the systematic reasoning with a constructed syllogism is considered valid in a deductive argument in making its conclusion certain with a degree of logical

certainty. Plainly speaking, deductive reasoning is the rationality of reasoning from *pure logic*. It is considered sound and pure logic.

Inductive Reasoning

Inductive reasoning is a form of reasoning that uses analogies, examples, observations, and experiences to form conclusive propositions. Inductive logic also uses experiences to formulate statements based on general observations of recurring patterns in nature, science, and everyday occurrences pulling from such things as samples cases, experiments, and natural eye observations.

It is used mostly to explain properties and relations to objects or types based on previous observations.

Abductive Reasoning

In laymen's terms abductive reasoning is an argument to the best explanation. It is a form of reasoning that concludes in an abductive argument of what is plausible or most possibly true. Abductive logic is also considered *inference to the best explanation*. It is choosing the most likely or best hypothesis or explanation based upon the (most) relevant evidence.

Reductive Reasoning

Reductive reasoning is a subset of argumentative reasoning which seeks to demonstrate that a statement is true by showing that a false or absurd result/circumstance follows from its denial. It is proving a statement true by reducing to the opposite of it and showing the absurdity of the opposite result.

Fallacious Reasoning

Fallacious Reasoning is not real reasoning, it is the faulty premises for critical thinking and logic. One of the tell tell signs of fallacious reasoning is a logical fallacy. A fallacy is usually an error in reasoning and argumentation often due to a misconception, false premises, or presumptuous conclusions.

Circular Reasoning is actually considered more of a form of fallacious reasoning. It would not be considered valid nor useful in a live debate.

UNIT III

Uncertain Knowledge and Reasoning: uncertainty and methods, Bayesian probability and belief network, probabilistic reasoning, probabilistic reasoning over time, forward and backward reasoning, perception, making simple decisions, making complex decisions, other techniques in uncertainty and reasoning process. Planning problem, simple planning agent, planning languages, blocks world, goal stack planning, means-ends analysis, planning as a state-space search.

Learning: what is machine learning? Learning paradigms, learning concepts, methods and models, statistical learning methods, artificial neural networks-based learning, support vector machines, reinforcement learning.

3.1 UNCERTAIN KNOWLEDGE AND REASONING

3.1.1 Uncertainty and Methods

Q1. What is Uncertainty? How uncertainty can be handled in AI.

Ans :

Though there are various types of uncertainty in various aspects of a reasoning system, the “reasoning with uncertainty” (or “reasoning under uncertainty”) research in AI has been focused on the uncertainty of truth value, that is, to allow and process truth values other than “true” and “false”.

Generally speaking, to develop a system that reasons with uncertainty means to provide the following:

- A semantic explanation about the origin and nature of the uncertainty
- A way to represent uncertainty in a formal language
- A set of inference rules that derive uncertain (though well-justified) conclusions
- An efficient memory-control mechanism for uncertainty management

Let action A_i = leave for airport_i minutes before flight

Will A_i get me there on time?

Problems

1. Partial observability (road state, other drivers' plans, etc.)
2. Noisy sensors (traffic reports)
3. Uncertainty in action outcomes (flat tire, etc.)
4. Immense complexity of modeling and predicting traffic

Hence a purely logical approach either

1. Risks falsehood: “ A_{25} will get me there on time”, or
2. Leads to conclusions that are too weak for decision making:

“ A_{25} will get me there on time if there's no accident on the bridge and it doesn't rain and my tires remain intact etc etc.”

(A_{1440} might reasonably be said to get me there on time but I'd have to stay overnight in the airport ...)

Methods for handling uncertainty

- Default or nonmonotonic logic:
 - Assume my car does not have a flat tire
 - Assume A_{25} works unless contradicted by evidence
- **Issues:** What assumptions are reasonable? How to handle contradiction?
- Rules with fudge factors:
 - $A_{25} \mid \rightarrow_{0.3}$ get there on time
 - Sprinkler $\mid \rightarrow_{0.99}$ WetGrass
 - WetGrass $\mid \rightarrow_{0.7}$ Rain
- **Issues:** Problems with combination, e.g., Sprinkler causes Rain?
- Probability
 - Model agent's degree of belief
 - Given the available evidence,
 - A_{25} will get me there on time with probability 0.04

Making decisions under uncertainty

Suppose I believe the following:

$$P(A_{25} \text{ gets me there on time} \mid \dots) = 0.04$$

$$P(A_{90} \text{ gets me there on time} \mid \dots) = 0.70$$

$$P(A_{120} \text{ gets me there on time} \mid \dots) = 0.95$$

$$P(A_{1440} \text{ gets me there on time} \mid \dots) = 0.9999$$

Which action to choose?

Depends on my preferences for missing flight vs. time spent waiting, etc.

- Utility theory is used to represent and infer preferences
- Decision theory = probability theory + utility theory

Syntax

- Basic element: random variable
- Similar to propositional logic: possible worlds defined by assignment of values to random variables.
- Boolean random variables
e.g., Cavity (do I have a cavity?)
- Discrete random variables
 - Weather is one of <sunny, rainy, cloudy, snow>
 - Domain values must be exhaustive and mutually exclusive
- Elementary proposition constructed by assignment of a value to a random variable:
e.g., Weather = sunny, Cavity = false
(abbreviated as \neg cavity)
- Complex propositions formed from elementary propositions and standard logical connectives e.g., Weather = sunny \vee Cavity = false.

3.1.2 Bayesian Probability and Belief Network**Q2. Write a note on bayesian probability.**

Ans :

Bayes' theorem can be used to calculate the probability that a certain event will occur or that a certain proposition is true.

The theorem is stated as follows:

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

$P(B)$ is called the prior probability of B. $P(B \mid A)$, as well as being called the conditional probability, is also known as the posterior probability of B.

$$P(A \wedge B) = P(A \mid B)P(B)$$

Note that due to the commutativity of \wedge , we can also write

$$P(A \wedge B) = P(B \mid A)P(A)$$

Hence, we can deduce: $P(B \mid A)P(A) = P(A \mid B)P(B)$

This can then be rearranged to give Bayes' theorem:

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

Bayes theorem states

$$P(H_1 \mid E) = \frac{P(E \mid H_1)P(H_1)}{\sum_{k=1}^1 P(E \mid H_1)P(H_1)}$$

This reads that given some evidence E then probability that hypothesis H_1 is true is equal to the ratio of the probability that E will be true given H_1 times the a priori evidence on the probability of H_1 and the sum of the probability of E over the set of all hypotheses times the probability of these hypotheses.

The set of all hypotheses must be mutually exclusive and exhaustive.

Thus to find if we examine medical evidence to diagnose an illness. We must know all the prior probabilities of find symptom and also the probability of having an illness based on certain symptoms being observed.

Bayesian statistics lie at the heart of most statistical reasoning systems. How is Bayes theorem exploited?

The key is to formulate problem correctly:

$P(A \mid B)$ states the probability of A given only B's evidence. If there is other relevant evidence then it must also be considered.

All events must be mutually exclusive. However in real world problems events are not generally unrelated. For example in diagnosing measles, the symptoms of spots and a fever are related. This means that computing the conditional probabilities gets complex.

In general if a prior evidence, p and some new observation, N then

$$P(H|N,p) = P(H|N) \frac{P(p|N,H)}{P(p|N)}$$

computing grows exponentially for large sets of p

All events must be exhaustive. This means that in order to compute all probabilities the set of possible events must be closed. Thus if new information arises the set must be created afresh and all probabilities recalculated.

Thus Simple Bayes rule-based systems are not suitable for uncertain reasoning.

Knowledge acquisition is very hard.

Too many probabilities needed - too large a storage space.

Computation time is too large.

Updating new information is difficult and time consuming.

Exceptions like "none of the above" cannot be represented.

Humans are not very good probability estimators.

However, Bayesian statistics still provide the core to reasoning in many uncertain reasoning systems with suitable enhancement to overcome the above problems. We will look at three broad categories:

- Certainty factors
- Dempster-Shafer models
- Bayesian networks.

Q3. Explain in detail about belief networks.

Ans :

Belief Networks

Bayesian networks are also called Belief Networks or Probabilistic Inference Networks. Belief networks are used to model uncertainty in a domain.

The term "belief networks" encompasses a whole range of different but related techniques which deal with reasoning under uncertainty.

Belief networks are used to develop knowledge based applications in domains which are characterised by inherent uncertainty. Increasingly, belief network techniques are being employed to deliver advanced knowledge based systems to solve real world problems. Belief networks are particularly useful for diagnostic applications and have been used in many deployed systems.

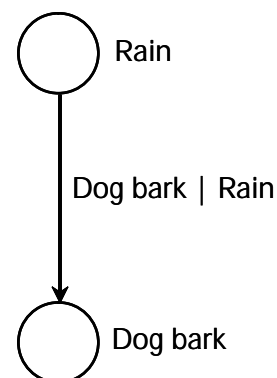
The basic idea in belief networks is that the problem domain is modelled as a set of nodes interconnected with arcs to form a directed acyclic graph. Each node represents a random variable, or uncertain quantity, which can take two or more possible values. The arcs signify the existence of direct influences between the linked variables, and the strength of each influence is quantified by a forward conditional probability.

Bayesian Networks as Graphs

People usually represent Bayesian networks as directed graphs in which each node is a **hypothesis** or a **random process**. In other words, something that takes at least 2 possible values you can assign probabilities to. For example, there can be a node that represents the state of the dog (barking or not barking at the window), the weather (raining or not raining), etc.

The arrows between nodes represent the **conditional probabilities** between them - how information about the state of one node changes the probability distribution of another node it's connected to.

Here's how the events "it rains/doesn't rain" and "dog barks/doesn't bark" can be represented as a simple Bayesian network:



The nodes are the empty circles. Next to each node you see the event whose probability distribution it represents. Next to the arrow is the conditional probability distribution of the second event, given the first event. It reads something like:

- The probability that the dog will start barking, given that it's currently raining.

In general, the nodes don't represent a particular event, but all possible alternatives of a hypothesis (or, more generally, states of a variable). In this case, the set of possible events for the first node consists of:

- It rains

- It doesn't rain

And for the second node:

- The dog barks
- The dog doesn't bark

But in most cases, the nodes can take more than two and often an infinite number of possible values.

Bayesian networks as joint probability distributions

The simple graph above is a Bayesian network that consists of only 2 nodes. It represents a joint probability distribution over their possible values. That's simply a list of probabilities for all possible event combinations:

	Rains	Doesnot rain	
Dog barks	9 / 18	18 / 18	27 / 48
Dog does not bark	3 / 48	18 / 48	21 / 48
	12 / 48	36 / 48	48 / 48

The blue numbers are the joint probabilities of the 4 possible combinations (that is, the probabilities of both events occurring):

- $P(\text{Rains \& Dog barks}) = 9/48 \cong 0.19$
- $P(\text{Rains \& Dog doesn't bark}) = 3/48 \cong 0.06$
- $P(\text{Doesn't rain \& Dog barks}) = 18/48 = 0.375$
- $P(\text{Doesn't rain \& Dog doesn't bark}) = 18/48 = 0.375$

Notice how the 4 probabilities sum up to 1, since the four event combinations cover the entire sample space.

The orange numbers are the so called **marginal probabilities**. You can think of them as the overall probabilities of the events:

- $P(\text{Rains}) = 12/48$
- $P(\text{Doesn't rain}) = 36/48$
- $P(\text{Dog barks}) = 27/48$
- $P(\text{Dog doesn't bark}) = 21/48$

These are obtained by simply summing the probabilities of each row and column.

3.1.3 Probabilistic Reasoning

Q4. What is probabilistic reasoning? Explain.

Ans :

Probability theory is used to discuss events, categories, and hypotheses about which there is not 100% certainty.

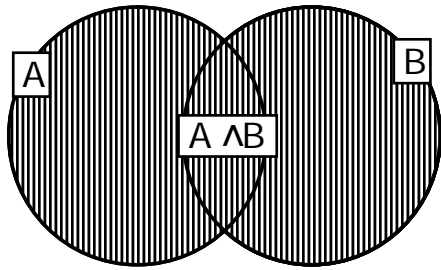
We might write $A \rightarrow B$, which means that if A is true, then B is true. If we are unsure whether A is true, then we cannot make use of this expression.

$$P(S) = 0.1$$

$$P(R) = 0.7$$

The first of these statements says that the probability of S ("it is sunny") is 0.1. The second says that the probability of R is 0.7. Probabilities are always expressed as real numbers between 0 and 1. A probability of 0 means "definitely not" and a probability of 1 means "definitely so." Hence, $P(S) = 1$ means that it is always sunny.

Many of the operators and notations that are used in prepositional logic can also be used in probabilistic notation. For example, $P(\neg S)$ means "the probability that it is not sunny"; $P(S \wedge R)$ means "the probability that it is both sunny and rainy." $P(A \vee B)$, which means "the probability that either A is true or B is true," is defined by the following rule: $P(A \vee B) = P(A) + P(B) - P(A \wedge B)$.



The notation $P(B|A)$ can be read as “the probability of B, given A.” This is known as conditional probability - it is conditional on A. In other words, it states the probability that B is true, given that we already know that A is true. $P(B|A)$ is defined by the following rule: Of course, this rule cannot be used in cases where $P(A) = 0$.

For example, let us suppose that the likelihood that it is both sunny and rainy at the same time is 0.01. Then we can calculate the probability that it is rainy, given that it is sunny as follows:

$$P(R|S) = \frac{P(R \wedge S)}{P(S)} = \frac{0.01}{0.1} = 0.1$$

The basic approach statistical methods adopt to deal with uncertainty is via the axioms of probability:

Probabilities are (real) numbers in the range 0 to 1.

A probability of $P(A) = 0$ indicates total uncertainty in A, $P(A) = 1$ total certainty and values in between some degree of (un)certainty.

Probabilities can be calculated in a number of ways.

$$\text{Probability} = \frac{\text{(number of desired outcomes)}}{\text{(total number of outcomes)}}$$

3.1.4 Probabilistic Reasoning Over Time

Q5. Explain about Markov chains.

Ans:

Markov Chains

A Markov Chain is a special sort of belief network used to represent sequences of values, such as the sequence of states in a dynamic system or the sequence of words in a sentence.

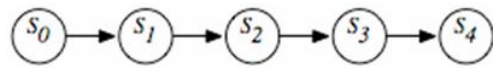


Fig.: A Markov chain as a belief network

Figure shows a generic Markov chain as a belief network. The network does not have to stop at stage s_4 , but it can be extended indefinitely. The belief network conveys the independence assumption

$P(S_{t+1} | S_0, \dots, S_t) = P(S_{t+1} | S_t)$, which is called the Markov assumption.

Often, S_t represents the state at time t . Intuitively, S_t conveys all of the information about the history that can affect the future states. At S_t , you can see that “the future is conditionally independent of the past given the present.”

A Markov chain is stationary if the transition probabilities are the same for each time point [i.e., for all $t > 0, t' > 0, P(S_{t+1} | S_t) = P(S_{t'+1} | S_{t'})$]. To specify a stationary Markov chain, two conditional probabilities must be specified:

- $P(S_0)$ specifies initial conditions.
- $P(S_{t+1} | S_t)$ specifies the dynamics, which is the same for each $t \geq 0$.

Stationary Markov chains are of interest because

- They provide a simple model that is easy to specify.
- The assumption of stationarity is often the natural model, because the dynamics of the world typically does not change in time.
- The network can extend indefinitely.

To determine the probability distribution of state S_i , VE can be used to sum out the preceding variables. Note that the variables after S_i are irrelevant to the probability of S_i and need not be considered. This computation is normally specified as matrix multiplication, but note that matrix multiplication is a simple form of VE. Similarly, to compute $P(S_i | S_k)$, where $k > i$, only the variables before S_k need to be considered.

Q6. Explain hidden Markov models*Ans :***Hidden Markov Models**

A hidden Markov model (HMM) is an augmentation of the Markov chain to include observations. Just like the state transition of the Markov chain, an HMM also includes observations of the state. These observations can be partial in that different states can map to the same observation and noisy in that the same state can stochastically map to different observations at different times.

The assumptions behind an HMM are that the state at time $t+1$ only depends on the state at time t , as in the Markov chain. The observation at time t only depends on the state at time t . The observations are modeled using the variable O_t for each time t whose domain is the set of possible observations. The belief network representation of an HMM is depicted in Figure. Although the belief network is shown for four stages, it can proceed indefinitely.

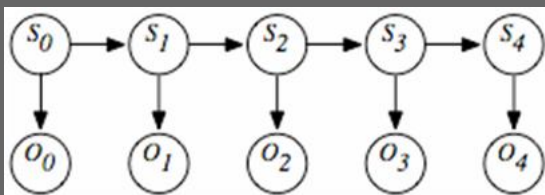


Fig.: A hidden Markov model as a belief network

A stationary HMM includes the following probability distributions:

- $P(S_0)$ specifies initial conditions.
- $P(S_{t+1} | S_t)$ specifies the dynamics.
- $P(O_t | S_t)$ specifies the sensor model.

There are a number of tasks that are common for HMMs.

The problem of **filtering** or belief-state **monitoring** is to determine the current state based on the current and previous observations, namely to determine

$$P(S_i | O_0, \dots, O_i).$$

The problem of **smoothing** is to determine a state based on past and future observations.

Suppose an agent has observed up to time k and wants to determine the state at time i for $i < k$; the smoothing problem is to determine

$$P(S_i | O_0, \dots, O_k).$$

All of the variables S_i and V_i for $i > k$ can be ignored.

Q7. Explain Dynamic Belief Networks*Ans :***Dynamic Belief Networks**

A dynamic belief network (DBN) is a belief network with regular repeated structure. It is like a (hidden) Markov model, but the states and the observations are represented in terms of features. Assume that time is discrete. If F is a feature, we write F_t as the random variable that represented the value of variable F at time t . A dynamic belief network makes the following assumptions:

- The set of features is the same at each time.
- For any time $t > 0$, the parents of variable F_t are variables at time t or time $t-1$, such that the graph for any time is acyclic. The structure does not depend on the value of t (except $t = 0$ is a special case).
- The conditional probability distribution of how each variable depends on its parents is the same for every time $t > 0$.

Thus, a dynamic belief network specifies a belief network for time $t=0$, and for each variable F_t specifies $P(F_t | \text{parents}(F_t))$, where the parents of F_t are in the same or previous time step. This is specified for t as a free parameter; the conditional probabilities can be used for any time $t > 0$. As in a belief network, directed cycles are not allowed.

The model for a dynamic belief network can be represented as a two-step belief network that represents the variables at the first two times (times 0 and 1). That is, for each feature F there are two variables, F_0 and F_1 ; $\text{parents}(F_0)$ only include variables for time 0, and $\text{parents}(F_1)$ can be variables at time 0 or 1, as long as the resulting graph is acyclic. Associated with the network are the probabilities $P(F_0 | \text{parents}(F_0))$ and $P(F_1 | \text{parents}(F_1))$. Because of the repeated structure, $P(F_i | \text{parents}(F_i))$, for $i > 1$, has exactly the same structure and the same conditional probability values as $P(F_1 | \text{parents}(F_1))$.

Example

Suppose the trading agent wants to model the dynamics of the price of a commodity such as printer paper. To represent this domain, the designer models what variables affect the price and the other variables. Suppose the cost of pulp and the transportation costs directly affect the price of paper. The transportation costs are affected by the weather. The pulp cost is affected by the prevalence of tree pests, which in turn depend on the weather. Suppose that each variable depends on the values of the previous time step. A two-stage dynamic belief network representing these dependencies is shown in Figure.

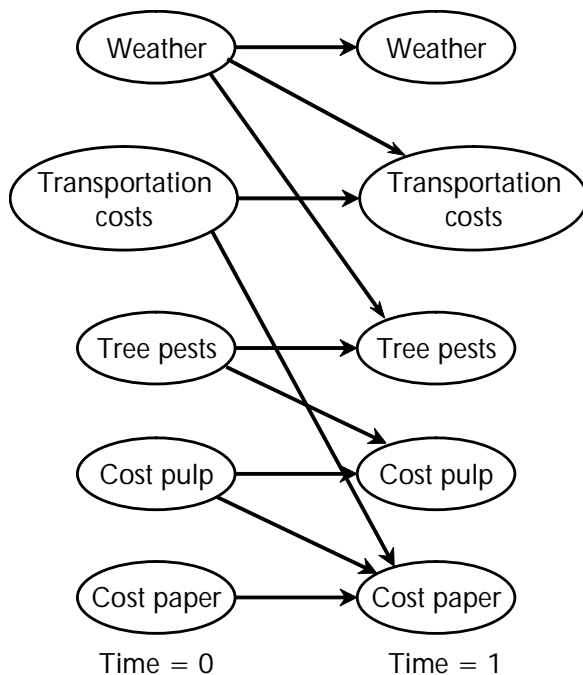


Fig.: Two-stage dynamic belief network for paper pricing

Note that, in this figure, the variables are initially independent.

This two-stage dynamic belief network can be expanded into a regular dynamic belief network by replicating the nodes for each time step, and the parents for future steps are a copy of the parents for the time 1 variables.

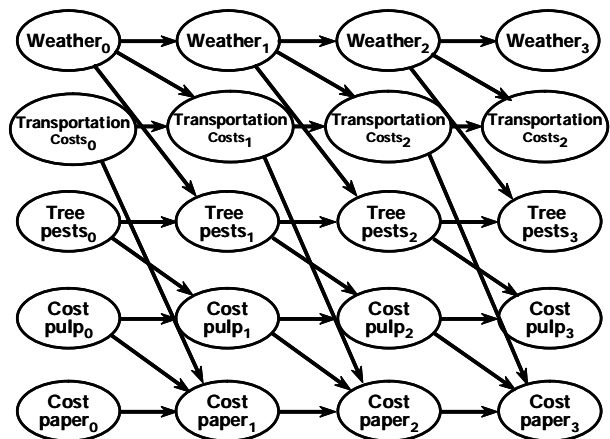


Fig.: Expanded dynamic belief network for paper pricing

An expanded belief network is shown in Figure. The subscripts represent the time that the variable is referring to.

3.1.5 Forward and Backward Reasoning

Q8. Write about forward and backward reasoning.

Ans :

Forward chaining

Forward Chaining is one of the two main methods of reasoning when using an inference engine and can be described logically as repeated application of modus ponens. Forward chaining is a popular implementation strategy for expert systems, business and production rule systems. Forward chaining starts with the available data and uses inference rules to extract more data (from an end user, for example) until a goal is reached. An inference engine using forward chaining searches the inference rules until it finds one where the antecedent (If clause) is known to be true. When such a rule is found, the engine can conclude, or infer, the consequent (Then clause), resulting in the addition of new information to its data.

A horn clause C is called definite if it contains exactly one positive literal, i.e., implications of type

$$\left(\bigvee_{i=1}^n \neg L_i \right) \Rightarrow \left(\bigvee_{i=1}^n L_i \rightarrow \text{false} \right)$$

are not possible.

If the knowledge base consists of Horn clauses only, then generalized modus ponens can be used just like modus ponens to infer statements iteratively by forward chaining

Example

The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American. Prove that Col. West is a criminal.

The law says that it is a crime for an American to sell weapons to hostile nations.

$\forall \text{ American}(x) \wedge \text{Weapon}(y) \wedge \text{Hostile}(z) \wedge \text{Sell}(x, y, z) \rightarrow \text{Criminal}(x)$

The country Nono,
Country (Nono)
an enemy of America,
 $\text{Enemy}(\text{Nono}, \text{America})$
has some missiles,

$\exists x \text{ Missile}(x) \wedge \text{Owns}(\text{Nono}, x)$

and all of its missiles were sold to it by Colonel West,

$\forall \text{ Missile}(x) \wedge \text{Owns}(\text{Nono}, x) \rightarrow \text{Sell}(\text{West}, x, \text{Nono})$

who is American.
 $\text{American}(\text{West})$

Additional background knowledge

Missiles are weapons.

$\forall x \text{ Missile}(x) \rightarrow \text{Weapon}(x)$

Enemies of America are hostile.

$\forall x \text{ Enemy}(x, \text{America}) \rightarrow \text{Hostile}(x)$

Prove that Col. West is a criminal

$\text{Criminal}(\text{West})?$

The knowledge base can be simplified by

- Existential instantiation and
- Omitting universal quantifiers

(as all free variables are universally quantified any way).

$\text{American}(x) \wedge \text{Weapon}(y) \wedge \text{Hostile}(z) \wedge \text{Sell}(x, y, z) \rightarrow \text{Criminal}(x)$

$\text{Country}(\text{Nono})$

$\text{Enemy}(\text{Nono}, \text{America})$

$\text{Missile}(M_1) \wedge \text{Owns}(\text{Nono}, M_1)$

$\text{Missile}(x) \wedge \text{Owns}(\text{Nono}, x) \rightarrow \text{Sell}(\text{West}, x, \text{Nono})$

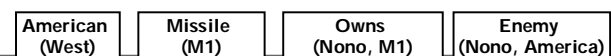
$\text{American}(\text{West})$

$\text{Missile}(x) \rightarrow \text{Weapon}(x)$

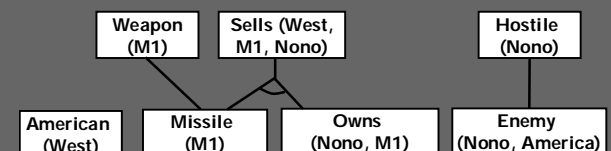
$\text{Enemy}(x, \text{America}) \rightarrow \text{Hostile}(x)$

Step 1

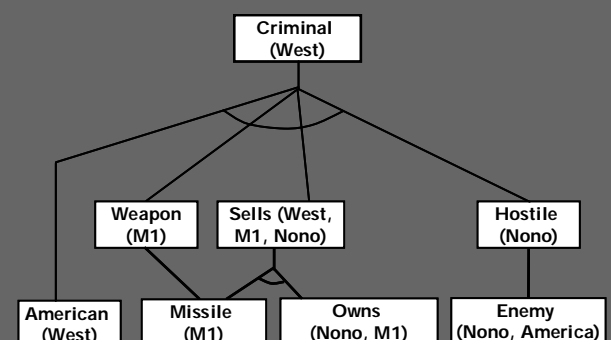
Forward Chaining/Example



Step 2



Step 3



Finally from the given facts we proved that Col. West is Criminal by using Forward Chaining algorithm.

Backward Chaining

Backward chaining (or **backward reasoning**) is an inference method that can be described (in lay terms) as working backward from the goal(s). It is used in automated theorem provers, inference engines, proof assistants and other artificial intelligence applications.

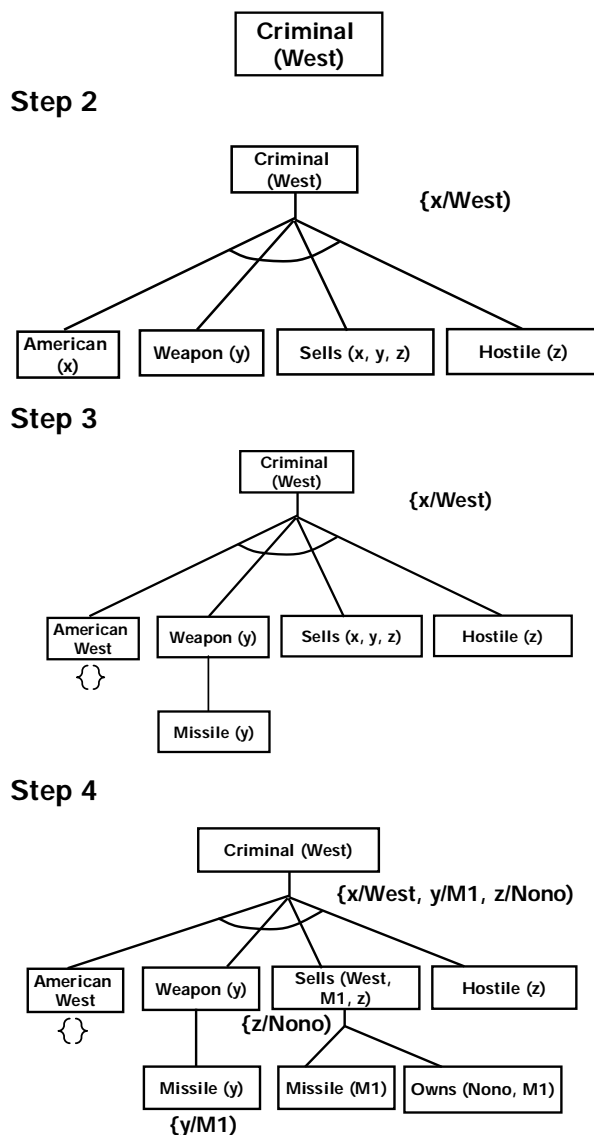
In game theory, its application to (simpler) subgames in order to find a solution to the game is called backward induction. In chess, it is called retrograde analysis, and it is used to generate tablebases for chess endgames for computer chess.

Backward chaining is implemented in logic programming by SLD resolution. Both rules are based on the modus tollens inference rule. It is one of the two most commonly used methods of reasoning with inference rules and logical implications – the other is forward chaining. Backward chaining systems usually employ a depth-first search strategy

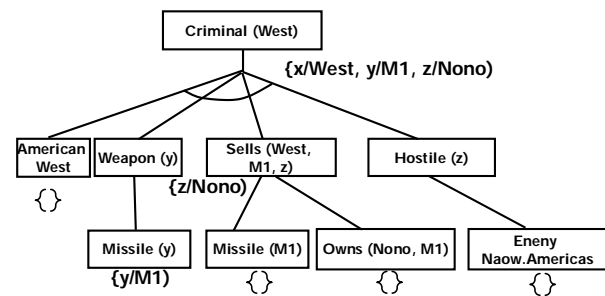
Let us consider the same example and will prove Col. West is Criminal using Backward Chaining.

Step 1

Backward Chaining/Example



Step 5



Finally we proved Col. West is criminal using backward chaining algorithm.

3.1.6 Perception Making Simple Decisions

Q9. How perception is used in making simple decisions.

Ans :

Perception is an essential component of intelligent behavior. We perceive the world around us through five basic senses of sight, hearing, touch, smell, and taste. Of these, sight and hearing have been the main areas of Artificial Intelligence research leading to speech understanding. When we perceive some signal, it may be a sound or light. We respond appropriately to that signal. To produce an appropriate response we must categorize or analyze that signal. For example, to analyze a sentence we must first identify individual sounds, then combine these sounds into words, and then combine words into a meaningful sentence structure. But this is hard because dividing sounds into words needs additional knowledge or information about the situation. A series of sounds may be interpreted in many ways. For instance

"Tigers care their kids"

and "Tiger scare their kids"

might both have been the possible interpretations of the same series of sounds.

To overcome the perceptual problems in speech understanding, the process of analyzing a speech is divided into five stages.

1. Digitization

The continuous input is divided into discrete chunks in speech. The division is done on a time scale and in images, it may be based on color or area or tint.

2. Smoothing

Since the real world is usually continuous, large spikes and variation in the input is avoided.

3. Segmentation

Group the smaller chunks produced by digitization into larger chunks corresponding to logic components of the signal. For speech understanding segments correspond to individual sounds called phonemes.

4. Labeling

Each segment is given a label.

5. Analysis

The labeled segments are put together to form a coherent object.

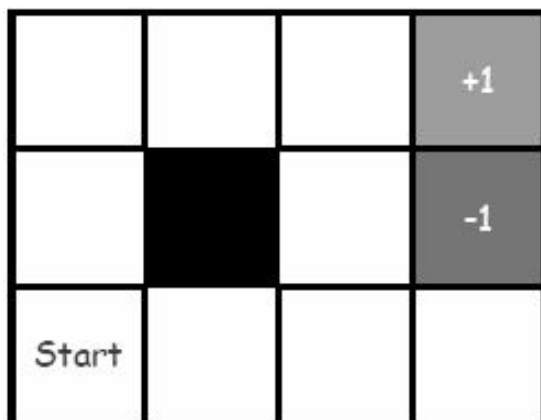
3.1.7 Making Complex Decisions

Q10. Explain how complex decisions are made in AI.

Ans :

The agent's utility now depends on a sequence of decisions.

- In the following 4×3 grid environment the agent makes a decision to move (U, R, D, L) at each time step
- When the agent reaches one of the goal states, it terminates
- The environment is fully observable - the agent always knows where it is.



If the environment were deterministic, a solution would be easy: the agent will always reach +1 with moves [U, U, R, R, R]

- Because actions are unreliable, a sequence of moves will not always lead to the desired outcome
- Let each action achieve the intended effect with probability 0.8 but with probability 0.1 the action moves the agent to either of the right angles to the intended direction
- If the agent bumps into a wall, it stays in the same square.

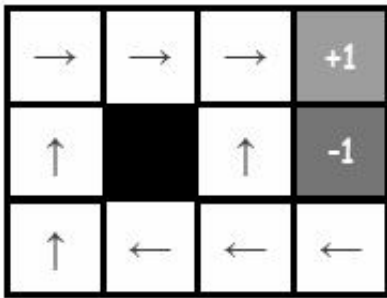
- Now the sequence [U, U, R, R, R] leads to the goal state with probability $0.85 = 0.32768$

- In addition, the agent has a small chance of reaching the goal by accident going the other way around the obstacle with a probability 0.14×0.8 , for a grand total of 0.32776.

For our particular example, the reward is -0.04 in all states except in the terminal states.

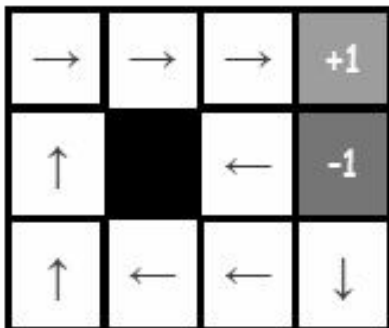
- The utility of an environment history is just (for now) the sum of rewards received.
- If the agent reaches the state +1, e.g., after ten steps, its total utility will be 0.6.
- The small negative reward gives the agent an incentive to reach [4, 3] quickly.
- A sequential decision problem for a fully observable environment with a Markovian transition model and Additive rewards is called a Markov decision problem (MDP).

- Each time a given policy P is executed starting from the initial state, the stochastic nature of the environment will lead to a different environment history.
- The quality of a policy Π is therefore measured by the expected utility of the possible environment histories generated by the policy
- An optimal policy Π yields the highest expected utility of the possible environment histories generated by the policy.

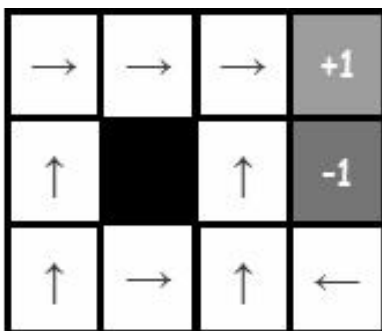


- A policy represents the agent function explicitly and is therefore a description of a simple reflex agent.

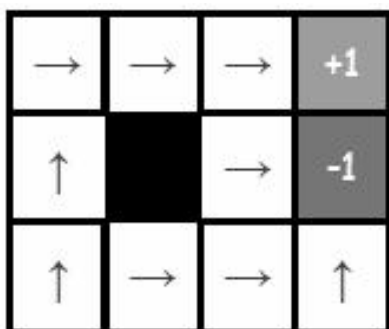
– $-0.0221 < R(S) < 0$:



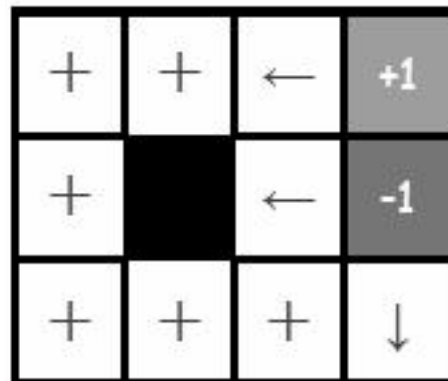
– $-0.4278 < R(S) < -0.0850$:



$R(s) < -1.6284$:



$R(S) > 0$



Utilities over time

- In case of an *infinite horizon* the agent's action time has no upperbound
- With a finite time horizon, the optimal action in a given state could change over time - the optimal policy for a finite horizon is *non-stationary*
- With no fixed time limit, on the other hand, there is no reason to behave differently in the same state at different times, and the optimal policy is stationary
- The *discounted* utility of a state sequence s_0, s_1, s_2, \dots is $R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots$, where $0 < \gamma \leq 1$ is the discount factor
When $\gamma = 1$, discounted rewards are exactly equivalent to additive rewards
- The latter rewards are a special case of the former ones. When γ is close to 0, rewards in the future are viewed as insignificant.
- If an infinite horizon environment does not contain a terminal state or if the agent never reaches one, then all environment histories will be infinitely long.
- Then, utilities with additive rewards will generally be infinite.
- With discounted rewards ($\gamma < 1$), the utility of even an infinite sequence is finite.

Value Iteration

For calculating an optimal policy we calculate the utility of each state and then use the state utilities to select an optimal action in each state.

- The utility of a state is the expected utility of the state sequence that might follow it.
- Obviously, the state sequences depend on the policy P that is executed.
- Let s_t be the state the agent is in after executing P for t steps.
- Note that s_t is a random variable.
- Then, executing starting in $s(=s_0)$ we have

$$U^{\pi}(s) = E[\sum_{t=0, \dots, \infty} \gamma^t R(s_t)]$$

The true utility of a state $U(s)$ is just $U^{\pi^*}(s)$

- $R(s)$ is the short-term reward for being in s , whereas $U(s)$ is the long-term total reward from s onwards
- In our example grid the utilities are higher for states closer to the +1 exit, because fewer steps are required to reach the exit

0.812	0.868	0.912	+1
0.762		0.660	-1
0.705	0.655	0.611	0.388

3.1.8 Other Techniques in Uncertainty and Reasoning Process

Q11. Explain Dempster – Shafer Theory.

Ans :

An alternative to Bayesian Networks is Dempster - Shafer Theory which is designed to deal directly with the distinction between uncertainty and ignorance. Rather than computing probabilities of propositions, it computes probabilities that evidence supports the propositions. This theory is an approach to combining evidence. Each fact has a degree of support, between 0 and 1:

- 0 No support for the fact
- 1 full support for the fact

Set of possible conclusions

Bayes Theorem concerned with evidence that supported single conclusions. (eg. evidence for each outcome in)

D-S theory concerned with evidences which support subsets of outcome in

Frame of Discernment

The Frame of Discernment or power set of is the set of all possible subsets of

Then the Frame of Discernment of is: the empty set, has a probability of 0, Since one of the outcome has to be true. Each of the other elements in the powerset has a probability between 0 and 1.

The probability of is 1.0. Since one has to be true.

Mass function : $m(A)$: (where A is a member of the power set) = proportion of all evidence that supports this element of the power set.

“The mass $m(A)$ of a given member of the power set, A , expresses the proportion of all relevant and available evidence that supports the claim that the actual state belongs to A but to no particular subset of A .”

“The value of $m(A)$ pertains *only* to the set A and makes no additional claims about any subsets of A , each of which has, by definition, its own mass.

Belief function: $bel(A)$

The **belief** in an element A of the Power set is the sum of the masses of elements which are subsets of A (including A itself).

E.g., given $A = \{q_1, q_2, q_3\}$

$$Bel(A) = m(q_1) + m(q_2) + m(q_3) + m(\{q_1, q_2\}) + m(\{q_2, q_3\}) + m(\{q_1, q_3\}) + m(\{q_1, q_2, q_3\})$$

Disbelief (or Doubt) in A : $dis(A)$

The disbelief in A is simply $bel(\neg A)$.

It is calculated by summing all masses of elements which do not intersect with A .

Plausibility of A: $pl(A)$

The plausability of an element A, $pl(A)$, is the sum of all the masses of the sets that intersect with the set A

Let us consider an example,

There are four people named as (B, J, S, K) are locked in room when the lights go out. When the lights come on, K is dead, stabbed with a knife. Not suicide (stabbed in the back). No-one entered the room. Assume only one killer.

Power Set

Detectives, after reviewing the crime-scene, assign mass probabilities to various elements of the power set:

Belief Function

Given the mass assignments as assigned by the detectives:

$$bel(\{B\}) = m(\{B\}) = 0.1$$

$$\begin{aligned} bel(\{B,J\}) &= m(\{B\}) + m(\{J\}) + m(\{B,J\}) \\ &= 0.1 + 0.2 + 0.1 = 0.4 \end{aligned}$$

Result

Plausibility of A: $pl(A)$

As we know that, The plausability of an element A, $pl(A)$, is the sum of all the masses of the sets that intersect with the set A:

$$\begin{aligned} \text{E.g: } pl(\{B, J\}) &= m(B) + m(J) + m(B, J) \\ &\quad + m(B,S) + m(J,S) + m(B,J,S) \\ &= 0.9 \end{aligned}$$

Disbelief (or Doubt) in A: $dis(A)$

Now, we need to find the disbelief fn for that we need to calculate $dis(A)$, by summing all masses of elements which do not intersect with A.

Then, the plausibility of A is thus $1 - dis(A)$:

$$pl(A) = 1 - dis(A)$$

Belief Intervals and Probability

The probability in A falls somewhere between $bel(A)$ and $pl(A)$. $bel(A)$ represents the evidence we have for A directly.

So $prob(A)$ cannot be less than this value.

- $pl(A)$ represents the maximum share of the evidence we could possibly have, if, for all sets that intersect with A, the part that intersects is actually valid. So $pl(A)$ is the maximum possible value of $prob(A)$.

Belief Intervals

Belief intervals allow Detspster-Shafer theory to reason about the degree of certainty or certainty of our beliefs.

- A small difference between belief and plausibility shows that we are certain about our belief.
- A large difference shows that we are uncertain about our belief.
- However, even with a 0 interval, this does not mean we know which conclusion is right. Just how probable it is!

3.2 PLANNING PROBLEM

Q12. What is planning system Explain about it with the help of block world puzzle example.

Ans :

Planning refers to the process of computing several steps of a problem solving before executing any of them. Planning is useful as a problem solving technique for non decomposable problem.

Components of Planning System

In any general problem solving systems, elementary techniques to perform following functions are required

- Choose the best rule (based on heuristics) to be applied
 - Apply the chosen rule to get new problem state
 - Detect when a solution has been found
 - Detect dead ends so that new directions are explored.
- To choose the rules,
- First isolate a set of differences between the desired goal state and current state,
 - Identify those rules that are relevant to reducing these difference,

- If more rules are found then apply heuristic information to choose out of them.

To apply rules, In simple problem solving system,

- Applying rules was easy as each rule specifies the problem state that would result from its application.
- In complex problem we deal with rules that specify only a small part of the complete problem state.

Example

Let us consider the famous problem name as Block World Problem, which helps to understand the importance of planning in artificial intelligent system.

The block world environment has,

- Square blocks of same size
- Blocks can be stacked one upon another.
- Flat surface (table) on which blocks can be placed.
- Robot arm that can manipulate the blocks. It can hold only one block at a time.

In block world problem, the state is described by a set of predicates representing the facts that were true in that state. One must describe for every action, each of the changes it makes to the state description. In addition, some statements that everything else remains unchanged is also necessary. We are having four types of operations done by robot in block world environment. They are

UNSTACK (X, Y): [US (X, Y)]

- Pick up X from its current position on block Y. The arm must be empty and X has no block on top of it.

STACK (X, Y): [S (X, Y)]

- Place block X on block Y. Arm must holding X and the top of Y is clear.

PICKUP (X): [PU (X)]

- Pick up X from the table and hold it. Initially the arm must be empty and top of X is clear.

PUTDOWN (X): [PD (X)]

- Put block X down on the table. The arm must have been holding block X.

Along with the operations, some predicates to be used to describe an environment clearly. Those predicates are,

- ON(X, Y) - Block X on block Y.
- ONT() - Block X on the table.
- CL(X) - Top of X clear.
- HOLD(X) - Robot-Arm holding X.
- AE - Robot-arm empty.

Logical statements true in this block world.

X Holding X means, arm is not empty

$$(\exists X) \text{HOLD}(X) \rightarrow \text{AE}$$

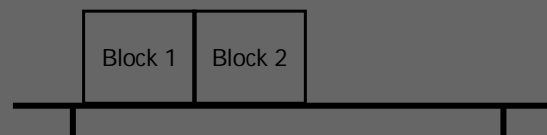
X is on a table means that X is not on the top of any block

$$(\forall X) \text{ONT}(X) \rightarrow (\exists Y) \text{ON}(X, Y)$$

Any block with no block on has clear top

$$(\forall X) (\sim (\exists Y) \text{ON}(Y, X)) \rightarrow \text{CL}(X)$$

Initial State



Arm empty

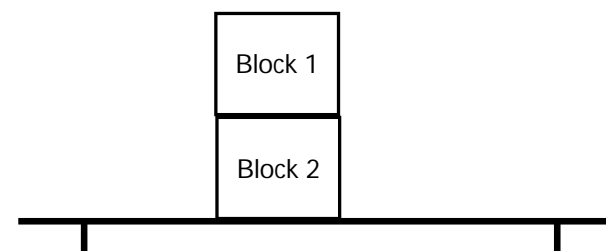
clear(block2)

ontable(block2)

ontable(block1)

clear(block1)

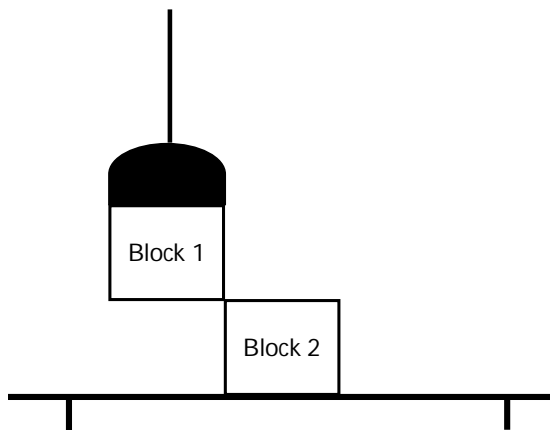
Goal State



Arm empty
 ontable(block2)
 on(block1, block2)
 clear(block1)

We have to generate a plan to reach goal state from initial state given. In this example the initial state has two blocks Block1 and Block 2. Both is placed on table. To reach the goal state first we have to

PICKUP(Block 1)



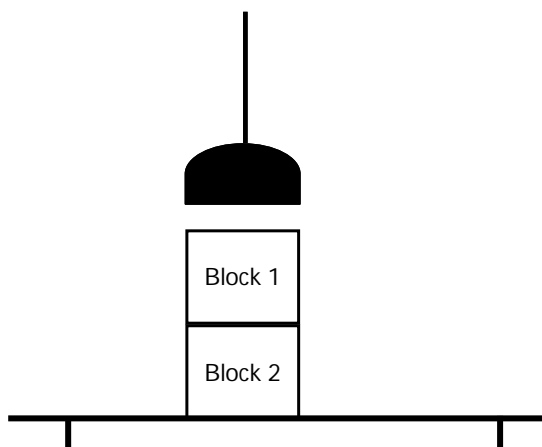
We need to check whether we reach goal state or not, after completion of each and every operation. Here the environment looks like,

Hold(block1)

Clear(Block2)

OnTable(Block2)

This is not the goal state. so, we have to continue the process. Next the block 1 needs to be place on block 2, to achieve this do the operation **STACK (Block1, Block2)**. After this operation the environment looks like,



Arm Empty, on (Block1, Block2), Clear (Block1), On Table (Block2)

We reach the goal state, the plan for reaching goal state is PICKUP(Block1) and Stack (Block1, Block2).

3.2.1 Simple Planning Agent

Q13. Write a note on simple planning agent.

Ans :

Earlier we saw that problem-solving agents are able to plan ahead - to consider the consequences of sequences of actions - before acting. We also saw that a knowledgebased agents can select actions based on explicit, logical representations of the current state and the effects of actions. This allows the agent to succeed in complex, inaccessible environments that are too difficult for a problem-solving agent.

Problem Solving Agents + Knowledge-based Agents = Planning Agents

In this module, we put these two ideas together to build planning agents. At the most abstract level, the task of planning is the same as problem solving.

Algorithm of a simple planning agent:

1. Generate a goal to achieve.
2. Construct a plan to achieve goal from current state.
3. Execute plan until finished.
4. Begin again with new goal.

3.2.2 Planning Languages

Q14. Write about Artificial intelligence planning language STRIPS.

Ans :

To represent planning problems we use **Artificial Intelligence planning languages** that describe environment's conditions which then lead to desired goals by generating chain of actions based on these conditions.

STRIPS

STRIPS is an action language which was a part of the first major planning system with the same name.

STRIPS stands for "Stanford Research Institute Problem Solver," was the planner used in Shakey, one of the first robots built using AI technology, which is an action-centric representation, for each action, specifies the effect of an action.

STRIPS as a classical planning language is composed from states, goals and set of actions:

- **State** is a conjunction of positive literals which cannot contain variables and invoke functions.
- **Goal**, similarly to the state, is conjunction of positive and ground (no variables and no functions) literals.
- **Actions** (also called operators) include preconditions and postconditions. Both represented as a conjunction of function-free literals. Preconditions describe the state of world required to perform action, while postconditions describe state of the world after action is executed.

The STRIPS representation for an action consists of three lists,

- Pre_Cond list contains predicates which have to be true before operation.
- ADD list contains those predicates which will be true after operation
- DELETE list contain those predicates which are no longer true after operation

Predicates not included on either of these lists are assumed to be unaffected by the operation. Frame axioms are specified implicitly in STRIPS which greatly reduces amount of information stored.

Example from Shakey the robot paper can be helpful with understanding the basics of STRIPS language. It describes task of fetching box from adjacent room.

Initial state of the world is presented by this image:

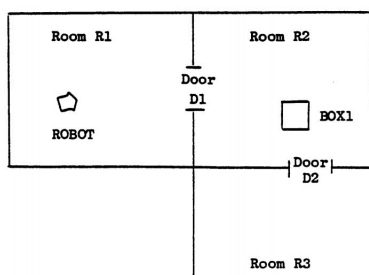


Image from **Shakey the robot paper, STRIPS**

Note: Capital letters are constants, while small letters are variable.

As mentioned before, action can be applied only if current state of the world meets all of its preconditions. When it's applied, literals from postcondition are: added to world state if they are positive, removed from world state if they are negative.

Here is the solution (sequence of actions to achieve the goal, while starting from initial state) for described example:

1. GOTHRU(D1, R1, R2)
2. PUSHTHRU(BOX1, D1, R2, R1).

3.2.3 Blocks World

Q15. Explain Blocks World problem with the help of STRIPS.

Ans :

Let us discuss about the action lists for operations of block world problem.

Stack (X, Y)

Pre: CL (Y), HOLD (X)

Del: CL (Y), HOLD (X)

Add: AE, ON (X, Y)

UnStack (X, Y)

Pre: ON (X, Y), CL (X), AE

Del: ON (X, Y), AE

Add: HOLD (X), CL (Y)

Pickup (X)

Pre: ONT (X) , CL (X) ,AE

Del: ONT (X), AE

Add: HOLD (X)

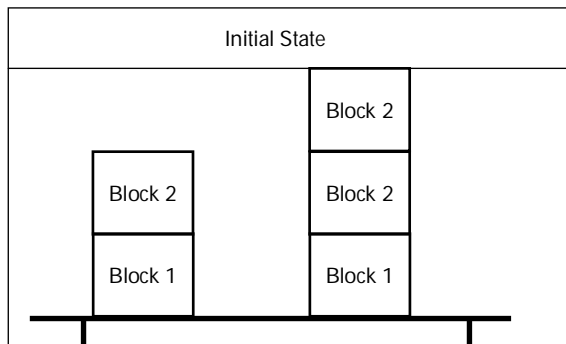
Putdown (X)

Pre: HOLD (X)

Del: HOLD (X)

Add: ONT (X) AE

Consider a Block world problem,



Initial State

on(block2, block1)

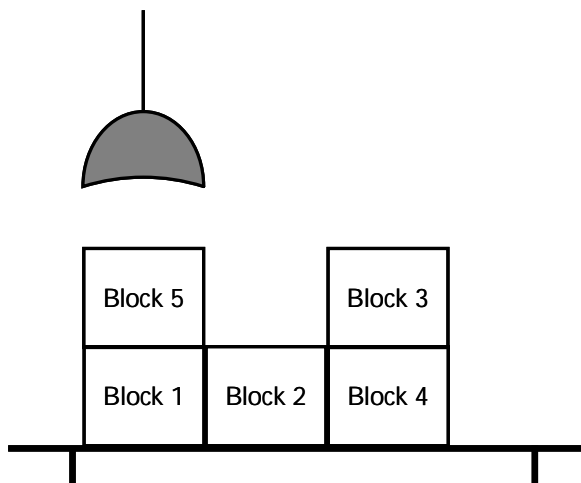
clear(block2)

ontable(block3)

on(block4, block3)

on(block5, block4)

clear(block5)



Goal State

empty

on(block3, block4)

on(block5, block1)

ont(block2)

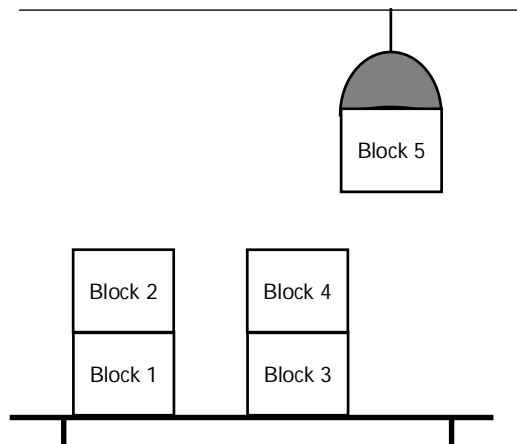
Identify the style operators for given problem,

Unstack(block5,block4)

Pre: ON (block5, block4), CL (block5), AE

Del: ON (block5,block4) , AE

Add: HOLD (block5), CL (block4)

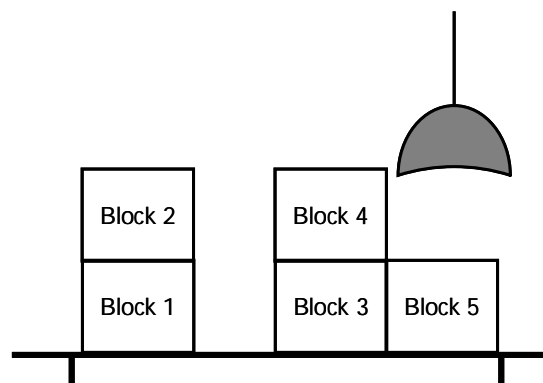


Putdown (block 5)

Pre: HOLD (block5)

Del: HOLD (block5)

Add: ONT (block5) AE

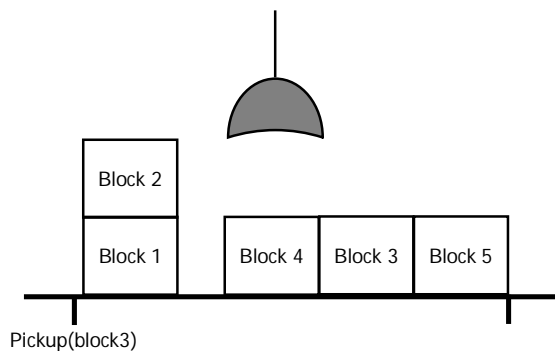


Unstack(block4, block3)

Pre: ON (block4, block3), CL(block 4), AE

Del: ON (block4, block3), AE

Add: HOLD (block4), CL (block 3)



Pre: ONT (block3), CL (block3), AE

Del: ONT (block3), AE

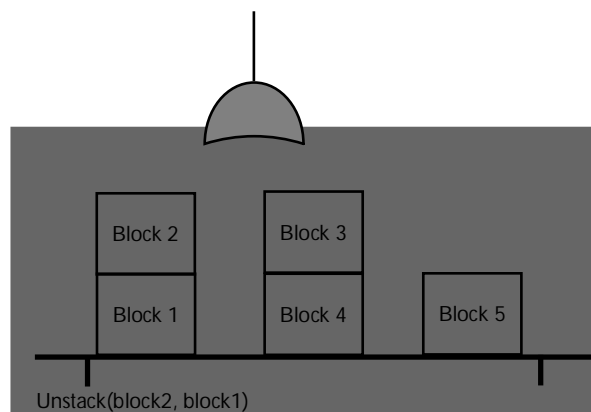
Add: HOLD (block3)

Stack(block3, block4)

Pre: CL (block3) HOLD (block3)

Del: CL (block4) HOLD (block3)

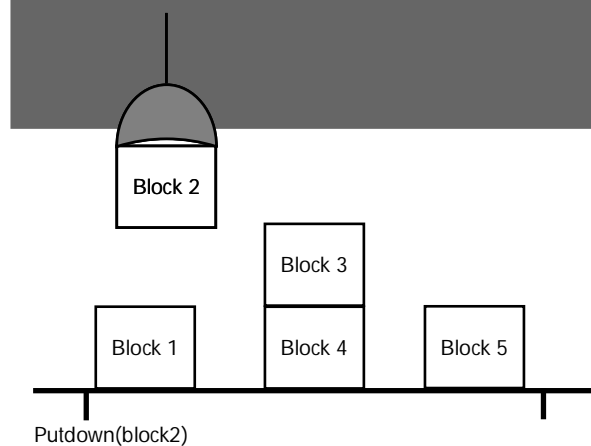
Add: AE ON (block3, block4)



Pre: ON (block4, block3), CL (block4), AE

Del: ON (block4, block3), AE

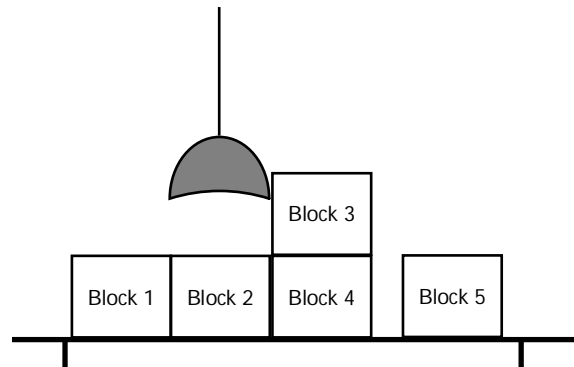
Add: HOLD (block4), CL (block3)



Pre: HOLD (block4)

Del: HOLD (block4)

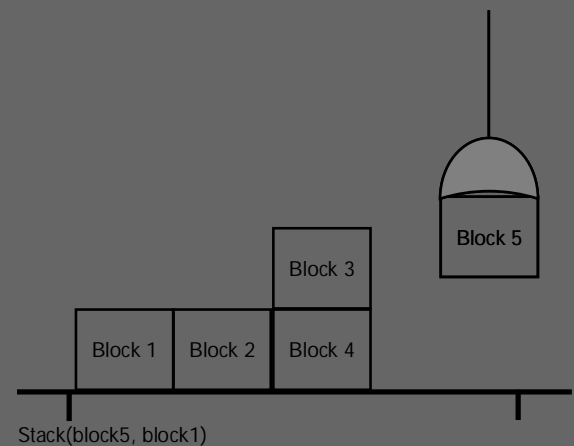
Add: ONT (block4) AE



Pre: ONT (block5), CL (block5), AE

Del: ONT (block5), AE

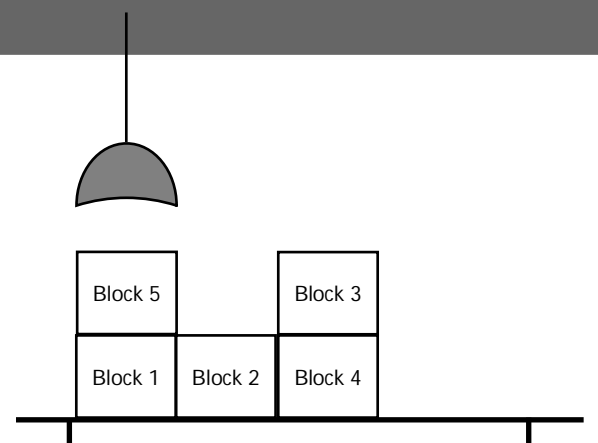
Add: HOLD (block5)



Pre: CL (block1) HOLD (block5)

Del: CL (block1), HOLD (block5)

Add: AE, ON (block5, block1)



After completing all the operations what we found for the given problem, we had reached the goal state.

armempty
on(block3, block4)
on(block5, block1)
ont(block2)

3.2.4 Goal Stack Planning Means-ends Analysis

Q16. Write Planning-Goal Stack Algorithm.

Ans :

Planning-Goal Stack Algorithm

One of the earliest techniques is planning using goal stack. Problem solver uses single stack that contains.

- Sub goals and operators both
- Sub goals are solved linearly and then finally the conjoined sub goal is solved.

Plans generated by this method will contain complete sequence of operations for solving one goal followed by complete sequence of operations for the next etc.

Problem solver also relies on

- A database that describes the current situation.
- Set of operators with precondition, add and delete lists.

Let us assume that the goal to be satisfied is:

$$\text{GOAL} = G1 \perp G2 \perp \dots \perp Gn$$

Sub-goals $G1, G2, \dots Gn$ are stacked with compound goal $G1 \wedge G2 \wedge \dots \wedge Gn$ at the bottom.

Top	$G1$
	$G2$
	\vdots
	Gn
Bottom	$G1 \perp G2 \perp \dots \perp Gn$

At each step of problem solving process, the top goal on the stack is pursued.

Algorithm

Find an operator that satisfies sub goal $G1$ (makes it true) and replace $G1$ by the operator.

- If more than one operator satisfies the sub goal then apply some heuristic to choose one.

In order to execute the top most operation, its preconditions are added onto the stack.

- Once preconditions of an operator are satisfied, then we are guaranteed that operator can be applied to produce a new state.

- New state is obtained by using ADD and DELETE lists of an operator to the existing database.

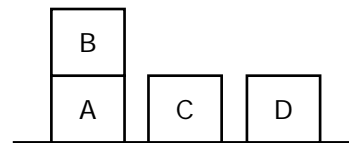
Problem solver keeps track of operators applied.

- This process is continued till the goal stack is empty and problem solver returns the plan of the problem.

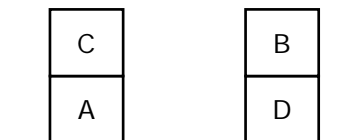
Goal Stack Example

With this example, let us explain the working method of Goal Stack Algorithm.

Initial State



Goal State



Initial State

$$\text{ON}(B, A) \wedge \text{ONT}(C) \perp \text{ONT}(A) \perp \text{ONT}(D) \perp \text{CL}(B) \wedge \text{CL}(C) \wedge \text{CL}(D) \wedge \text{AE}$$

Goal State

$$\text{ON}(C, A) \perp \text{ON}(B, D) \perp \text{ONT}(A) \perp \text{ONT}(D) \perp \text{CL}(C) \perp \text{CL}(B) \perp \text{AE}$$

We notice that following sub-goals in goal state are also true in initial state.

$$\text{ONT}(A) \wedge \text{ONT}(D) \perp \text{CL}(C) \perp \text{CL}(B) \perp \text{AE}$$

Represent for the sake of simplicity -
TSUBG

Only sub-goals $ON(C, A)$ & $ON(B, D)$ are to be satisfied and finally make sure that **TSUBG** remains true.

Either start solving first **$ON(C, A)$ or $ON(B, D)$** . Let us solve first $ON(C, A)$.

Goal Stack

$ON(C, A)$

$ON(B, D)$

$ON(C, A) \perp ON(B, D) \wedge \text{TSUBG}$

$S(C, A)$ can be applied if its preconditions are true. So add its preconditions on the stack.

Goal Stack

HOLD(C) Preconditions of STACK

CL(A)

$CL(A) \perp \text{HOLD}(C)$

S (C, A) Operator

$ON(B, D)$

$ON(C, A) \perp ON(B, D) \wedge \text{TSUBG}$

To do the $S(C, A)$ operation all preconditions should be true. In the given problem $CL(A)$ is not true. So, to make the state true, replace $CL(A)$ by **$U(B, A)$** and write the preconditions of Unstack operator.

Goal Stack

$ON(B, A)$

CL(B) Preconditions of UNSTACK

AE

$ON(B, A) \wedge CL(B) \perp \text{AE}$

US(B, A) Operator

HOLD(C) Preconditions of STACK

$CL(A) \wedge \text{HOLD}(C)$

S (C, A) Operator

$ON(B, D)$

$ON(C, A) \wedge ON(B, D) \wedge \text{TSUBG}$

➤ $ON(B, A)$, $CL(B)$ and AE are all true in initial state, so pop these along with its compound goal.

➤ Next pop top operator $US(B, A)$ and produce new state by using its ADD and DELETE lists.

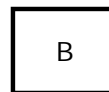
➤ Add $US(B, A)$ in a queue of sequence of operators.

SQUEUE = US (B, A)

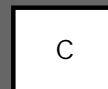
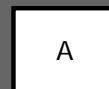
State-1

$ONT(A) \perp ONT(C) \perp ONT(D) \perp \text{HOLD}(B)$
 $\perp CL(A) \perp CL(C) \perp CL(D)$

STATE 1



Unstack (B, A)



Goal Stack

HOLD(C) Preconditions of STACK

$CL(A) \wedge \text{HOLD}(C)$

S (C, A) Operator

$ON(B, D)$

$ON(C, A) \wedge ON(B, D) \wedge \text{TSUBG}$

To execute the **$S(C, A)$** , all the preconditions of Stack operator should be true. But in this case $\text{HOLD}(C)$ is not true. To make the state true use the operator **$S(B, D)$**

S(B, D) Operator

HOLD(C)

$CL(A) \wedge \text{HOLD}(C)$ Preconditions of STACK

S(C, A) Operator

$ON(B, D)$

$ON(C, A) \wedge ON(B, D) \wedge \text{TSUBG}$

Write down the preconditions of **$S(B, D)$**

Goal Stack

$CL(D) \perp \text{HOLD}(B)$ Preconditions of STACK

S(B, D) Operator

HOLD(C)

$CL(A) \perp HOLD(C)$ Preconditions of STACK

S (C, A) Operator

$ON(B, D)$

$ON(C, A) \wedge ON(B, D) \wedge \mathbf{TSUBG}$

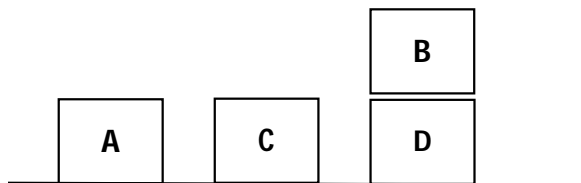
Add S(B, D) in a queue of sequence of operators.

SQUEUE = US (B, A), S (B, D)

State 2

$ONT(A) \wedge ONT(C) \perp ONT(D) \perp ON(B, D) \perp CL(A) \perp CL(C) \perp CL(B) \perp AE$

State 2



Goal Stack

HOLD(C)

$CL(A) \perp HOLD(C)$ Preconditions of STACK

S (C, A) Operator

$ON(B, D)$

$ON(C, A) \wedge ON(B, D) \wedge \mathbf{TSUBG}$

To execute **S(C,A)** all the preconditions should be true. here HOLD(C) is not true, to make the state true use the operator **PU(C)** and write the preconditions.

Goal Stack

$ONT(C) \wedge CL(C) \wedge AE$ Preconditions of PICKUP

PU(C) Operator

HOLD(C)

$CL(A) \wedge HOLD(C)$ Preconditions of STACK

S (C, A) Operator

$ON(B, D)$

$ON(C, A) \wedge ON(B, D) \wedge \mathbf{TSUBG}$

Here, all the preconditions of PU operator is true, so add **PU(C)** in a queue of sequence of operators.

SQUEUE = US (B, A), S(B, D), PU(C)

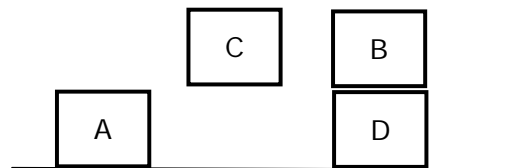
State_3

$ONT(A) \wedge HOLD(C) \wedge ONT(D) \wedge ON(B, D) \wedge CL(A) \wedge CL(B).$

Pick (C)

STATE 3

Pick (C)



Goal Stack

HOLD(C)

$CL(A) \wedge HOLD(C)$ Preconditions of STACK

S (C, A) Operator

$ON(B, D)$

$ON(C, A) \wedge ON(B, D) \wedge \mathbf{TSUBG}$

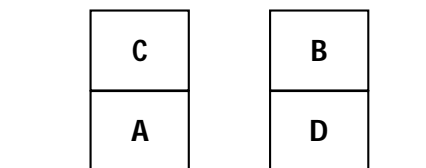
Here all the preconditions of **S(C, A)** is true, so add **S(C, A)** in queue

SQUEUE = US(B, A), S (B, D), PU(C), S(C, A)

State - 4

$ONT(A) \wedge ON(C, A) \wedge ONT(D) \wedge ON(B, D) \wedge CL(C) \wedge CL(B) \wedge AE$

Goal State



Finally, we reached goal state after **S(C,A)** using Goal Stack algorithm, so the plan for the given problem is,

UnStack (B, A)

Stack (B, D)

PickUp (C)

Stack (C, A)

3.2.5 Planning as a State-space Search

Q17. Write about planning as state-space search.

Ans :

Planning as Search

There are two main approaches to solving planning problems, depending on the kind of search space that is explored:

1. Situation-space search
2. Planning-space search in situation space search

In Situation-Space search

- The search space is the space of all possible states or situations of the world
- Initial state defines one node
- A goal node is a state where all goals in the goal state are satisfied
- A solution plan is the sequence of actions (e.g. operator instances) in the path from the start node to a goal node.

In Plan-Space Search

- The search space is the space of all possible plans
- A node corresponds to a partial plan
- Initially we will specify an "initial plan" which is one node in this space.
- A goal node is a node containing a plan which is complete, satisfying all of the goals in the goal state
- The node itself contains all of the information for determining a solution plan (e.g. sequence of actions).

Goal Interaction

Most planning algorithms assume that the goals to be achieved are independent or nearly independent in the sense that each can be solved separately and then the solutions concatenated together. If the order of solving a set of goals (either the original goals or a set of sub-goals which are the preconditions of an operator) fails because solving a latter goal undoes an earlier goal, then

this version of the STRIPS algorithm fails. Hence, situation-space planners do not allow for interleaving of steps in any solution it finds.

Principle of Least Commitment: The principle of least commitment is the idea of never making a choice unless required to do so. The advantage of using this principle is you won't have to backtrack later! In planning, one application of this principle is to never order plan steps unless it's necessary for some reason. So, partial-order planners exhibit this property because constraint ordering steps will only be inserted when necessary. On the other hand, situation-space progression planners make commitments about the order of steps as they try to find a solution and therefore may make mistakes from poor guesses about the right order of steps.

3.3 LEARNING

3.3.1 What Is Machine Learning?

Q18. What is machine learning?

Ans :

Learning

"Learning denotes changes in a system that enables system to do the same task more efficiently next time."

Machine Learning:

Definition

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

Components of Learning System

Performance Element

The performance element is the agent that acts in the world .It percepts and decides on external actions.

Learning Element

It responsible for making improvements, takes knowledge about performance element and some feedback, determines how to modify performance element.

Critic

It tells the learning element how agent is doing by comparing with the fixed standard of performance.

Problem Generator

This component suggests problems or actions that will generate new examples or experience that helps the system to train further.

Let us see the role of each component with an example.

Example

Automated Taxi on city roads

Performance Element: consists of knowledge and procedures for driving actions.

eg: turning, accelerating, breaking are the performance elements on roads.

Learning Element: It formulates goals.

Eg: Learn rules for breaking, accelerating, learn geography of the city.

Critic: Observes world and passes information to learning element.

Eg: quick right turn across three lanes of traffic, observe reaction of other drivers.

Problem Generator: Try south city road

3.3.2 Learning Paradigms

Q19. Explain about learning paradigms.

Ans :

Learning Paradigm

- Rote learning
- Induction
- Clustering
- Analogy
- Discovery
- Genetic algorithms
- Reinforcement

Rote Learning

Rote learning technique avoids understanding the inner complexities but focuses on memorizing the material so that it can be recalled by the learner exactly the way it read or heard.

Learning by memorization: which avoids understanding the inner complexities the subject that is being learned.

Learning something from Repeating: saying the same thing and trying to remember how to say it; it does not help to understand, it helps to remember, like we learn a poem, song, etc.

$$\phi = \{x(n), y(n)\}_{n=1-N} \Rightarrow (A \Rightarrow C)$$

There are two types of inductive learning,

- Supervised
- Unsupervised

Supervised learning: (The machine has access to a teacher who corrects it.)

Learning is the machine learning task of inferring a function from labeled training data. The training data consist of a set of training examples. In supervised learning, each example is a pair consisting of an input object (typically a vector) and a desired output value (also called the supervisory signal). **Example:** Face recognition.

Unsupervised Learning: (No access to teacher. Instead, the machine must search for "order" and "structure" in the environment.) since there is no desired output in this case that is provided therefore categorization is done so that the algorithm differentiates correctly between the face of a horse, cat or human (clustering of data)

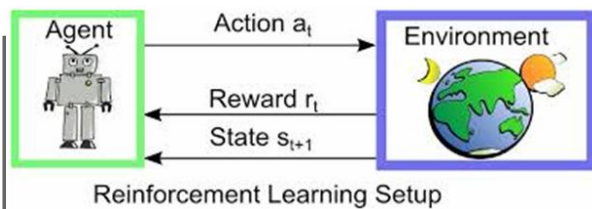
Clustering

In clustering or unsupervised learning, the target features are not given in the training examples. The aim is to construct a natural classification that can be used to cluster the data. The general idea behind clustering is to partition the examples into clusters or classes. Each class predicts feature values for the examples in the class. Each clustering has a prediction error on the predictions. The best clustering is the one that minimizes the error.

Example: An intelligent tutoring system may want to cluster students' learning behavior so that strategies that work for one member of a class may work for other members.

Reinforcement Learning

Imagine a robot that can act in a world, receiving rewards and punishments and determining from these what it should do. This is the problem of **reinforcement learning**. Most Reinforcement Learning research is conducted with in the mathematical framework of **Markov Decision Process**.



3.3.3 Learning Concepts

Q20. Write about Learning Concepts.

Ans :

The problem of **learning** is to take in prior knowledge and data (e.g., about the experiences of the agent) and to create an internal representation (the knowledge base) that is used by the agent as it acts. This internal representation could be the raw experiences themselves, but it is typically a compact representation that summarizes the data. The problem of inferring an internal representation based on examples is often called **induction** and can be contrasted with deduction, which is deriving consequences of a knowledge base, and abduction, which is hypothesizing what may be true about a particular case.

There are two principles that are at odds in choosing a representation scheme:

- The richer the representation scheme, the more useful it is for subsequent problems solving. For an agent to learn a way to solve a problem, the representation must be rich enough to express a way to solve the problem.

- The richer the representation, the more difficult it is to learn. A very rich representation is difficult to learn because it requires a great deal of data, and often many different hypotheses are consistent with the data.

Learning techniques face the following issues

Task

Virtually any task for which an agent can get data or experiences can be learned. The most commonly studied learning task is **supervised learning**: given some input features, some target features, and a set of **training examples** where the input features and the target features are specified, predict the target features of a new example for which the input features are given. This is called **classification** when the target variables are discrete and **regression** when the target features are continuous.

Feedback

Learning tasks can be characterized by the feedback given to the learner. In **supervised learning**, what has to be learned is specified for each example. Supervised classification occurs when a trainer provides the classification for each example. Supervised learning of actions occurs when the agent is given immediate feedback about the value of each action. **Unsupervised learning** occurs when no classifications are given and the learner must discover categories and regularities in the data. Feedback often falls between these extremes, such as in **reinforcement learning**, where the feedback in terms of rewards and punishments occurs after a sequence of actions. This leads to the **credit-assignment problem** of determining which actions were responsible for the rewards or punishments. For example, a user could give rewards to the delivery robot without telling it exactly what it is being rewarded for. The robot then must either learn what it is being rewarded for or learn which actions are preferred in which situations. It is possible that it can learn what actions to perform without actually determining which consequences of the actions are responsible for rewards.

Representation

For an agent to use its experiences, the experiences must affect the agent's internal representation. Much of machine learning is studied

in the context of particular representations (e.g., decision trees, neural networks, or case bases). This chapter presents some standard representations to show the common features behind learning.

Online and offline

In **offline learning**, all of the training examples are available to an agent before it needs to act. In **online learning**, training examples arrive as the agent is acting. An agent that learns online requires some representation of its previously seen examples before it has seen all of its examples. As new examples are observed, the agent must update its representation. Typically, an agent never sees all of its examples. **Active learning** is a form of online learning in which the agent acts to acquire useful examples from which to learn. In active learning, the agent reasons about which examples would be useful to learn from and acts to collect these examples.

Measuring success

Learning is defined in terms of improving performance based on some measure. To know whether an agent has learned, we must define a measure of success. The measure is usually not how well the agent performs on the training experiences, but how well the agent performs for new experiences.

In classification, being able to correctly classify all training examples is not the problem. For example, consider the problem of predicting a Boolean feature based on a set of examples. Suppose that there were two agents *P* and *N*. Agent *P* claims that all of the negative examples seen were the only negative examples and that every other instance is positive. Agent *N* claims that the positive examples in the training set were the only positive examples and that every other instance is negative. Both of these agents correctly classify every example in the training set but disagree on every other example. Success in learning should not be judged on correctly classifying the training set but on being able to correctly classify unseen examples. Thus, the learner must generalize: go beyond the specific given examples to classify unseen examples.

A standard way to measure success is to divide the examples into a training set and a test set. A representation is built using the training set, and then

the predictive accuracy is measured on the test set. Of course, this is only an approximation of what is wanted; the real measure is its performance on some future task.

Bias

The tendency to prefer one hypothesis over another is called a bias. Consider the agents *N* and *P* defined earlier. Saying that a hypothesis is better than *N*'s or *P*'s hypothesis is not something that is obtained from the data - both *N* and *P* accurately predict all of the data given - but is something external to the data. Without a bias, an agent will not be able to make any predictions on unseen examples. The hypotheses adopted by *P* and *N* disagree on all further examples, and, if a learning agent cannot choose some hypotheses as better, the agent will not be able to resolve this disagreement. To have any inductive process make predictions on unseen data, an agent requires a bias. What constitutes a good bias is an empirical question about which biases work best in practice; we do not imagine that either *P*'s or *N*'s biases work well in practice.

Learning as search

Given a representation and a bias, the problem of learning can be reduced to one of search. Learning is a search through the space of possible representations, trying to find the representation or representations that best fits the data given the bias. Unfortunately, the search spaces are typically prohibitively large for systematic search, except for the simplest of examples. Nearly all of the search techniques used in machine learning can be seen as forms of local search through a space of representations. The definition of the learning algorithm then becomes one of defining the search space, the evaluation function, and the search method.

Noise

In most real-world situations, the data are not perfect. Noise exists in the data (some of the features have been assigned the wrong value), there are inadequate features (the features given do not predict the classification), and often there are examples with missing features. One of the important properties of a learning algorithm is its ability to handle noisy data in all of its forms.

Interpolation and extrapolation

For cases in which there is a natural interpretation of "between," such as where the prediction is about time or space, interpolation involves making a prediction between cases for which there are data. Extrapolation involves making a prediction that goes beyond the seen examples. Extrapolation is usually much more inaccurate than interpolation. For example, in ancient astronomy, the Ptolemaic system and heliocentric system of Copernicus made detailed models of the movement of solar system in terms of epicycles (cycles within cycles). The parameters for the models could be made to fit the data very well and they were very good at interpolation; however, the models were very poor at extrapolation. As another example, it is often easy to predict a stock price on a certain day given data about the prices on the days before and the days after that day. It is very difficult to predict the price that a stock will be tomorrow, and it would be very profitable to be able to do so. An agent must be careful if its test cases mostly involve interpolating between data points, but the learned model is used for extrapolation.

3.3.4 Methods and Models

Q21. Explain about various models in machine learning.

Ans :

For classification and regression problem, there are different choices of Machine Learning Models each of which can be viewed as a blackbox that solve the same problem. However, each model come from a different algorithm approaches and will perform differently under different data set. The best way is to use cross-validation to determine which model perform best on test data.

Decision Tree based methods

The fundamental learning approach is to recursively divide the training data into buckets of homogeneous members through the most discriminative dividing criteria. During the learning, various dividing criteria based on the input will be tried (using in a greedy manner); when the input is a category (Mon, Tue, Wed ...), it will first be turned into binary (isMon, isTue, isWed ...) and then use the true/false as a decision boundary to evaluate the homo-

geneity; when the input is a numeric or ordinal value, the lessThan, greaterThan at each training data input value will be used as the decision boundary. The training process stops when there is no significant gain in homogeneity by further split the Tree. The members of the bucket represented at leaf node will vote for the prediction; majority wins when the output is a category and member average when the output is a numeric.

The good part of Tree is that it is very flexible in terms of the data type of input and output variables which can be categorical, binary and numeric value. The level of decision nodes also indicate the degree of influences of different input variables. The limitation is each decision boundary at each split point is a concrete binary decision. Also the decision criteria only consider one input attributes at a time but not a combination of multiple input variables. Another weakness of Tree is that once learned it cannot be updated incrementally. When new training data arrives, you have to throw away the old tree and retrain every data from scratch.

Linear regression based methods

The basic assumption is that the output variable (a numeric value) can be expressed as a linear combination (weighted sum) of a set of input variable (which is also numeric value).

$$y = w_1x_1 + w_2x_2 + w_3x_3 \dots$$

The whole objective of the training phase is to learn the weights $w_1, w_2 \dots$ by minimizing the error function lost ($y, w_1x_1 + w_2x_2 + \dots$). Gradient descent is the classical technique of solving this problem with the general idea of adjusting $w_1, w_2 \dots$ along the direction of the maximum gradient of the loss function.

The input variable is required to be numeric. For binary variable, this will be represented as 0, 1. For categorical variable, each possible value will be represented as a separate binary variable (and hence 0, 1). For the output, if it is a binary variable (0, 1) then a logit function is used to transform the range of -infinity to +infinity into 0 to 1. This is called logistic regression and a different loss function (based on maximum likelihood) is used.

To avoid overfitting, regularization technique (L1 and L2) is used to penalize large value of $w_1, w_2 \dots$ L1 is by adding the absolute value of w_1 into

the loss function while L2 is by adding the square of w_1 into the loss function. L1 has the property that it will penalize redundant features or irrelevant feature more (with very small weight) and is a good tool to select highly influential features.

The strength of Linear model is that it has very high performance in both scoring and learning. The Stochastic gradient descent-based learning algorithm is highly scalable and can handle incremental learning.

The weakness of linear model is linear assumption of input features, which is often false. Therefore, an important feature engineering effort is required to transform each input feature, which usually involved domain expert. Another common way is to throw different transformation functions $1/x$, x^2 , $\log(x)$ in the hope that one of them will have a linear relationship with the output. Linearity can be checked by observing whether the residual ($y - \text{predicted}_y$) is normally distributed or not (using the Qplot with the Gaussian distribution).

Neural Network

Neural Network can be considered as multiple layer of perceptron (each is a logistic regression unit with multiple binary input and one binary output). By having multiple layers, this is equivalent to : $z = \text{logit}(v_1 \cdot y_1 + v_2 y_2 + \dots)$, while $y_1 = \text{logit}(w_1 x_1 + w_2 x_2 + \dots)$

This multi-layer model enables Neural Network to learn non-linear relationship between input x and output z . The typical learning technique is “backward error propagation” where the error is propagate from the output layer back to the input layer to adjust the weight.

Bayesian Network

It is basically a dependency graph where each node represents a binary variable and each edge (directional) represents the dependency relationship. If Node A and Node B has an edge to Node C. This means the probably of C is true depends on different combinations of the boolean value of A and B. Node C can point to Node D and now Node D depends on Node A and Node B as well.

The learning is about finding at each node the join-probability distribution of all incoming edges. This is done by counting the observed values of A,

B and C and then update the joint probability distribution table at Node C.

Once we have the probability distribution table at every node, then we can compute the probability of any hidden node (output variable) from the observed nodes (input variables) by using the Bayes rule.

The strength of Bayesian network is it is highly scalable and can learn incrementally because all we do is to count the observed variables and update the probability distribution table. Similar to Neural Network,

Support Vector Machine

Support Vector Machine takes numeric input and binary output. It is based on finding a linear plane with maximum margin to separate two class of output. Categorical input can be turned into numeric input as before and categorical output can be modeled as multiple binary output.

With a different lost function, SVM can also do regression (called SVR). I haven't used this myself so I can't talk much.

The strength of SVM is it can handle large number of dimensions. With the kernel function, it can handle non-linear relationship as well.

Nearest Neighbour

We are not learning a model at all. The idea is to find K similar data point from the training set and use them to interpolate the output value, which is either the majority value for categorical output, or average (or weighted average) for numeric output. K is a tunable parameter which needs to be cross-validated to pick the best value.

Nearest Neighbour require the definition of a distance function which is used to find the nearest neighbour. For numeric input, the common practice is to normalize them by minus the mean and divided by the standard deviation. Euclidean distance is commonly used when the input are independent, otherwise mahalanobis distance (which account for correlation between pairs of input features) should be used instead. For binary attributes, Jaccard distance can be used.

The strength of K nearest neighbor is its simplicity as no model needs to be trained.

Incremental learning is automatic when more data arrives (and old data can be deleted as well). Data, however, needs to be organized in a distance-aware tree such that finding the nearest neighbor is $O(\log N)$ rather than $O(N)$. On the other hand, the weakness of KNN is it doesn't handle high number of dimensions well. Also, the weighting of different factors needs to be hand tuned (by cross-validation on different weighting combination) and can be a very tedious process.

3.3.5 Statistical Learning Methods

Q22. Explain about statistical learning methods.

Ans :

Full Bayesian learning

View learning as Bayesian updating of a probability distribution over the hypothesis space

- H is the hypothesis variable, values h_1, h_2, \dots , prior $P(H)$.
- J th observation d_j gives the outcome of random variable D | training data $d = d_1, \dots, d_N$
- Given the data so far, each hypothesis has a posterior probability:

$P(h_i | d) = \alpha P(d | h_i) P(h_i)$ where $P(d | h_i)$ is called the likelihood

Predictions use a likelihood-weighted average over the hypotheses:

$$P(X | d) = \sum_i P(X | d, h_i) P(h_i | d) = \sum_i P(X | h_i) P(h_i | d).$$

No need to pick one best-guess hypothesis!

Example

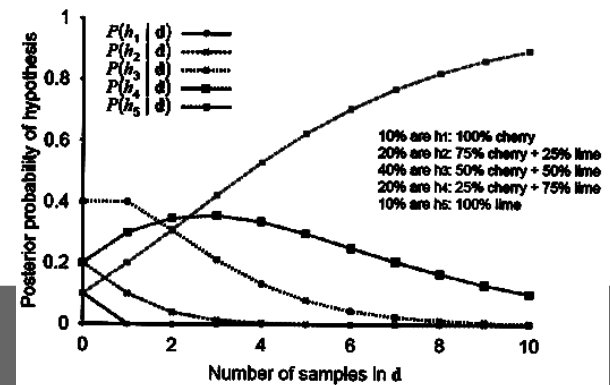
Suppose there are five kinds of bags of candies:

- 10% are h_1 : 100% cherry candies
- 20% are h_2 : 75% cherry candies + 25% lime candies
- 40% are h_3 : 50% cherry candies + 50% lime candies
- 20% are h_4 : 25% cherry candies + 75% lime candies
- 10% are h_5 : 100% lime candies

Then we observe candies drawn from some bag:

What kind of bag is it? What flavor will the next candy be?

Posterior probability of hypotheses



MAP approximation

Maximum a posteriori (MAP) learning: choose h_{MAP} maximizing $P(h_i | d)$ instead of calculating $P(h_i | d)$ for all hypothesis h_i , i.e., maximize $P(d | h_i) P(h_i)$ or $\log P(d | h_i) + \log P(h_i)$

Overfitting in MAP and Bayesian learning

Overfitting when the hypothesis space is too expressive such that some hypotheses fit the data set well.

Use prior to penalize complexity

Log terms can be viewed as (negative of) bits to encode data given hypothesis + bits to encode hypothesis

This is the basic idea of minimum description length (MDL) learning For deterministic hypotheses (simplest), $P(d | h_i)$ is 1 if consistent, 0 otherwise \Rightarrow MAP = simplest hypothesis that is consistent with the data.

ML approximation

For large data sets, prior becomes irrelevant

Maximum likelihood (ML) learning: choose h_{ML} maximizing $P(d | h_i)$ i.e., simply get the best fit to the data; identical to MAP for uniform prior (which is reasonable if all hypotheses are of the same complexity)

ML is the "standard" (non-Bayesian) statistical learning method

1. Researchers distrust the subjective nature of hypotheses priors
2. Hypotheses are of the same complexity
3. Hypotheses priors is of less important when data set is large
4. Huge space of the hypotheses

ML parameter learning in Bayesnets

Bag from a new manufacturer; fraction q of cherry candies?

- Any q is possible: continuum of hypotheses h_q
- q is a parameter for this simple (binomial) family of models
- Suppose we unwrap N candies, c cherries and $\ell = N - c$ limes
- These are i.i.d. (independent, identically distributed) observations, so

$$P(d | h_\theta) = \prod_{j=1}^N P(d_j | h_\theta) = \theta^c \cdot (1 - \theta)^\ell$$

Maximize this w.r.t. θ – which is easier for the log-likelihood;

$$L(d | h_\theta) = \log P(d | h_\theta) = \sum_{j=1}^N \log P(d_j | h_\theta) = c \log \theta + \ell \log(1 - \theta)$$

$$\frac{dL(d | h_\theta)}{d\theta} = \frac{c}{\theta} - \frac{\ell}{1 - \theta} = 0 \Rightarrow \theta = \frac{c}{c + \ell} = \frac{c}{N}$$

Seems sensible, but causes problems with 0 counts!

Multiple parameters

Red/green wrapper depends probabilistically on flavor:

Likelihood for, e.g., cherry candy in green wrapper:

$$\begin{aligned} P(F = \text{cherry}, W = \text{green} | h_{\theta, \theta_1, \theta_2}) \\ &= P(F = \text{cherry} | h_{\theta, \theta_1, \theta_2}) P(W = \text{green} | F = \text{cherry}, h_{\theta, \theta_1, \theta_2}) \\ &= \theta \cdot (1 - \theta_1). \end{aligned}$$

N candies, r_c red-wrapped cherry candies, etc.:

$$P(d | h_{\theta, \theta_1, \theta_2}) = \theta^c (1 - \theta)^\ell \cdot \theta_1^{r_c} (1 - \theta_1)^{g_c} \cdot \theta_2^{r_\ell} (1 - \theta_2)^{g_\ell}$$

$$L = [c \log \theta + \ell \log(1 - \theta)] + [r_c \log \theta_1 + g_c \log(1 - \theta_1)] + [r_\ell \log \theta_2 + g_\ell \log(1 - \theta_2)]$$

Derivatives of L contain only the relevant parameter:

$$\begin{aligned} \frac{\partial L}{\partial \theta} &= \frac{c}{\theta} - \frac{\ell}{1 - \theta} = 0 & \Rightarrow & \theta = \frac{c}{c + \ell} \\ \frac{\partial L}{\partial \theta_1} &= \frac{r_c}{\theta_1} - \frac{g_c}{1 - \theta_1} = 0 & \Rightarrow & \theta_1 = \frac{r_c}{r_c + g_c} \\ \frac{\partial L}{\partial \theta_2} &= \frac{r_\ell}{\theta_2} - \frac{g_\ell}{1 - \theta_2} = 0 & \Rightarrow & \theta_2 = \frac{r_\ell}{r_\ell + g_\ell} \end{aligned}$$

With complete data, parameters can be learned separately.

3.3.6 Artificial Neural Networks–Based Learning

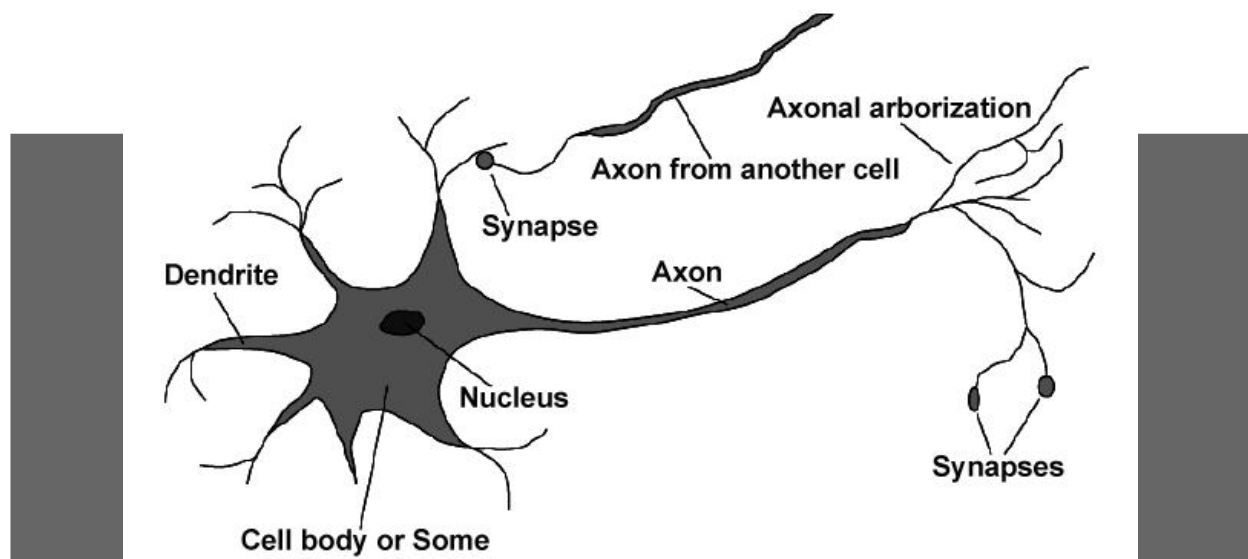
Q23. Write about Artificial Neural network based learning.

Ans :

Neural Networks

Brains

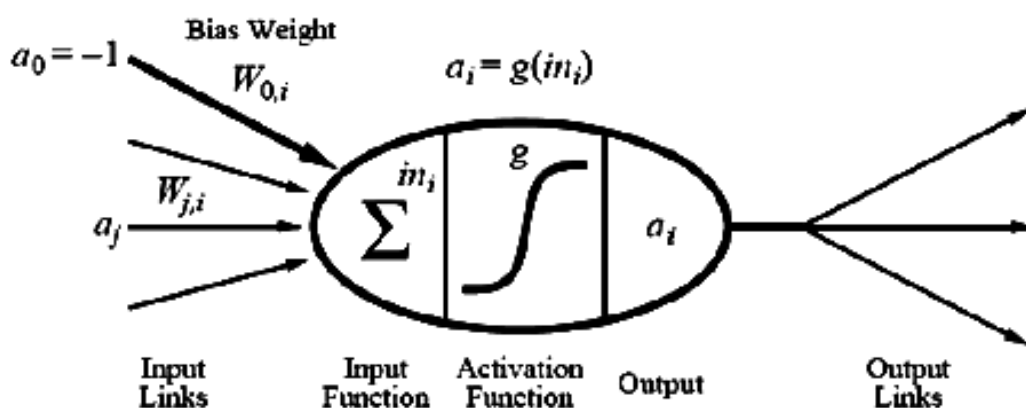
1011 neurons of > 20 types, 1014 synapses, 1 ms – 10 ms cycle time - Signals are noisy "spike trains of electrical potential.



McCulloch-Pitts "unit"

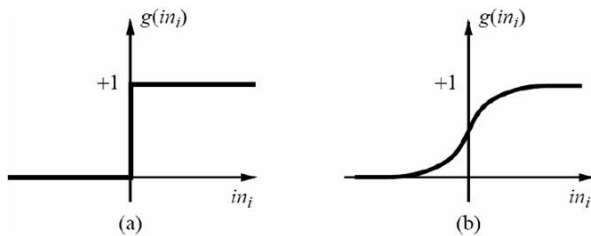
Output is a – squashed || linear function of the inputs:

$$a_i \leftarrow g(in_i) = g(\sum_j W_{j,i} a_j)$$



A gross oversimplification of real neurons, but its purpose is to develop understanding of what networks of simple units can do.

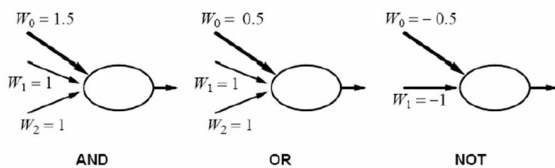
Activation Functions



- (a) is a step function or threshold function
- (b) is a sigmoid function $1/(1 + e^{-x})$

Changing the bias weight $W_{0,i}$ moves the threshold location.

Implementing Logical Functions



McCulloch and Pitts: every Boolean function can be implemented

Network structures

Feed-forward networks

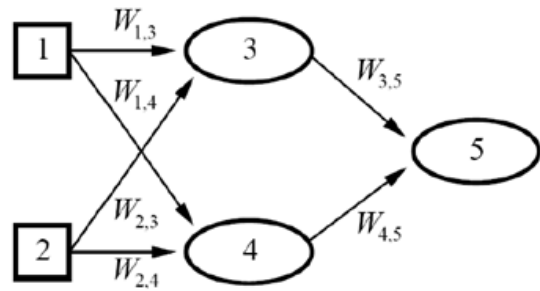
- single-layer perceptrons
- multi-layer perceptrons

Feed-forward networks implement functions, have no internal state.

Recurrent networks

- Hopfield networks have symmetric weights ($W_{i,j} = W_{j,i}$) $g(x) = \text{sign}(x)$, $a_i = \pm 1$; holographic associative memory
- Boltzmann machines use stochastic activation functions, \approx MCMC in Bayesnets
- Recurrent neural nets have directed cycles with delay have internal state (like flip-flops), can oscillate etc.

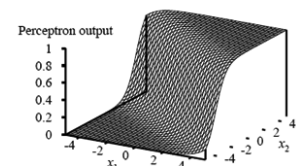
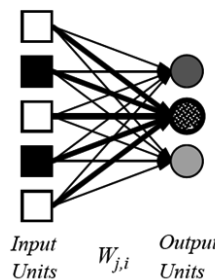
Feed-forward Example



Feed-forward network = a parameterized family of nonlinear functions:

$$a_5 = g(W_{3,5} \cdot a_3 + W_{4,5} \cdot a_4) \\ = g(W_{3,5} \cdot g(W_{1,3} \cdot a_1 + W_{2,3} \cdot a_2) + W_{4,5} \cdot g(W_{1,4} \cdot a_1 + W_{2,4} \cdot a_2))$$

Single-layer Perceptrons



Output units all operate separately - no shared weights

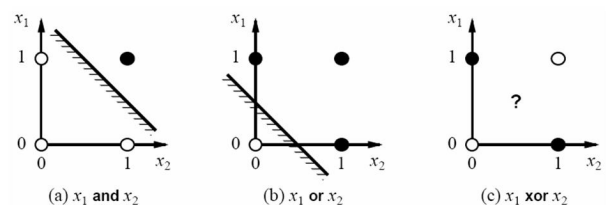
Adjusting weights moves the location, orientation, and steepness of cli

Expressiveness of perceptrons

Consider a perceptron with $g = \text{step function}$.

Can represent AND, OR, NOT, majority, etc., but not XOR

Represents a linear separator in input space: $\sum_j W_j x_j > 0$ or $W \cdot x > 0$.



Perceptron Learning

Learn by adjusting weights to reduce error on training set. The squared error for an example with input x and true output y is.

$$E = \frac{1}{2} \text{Err}^2 \equiv \frac{1}{2} (y - hW(x))^2$$

Perform optimization search by gradient descent:

$$\begin{aligned} \frac{\partial E}{\partial W_j} &= \text{Err} \times \frac{\partial \text{Err}}{\partial W_j} \\ &= \text{Err} \times \frac{\partial}{\partial W_j} (y - g(\sum_j W_j x_j)) \\ &= -\text{Err} \times g'(in) \times x_j \end{aligned}$$

Simple weight update rule:

$$W_j \leftarrow W_j + a \times \text{Err} \times g'(in) \times x_j$$

E.g., +ve error \Rightarrow increase network output

\Rightarrow increase weights on +ve inputs, decrease on -ve inputs

Perceptron learning rule converges to consistent function for any linearly separable data set

Perceptron learns majority function easily, DTL is hopeless.

DTL learns restaurant function easily, perceptron cannot represent it.

Multi-layer Perceptrons

Layers are usually fully connected;

Numbers of hidden units typically chosen by hand.

Output units

a_i

$W_{j,i}$

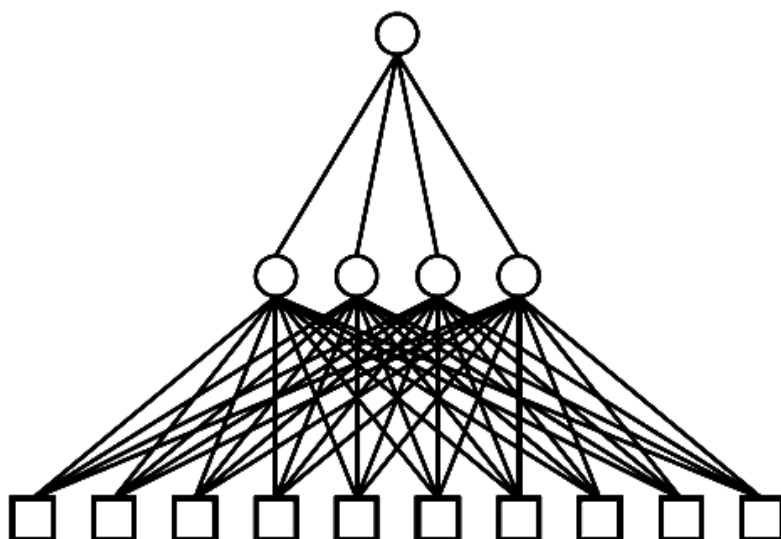
Hidden units

a_j

$W_{k,j}$

Input units

a_k



Back-propagation learning

Output layer: same as for single-layer perceptron,

$$W_{j,i} \leftarrow W_{j,i} + \alpha \times \alpha_j \times \Delta_i$$

Where $\Delta_i = \text{Err}_i \times g'(in_i)$

Hidden layer: back-propagate the error from the output layer:

Where $\Delta_i = \text{Err}_i \times g'(in_i)$

$$W_{k,j} \leftarrow W_{k,j} + \alpha \times \alpha_k \times \Delta_i$$

(Most neuroscientists deny that back-propagation occurs in the brain)

Back-propagation derivation

The squared error on a single example is defined as

$$E = \frac{1}{2} \sum_i (y_i - a_i)^2$$

where the sum is over the nodes in the output layer.

$$\frac{\partial E}{\partial W_{j,i}} = -(y_i - a_i) \frac{\partial a_i}{\partial W_{j,i}} = -(y_i - a_i) \frac{\partial g(in_i)}{\partial W_{j,i}}$$

$$= -(y_i - a_i) g'(in_i) \frac{\partial in_i}{\partial W_{j,i}}$$

$$= -(y_i - a_i) g'(in_i) \frac{\partial}{\partial W_{j,i}} (\sum_j W_{j,i} a_j)$$

$$= -(y_i - a_i) g'(in_i) a_j = -a_j \Delta_i$$

$$\begin{aligned} \frac{\partial E}{\partial W_{k,j}} &= -\sum_i (y_i - a_i) \frac{\partial a_i}{\partial W_{k,j}} = -\sum_i (y_i - a_i) \frac{\partial g(in_i)}{\partial W_{k,j}} \\ &= -\sum_i (y_i - a_i) g'(in_i) \frac{\partial in_i}{\partial W_{k,j}} = -\sum_i \Delta_i \frac{\partial}{\partial W_{k,j}} (\sum_j W_{j,i} a_j) \\ &= -\sum_i \Delta_i W_{j,i} \frac{\partial a_j}{\partial W_{k,j}} = -\sum_i \Delta_i W_{j,i} \frac{\partial g(in_j)}{\partial W_{k,j}} \\ &= -\sum_i \Delta_i W_{j,i} g'(in_j) \frac{\partial in_j}{\partial W_{k,j}} \\ &= -\sum_i \Delta_i W_{j,i} g'(in_j) \frac{\partial}{\partial W_{k,j}} (\sum_k W_{k,j} a_k) \\ &= -\sum_i \Delta_i W_{j,i} g'(in_j) a_k = -a_k \Delta_j \end{aligned}$$

At each epoch, sum gradient updates for all examples and apply

Training curve for 100 restaurant examples: finds exact fit

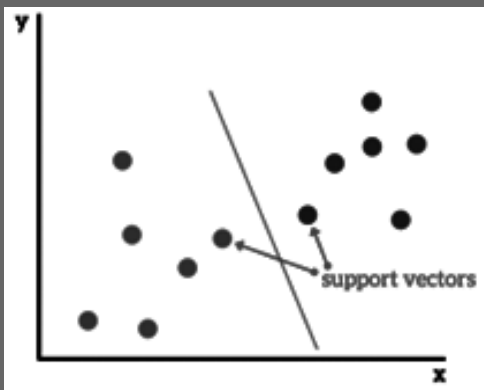
3.3.7 Support Vector Machines

Q24. What are Support Vector Machines ? Write about them.

Ans :

A Support Vector Machine (SVM) is a supervised machine learning algorithm that can be employed for both classification and regression purposes. SVMs are more commonly used in classification problems and as such, this is what we will focus on in this post.

SVMs are based on the idea of finding a hyperplane that best divides a dataset into two classes, as shown in the image below.



Support Vectors

Support vectors are the data points nearest to the hyperplane, the points of a data set that, if removed, would alter the position of the dividing hyperplane. Because of this, they can be considered the critical elements of a data set.

What is a hyperplane?

As a simple example, for a classification task with only two features (like the image above), you can think of a hyperplane as a line that linearly separates and classifies a set of data.

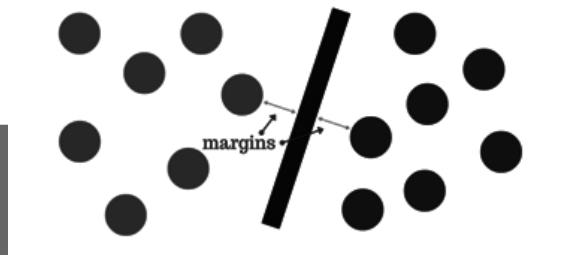
Intuitively, the further from the hyperplane our data points lie, the more confident we are that they have been correctly classified. We therefore want our data points to be as far away from the hyperplane as possible, while still being on the correct side of it.

So when new testing data is added, whatever side of the hyperplane it lands will decide the class that we assign to it.

How do we find the right hyperplane?

Or, in other words, how do we best segregate the two classes within the data?

The distance between the hyperplane and the nearest data point from either set is known as the margin. The goal is to choose a hyperplane with the greatest possible margin between the hyperplane and any point within the training set, giving a greater chance of new data being classified correctly.

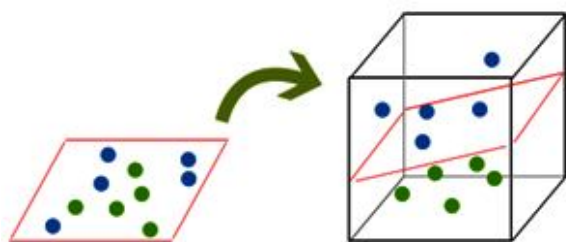


But what happens when there is no clear hyperplane?

This is where it can get tricky. Data is rarely ever as clean as our simple example above. A dataset will often look more like the jumbled balls below which represent a linearly non separable dataset.



In order to classify a dataset like the one above it's necessary to move away from a 2d view of the data to a 3d view. Explaining this is easiest with another simplified example. Imagine that our two sets of colored balls above are sitting on a sheet and this sheet is lifted suddenly, launching the balls into the air. While the balls are up in the air, you use the sheet to separate them. This 'lifting' of the balls represents the mapping of data into a higher dimension. This is known as kernelling. You can read more on Kernelling [here](#).



Because we are now in three dimensions, our hyperplane can no longer be a line. It must now be a plane as shown in the example above. The idea is that the data will continue to be mapped into higher and higher dimensions until a hyperplane can be formed to segregate it.

Pros & Cons of Support Vector Machines

Pros

- Accuracy
- Works well on smaller cleaner datasets
- It can be more efficient because it uses a subset of training points

Cons

- Isn't suited to larger datasets as the training time with SVMs can be high
- Less effective on noisier datasets with overlapping classes

SVM Uses

SVM is used for text classification tasks such as category assignment, detecting spam and sentiment analysis. It is also commonly used for image recognition challenges, performing particularly well in aspect-based recognition and color-based classification. SVM also plays a vital role in many areas of handwritten digit recognition, such as postal automation services.

3.3.8 Reinforcement Learning

Q25. What does Reinforcement Learning mean?

Ans :

Reinforcement learning, in the context of artificial intelligence, is a type of dynamic programming that trains algorithms using a system of reward and punishment.

A reinforcement learning algorithm, or agent, learns by interacting with its environment. The agent receives rewards by performing correctly and penalties for performing incorrectly. The agent learns without intervention from a human by maximizing its reward and minimizing its penalty.

Imagine a robot that can act in a world, receiving rewards and punishments and determining from these what it should do. This is the problem of **reinforcement learning**.

We can formalize reinforcement learning in terms of Markov decision processes, but in which the agent, initially, only knows the set of possible states and the set of possible actions. Thus, the dynamics, $P(s'/a,s)$, and the reward function, $R(s,a,s')$, are initially unknown. An agent can act in a world and, after each step, it can observe the state of the world and observe what reward it obtained. Assume the agent acts to achieve the optimal discounted reward with a discount factor γ .

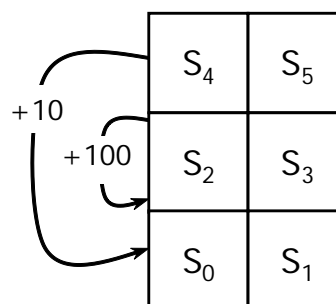


Fig.: The environment of a tiny reinforcement learning problem

Example

Consider the tiny reinforcement learning problem shown in Figure. There are six states the agent could be in, labeled as s_0, \dots, s_5 . The agent has four actions: UpC, Up, Left, Right. That is all the agent knows before it starts. It does not know how the states are configured, what the actions do, or how rewards are earned.

Figure shows the configuration of the six states. Suppose the actions work as follows:

upC

(for "up carefully") The agent goes up, except in states s_4 and s_5 , where the agent stays still, and has a reward of -1 .

right

The agent moves to the right in states s_0, s_2, s_4 with a reward of 0 and stays still in the other states, with a reward of -1.

left

The agent moves one state to the left in states s_1, s_3, s_5 . In state s_0 , it stays in state s_0 and has a reward of -1. In state s_2 , it has a reward of -100 and stays in state s_2 . In state s_4 , it gets a reward of 10 and moves to state s_0 .

up

With a probability of 0.8 it acts like upC, except the reward is 0. With probability 0.1 it acts as a left, and with probability 0.1 it acts as right.

Suppose there is a discounted reward with a discount of 0.9. This can be translated as having a 0.1 chance of the agent leaving the game at any step, or as a way to encode that the agent prefers immediate rewards over future rewards.

P_0	R			P_1
		M		
				M
M	M		M	
P_2				P_3

UNIT IV

Expert Systems: Architecture of expert system, confidence factors, existing expert systems, knowledge acquisition, shell and explanations, self-explaining system, rule-based expert systems, forward and backward chaining, frame-based expert systems, uncertainty management in expert systems, expert system and dss, pros and cons of expert systems, case study.

Pattern Recognition: Machine perception and pattern recognition, feature extraction, classification, object recognition, speech recognition, pattern mining. Game playing: Important concepts of game theory, game playing and knowledge structure, game as search problem, alpha-beta pruning, game theory problems, robotics. Concepts and terminology of ann, feed-forward nn, feedback networks, pattern associative net works, competitive learning, fuzzy sets, fuzzy inference process, neuro-fuzzy systems, range of ai applications, ai applications and examples, case study: Agricultural domain - farmer's intelligent assistant.

4.1 EXPERT SYSTEMS

4.1.1 Architecture of Expert System

Q1. What is Expert System?

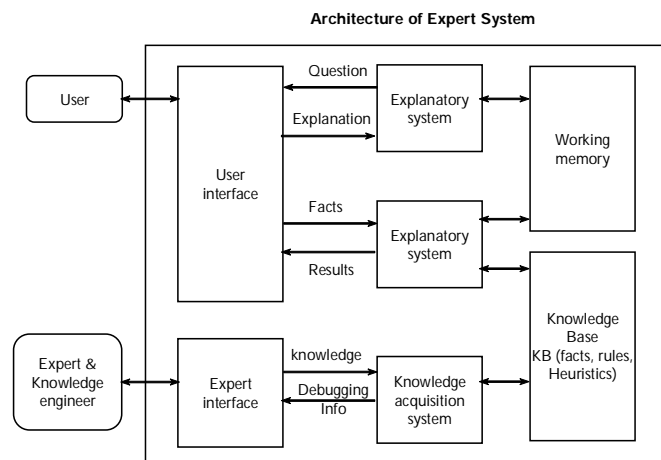
Ans :

1. An expert system, is an interactive computer-based decision tool that uses both facts and heuristics to solve difficult decision making problems, based on knowledge acquired from an expert.
2. An expert system is a model and associated procedure that exhibits, within a specific domain, a degree of expertise in problem solving that is comparable to that of a human expert.
3. An expert system compared with traditional computer : Inference engine + Knowledge = Expert system (Algorithm + Data structures = Program in traditional computer)
4. First expert system, called DENDRAL, was developed in the early 70's at Standard University

Expert systems are computer applications which embody some non-algorithmic expertise for solving certain types of problems. For example, expert systems are used in diagnostic applications. They also play chess, make financial planning decisions, configure computers, monitor real time systems, underwrite insurance policies, and perform many services which previously required human expertise.

Expert System Components And Human Interfaces

Expert systems have a number of major system components and interface with individuals who interact with the system in various roles. These are illustrated below. Draw and describe the architecture of expert system.



The Architecture of an Expert System (ES) consists of the following major components:

- **Knowledge Base (KB):** repository of special heuristics or rules that direct the use of knowledge, facts (productions). It contains the knowledge necessary for understanding, formulating, & problem solving.
- **Working Memory (Blackboard):** if forward chaining used It describes the current problem & record intermediate results Records Intermediate Hypothesis & Decisions: 1. Plan, 2. Agenda, 3. Solution

- **Inference Engine:** the deduction system used to infer results from user input & KB It is the brain of the ES, the control structure (rule interpreter)

It provides methodology for reasoning

- **Explanation Subsystem (Justifier):** Traces responsibility & explains the ES behaviour by interactively answering question: Why?, How?, What?, Where?, When?, Who?
- **User Interface:** interfaces with user through Natural Language Processing (NLP), or menus & graphics. Acts as Language Processor for friendly, problem-oriented communication

Shell = Inference Engine + User Interface

The Human Elements in ESs

- **Expert:** Has the special knowledge, judgement, experience and methods to give advice and solve problems.

Provides knowledge about task performance

- **Knowledge Engineer:** Usually also the System Builder.

Helps the expert(s) structure the problem area by interpreting and integrating human answers to questions, drawing analogies, posing counter examples, and bringing to light conceptual difficulties.

The Expert & the knowledge Engineer should Anticipate Users' needs & Limitations when designing Expert Systems.

User: Possible Classes of Users can be

- A non-expert client seeking direct advice (ES acts as a Consultant or Advisor).
- A student who wants to learn (ES acts as an Instructor)
- An ES builder improving or increasing the knowledge base(ES acts as a Partner)
- An Expert (ES acts as a Colleague or an Assistant)

4.1.2 Confidence Factors

Q2. What are confidence factors in expert systems.

Ans :

Some Expert systems incorporate certain factors(CF) 'Tomorrow it won't rain' might have the confidence factor of 99.9% for A Tacoma desert. CF s can be difficult to define objectively, are not catered for by all ES languages.

Rules with Confidence Factors

This approach t'o uncertainty combines probability with logic. It enhances rule-based systems with probability-like numbers that represent the confidence in either a fact or an inferred conclusion. For example, consider this rule:

If the engine will not start but it will turn over, then the injection system is bad.

In some cases the facts are uncertain. Suppose the user is uncertain whether the engine starts or whether it turns over. If the user is 70 percent sure that the engine does not start and 80 percent sure that the engine turns over, then the conclusion of a bad injection system will be uncertain as well. A typical inference with this uncertainty is to multiply the two probabilities. In this case, 70 percent times 80 percent results in 56 percent confidence that the injection system is bad.

Furthermore, the rule itself may be uncertain. An expert may be only 60 percent sure that an unstartable engine that turns over implies a bad injection system. In this case, even if the user were 100 percent sure that the engine does not start but does turn over, the confidence in the conclusion of a bad injection system would be only 60 percent.

The inference process propagates the uncertainties through to the conclusions, so that the expert system tells the user not only what its recommendation is, but also the level of confidence in the recommendation.

An example of an expert system using rules can be found in the Department of Veterans Affairs within their One VA initiative, which seeks to improve service by implementing improved information technology. A component of this initiative is the creation of an "expert system for the determination of potential benefits." This expert system utilizes a rule-based approach that analyzes customer data to determine proper eligibility levels.

4.1.3 Existing Expert Systems Expert Systems - Dendral, Mycin

Q3. Write shortly about DENDRAL – the expert system software.

Ans :

DENDRAL

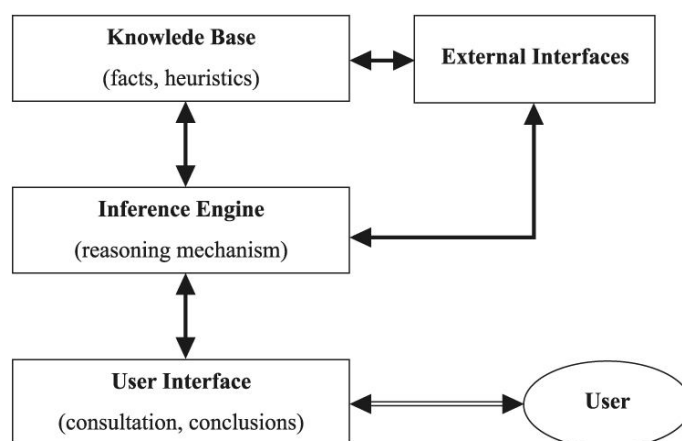
DENDRAL is a program that analyses organic compounds to determine their structure. It is one of the early example of a successful AI program .it uses a strategy called plan-generate-test in which a planning process that used constraint-satisfaction techniques, creates lists of recommended and contraindicated substructures.

One type in DENDRAL is meta - dendral.

Meta-Dendral

Meta-Dendral is a machine learning system that receives the set of possible chemical structures and corresponding mass spectra as input, and proposes a set of rules of mass spectrometry that correlate structural features with processes that produce the mass spectrum. These rules would be fed back to Heuristic Dendral (in the planning and testing programs described below) to test their applicability. Thus, "Heuristic Dendral is a performance system and Meta-Dendral is a learning system". The program is based on two important features: the plan-generate-test paradigm and knowledge engineering.

The following is the architecture of DENDRAL system



Q4. Explain about MYCIN.

Ans :

Mycin

Mycin is a program that diagnoses infectious diseases. It reasons backward from its goal of determining the cause of a patient illness. It attempts to solve its goal of recommending a therapy for a particular

patient by first finding the cause of the patient's illness. It uses its production rules to reason backward from goals to clinical observations. To solve the top-level diagnostic goal, it looks for rules whose right sides suggest diseases. It then uses the left sides of those rules (the preconditions) to set up sub goals whose success would enable the rules to be invoked. These sub goals are again matched against rules, and their preconditions are used to set up additional sub goals.

Mycin is a well known rule based deduction system. Its expertise lies in the domain of bacterial infections. Physicians usually must begin antibiotic treatment for patient who have bacterial infections without knowledge exactly which organism is the culprit. There is no time to wait for definitive laboratory culture evidence, which accumulates too slowly. For the desperately sick, therapy must begin at once – not 2 days from can either prescribe a broad – spectrum drug that covers all possibilities, or she can prescribed a better, disease – specific drug.

Mycin helps the physician to prescribe disease – specific drugs. Mycin in-forms it self about particular cases by requesting information from the physician about a patient's symptoms, general condition. History, and laboratory – test results that can be obtained easily and quickly. At each point, the question mycin asks is determined by Mycin's current hypothesis and the answers to all previous questions. Thus, the questions start as though taken from a checklist, but the questions then vary as evidence builds. Here is an except from a sample sessions:

For Example:

What is the patient's Name?

John Doe.

- Male or female?

Male

- Age ?

He is 55.

- Have you obtained positive cultures indicating general type ?

yes

- What type of infection is it ?

Primary bacteremia

- When did symptoms first appear?

May 5

- Let's call the most positive culture CI. From what site was CI taken ?

from the blood.

- When?

May 9

- Let's call the first significant organism from this culture UI. Do you know identify of UI?

no

- Is UI a rod or a coccus or something else ?

Rod

- What is the gram stain of UI?

Gram negative.

- Have there been positive culture from sites that are not associated with the

infections about which you are seeking advice?

No

As we proceed through the processing stages of compute vision. We will no doubt be impressed by the similarities and parallel one can draw between vision processing and natural language processing. The - sensor stage in vision corresponds to speech recognition language understanding, the low and intermediate processing levels of vision correspond to syntactic and semantic language processing respectively, and high level processing, in both cases corresponds to the process of building and interpreting high level knowledge structures.

4.1.4 Knowledge Acquisition

Q5. Write a note on knowledge acquisition.

Ans :

The success of knowledge based systems lies in the quality and extent of the knowledge available to the system. Acquiring and validating a large groups of consistent, correlated knowledge is not a trivial problem. This has give the acquisition process an especially important role in the design and implementation of these systems. Consequently,

effective acquisition methods have become one of the principal challenges for the AI researches.

The goals of this branch of AI are the discovery and development of efficient, cost effective methods of acquisition. Some important progress has recently been made in this area with the development of sophisticated editors and some general concepts related to acquisition and learning.

Definition

Knowledge acquisition is the process of adding new knowledge to a knowledge base and refining or otherwise improving knowledge that was previously acquired. Acquisition is usually associated with some purpose such as expanding the capabilities of a system or improving its performance at some specified task. It is goal oriented creation and refinement of knowledge . It may consist of facts, rules , concepts, procedures, heuristics, formulas, relationships, statistics or other useful information. Sources of this knowledge may include one or more of the following.

Experts in the domain of interest

Text Books

Technical papers

Databases

Reports

The environment

To be effective, the newly acquired knowledge should be integrated with existing knowledge in some meaningful way so that nontrivial inferences can be drawn from the resultant body of knowledge .the knowledge should, of course, be accurate, non redundant, consistent (non contradictory), and fairly complete in the sense that it is possible to reliably reason about many of the important conclusions for which the systems was intended.

Types of learning:- Classification or taxonomy of learning types serves as a guide in studying or comparing a differences among them. One can develop learning taxonomies based on the type of knowledge representation used (predicate calculus , rules, frames), the type of knowledge learned (concepts, game playing, problem solving), or by the area of application (medical diagnosis , scheduling , prediction and so on).

The classification is intuitively more appealing and is one which has become popular among machine learning researchers .it is independent of the knowledge domain and the representation scheme is used. It is based on the type of inference strategy employed or the methods used in the learning process. The five different learning methods under this taxonomy are:

- Memorization (rote learning)
- Direct instruction (by being told)
- Analogy
- Induction
- Deduction

Memorization

Learning by memorization is the simplest form of learning. It requires the least amount of inference and is accomplished by simply copying the knowledge in the same form that it will be used directly into the knowledge base. We use this type of learning when we memorize multiplication tables for example.

Direct instruction

A slightly more complex form of learning is by direct instruction. This type of learning requires more understanding and inference than role learning since the knowledge must be transformed into an operational form before being integrated into the knowledge base. We use this type of learning when a teacher presents a number of facts directly to us in a well organized manner.

Analogy

The third type listed, analogical learning, is the process of learning a new concept or solution through the use of similar known concepts or solutions. We use this type of learning when solving problems on an examination where previously learned examples serve as a guide or when we learn to drive a truck using our knowledge of car driving. We make frequent use of analogical learning. This form of learning requires still more inferring than either of the previous forms, since difficult transformations must be made between the known and unknown situations. This is a kind of application of knowledge in a new situation.

Induction

The fourth type of learning is also one that is used frequently by humans. It is a powerful form of learning which, like analogical learning, also requires more inferring than the first two methods. This form

of learning requires the use of inductive inference, a form of invalid but useful inference. We use inductive learning when we formulate a general concept after seeing a number of instances or examples of the concept. For example, we learn the concepts of color, sweet taste after experiencing the sensation associated with several examples of colored objects or sweet foods.

Deduction

The final type of acquisition is deductive learning. It is accomplished through a sequence of deductive inference steps using known facts. From the known facts, new facts or relationships are logically derived. Deductive learning usually requires more inference than the other methods. The inference method used is, of course, a deductive type, which is a valid form of inference.

In addition to the above classification, we will sometimes refer to learning methods as either methods or knowledge-rich methods. Weak methods are general purpose methods in which little or no initial knowledge is available. These methods are more mechanical than the classical AI knowledge-rich methods. They often rely on a form of heuristics search in the learning process.

4.1.5 Shell and Explanations

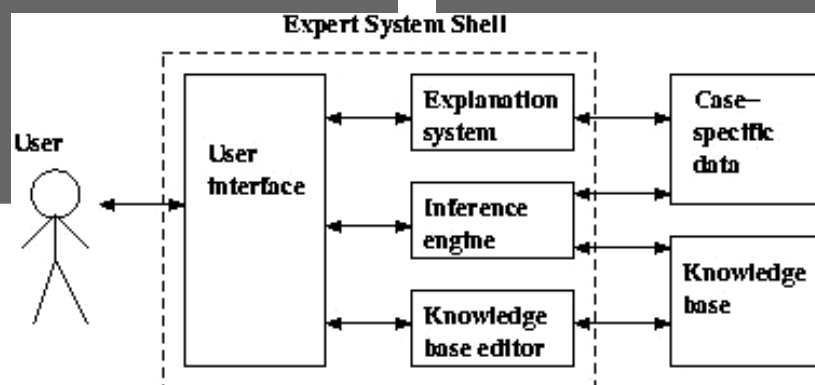
Q6. Explain Expert system shells

Ans :

Expert System Shells

Expert system shells provide methods of building expert systems without extensive knowledge of programming through mechanisms that

- 1) Input the decisions, questions and rules that are followed.
- 2) Construct a knowledge database that can be manipulated by subsequent parts of the system
- 3) Verifies possible violations of surface validity and
- 4) Operates the "inference engine" that operates on the rules, poses the questions to the users, and determines whether a particular decision is valid.



Most expert systems also allow the user to halt the processing at any time to query the system why a question was asked, or how a decision was reached

Most expert system shells can now run easily on most current micro-computers and are able to handle the manipulation of a relatively large number of rules and associated questions.

Expert system shells are expert system development tools consisting essentially of the expert system without the knowledge base, embodying the inference engine, working memory, and the user interface

(Sener, 1991). An example of the inference engine part of an expert system that deduces new conclusions from known facts is illustrated below

IF liquid limit=known

AND plastic limit=known

AND plastic limit>liquid limit

THEN soil=non plastic

Expert systems give advice or solve problems by drawing upon this knowledge stored in the IF/THEN rules.

4.1.6 Self-Explaining System

Q7. Write a note on self-explaining system.

Ans :

Self-explaining has been repeatedly shown to result in positive learning outcomes for students in a wide variety of disciplines. However, there are two potential accounts for why self-explaining works. First, those who self-explain experience more content than those who do not. Second, there are differences in the activity of generating the explanations versus merely comprehending them.

The first studies of self-explanation, which were based on analyses of verbal protocols, showed that the amount of self-explaining correlated strongly with performance on post-test measures of problem-solving performance. Subsequent studies showed that students who were prompted to self-explain sentences in a scientific text learned more than students who were asked to paraphrase the sentences instead. Other studies showed that self-explaining could be elicited by computers.

Because these studies compared self-explanation to the lack of any explanation at all, it is not clear why self-explanation produced learning. On the one hand, self-explaining generates additional information, namely the explanations themselves, that are not present in the instructional materials.

Let us label these hypotheses as follows:

1. Attention

Learning from self-generated explanations should produce comparable learning gains as author-provided explanations, provided

the learner pays attention to them. Both self-generated and author-provided explanations should exhibit better learning than no explanation.

2. Generation

Learning from self-generated explanations should produce greater learning gains than author-provided explanations because they are produced from the students' own background knowledge; however, author-provided explanations should be comparable to no explanation.

4.1.7 Rule-based Expert Systems

Q8. What is rule based expert systems?

Ans :

Rule-Based Expert Systems

An expert system is one designed to model the behaviour of an expert in some field, such as medicine or geology.

Rule-based expert systems are designed to be able to use the same rules that the expert would use to draw conclusions from a set of facts that are presented to the system.

The People Involved in an Expert System

1. The design, development, and use of expert systems involves a number of people.
2. The **end-user** of the system is the person who has the need for the system.
In the case of a medical diagnosis system, this may be a doctor, or it may be an individual who has a complaint that they wish to diagnose.
3. The **knowledge engineer** is the person who designs the rules for the system, based on either observing the expert at work or by asking the expert questions about how he or she works.
4. The **domain expert** is very important to the design of an expert system. In the case of a medical diagnosis system, the expert needs to be able to explain to the knowledge engineer how he or she goes about diagnosing illnesses.

4.1.8 Forward and Backward Chaining

Q9. Write about forward chaining.

Ans :

Forward Chaining

Forward chaining employs the system starts from a set of facts, and a set of rules, and tries to find a way of using those rules and facts to deduce a conclusion or come up with a suitable course of action.

This is known as **data-driven reasoning** because the reasoning starts from a set of data and ends up at the goal, which is the conclusion.

When applying forward chaining, the first step is to take the facts in the fact database and see if any combination of these matches all the antecedents of one of the rules in the rule database.

When all the antecedents of a rule are matched by facts in the database, then this rule is **triggered**.

Usually, when a rule is triggered, it is then **fired**, which means its conclusion is added to the facts database. If the conclusion of the rule that has fired is an action or a recommendation, then the system may cause that action to take place or the recommendation to be made.

For example, consider the following set of rules that is used to control an elevator in a three-story building:

Rule 1

IF on first floor and button is pressed on first floor

THEN open door

Rule 2

IF on first floor

AND button is pressed on second floor

THEN go to second floor

Rule 3

IF on first floor

AND button is pressed on third floor

THEN go to third floor

Rule 4

IF on second floor

AND button is pressed on first floor

AND already going to third floor

THEN remember to go to first floor later

This represents just a subset of the rules that would be needed, but we can use it to illustrate how forward chaining works.

Let us imagine that we start with the following facts in our database:

Fact 1

At first floor

Fact 2

Button pressed on third floor

Fact 3

Today is Tuesday

Now the system examines the rules and finds that Facts 1 and 2 match the antecedents of Rule 3. Hence, Rule 3 fires, and its conclusion "Go to third floor" is added to the database of facts. Presumably, this results in the elevator heading toward the third floor.

Note that Fact 3 was ignored altogether because it did not match the antecedents of any of the rules.

Now let us imagine that the elevator is on its way to the third floor and has reached the second floor, when the button is pressed on the first floor. The fact Button pressed on first floor

is now added to the database, which results in Rule 4 firing.

Conflict Resolution

In a situation where more than one conclusion can be deduced from a set of facts, there are a number of possible ways to decide which rule to fire.

For example, consider the following set of rules:

IF it is cold

THEN wear a coat

IF it is cold

THEN stay at home

IF it is cold

THEN turn on the heat

If there is a single fact in the fact database, which is "it is cold," then clearly there are three conclusions that can be derived. In some cases, it might be fine to follow all three conclusions, but in many cases the conclusions are incompatible.

In one conflict resolution method, rules are given priority levels, and when a conflict occurs, the rule that has the highest priority is fired, as in the following example:

IF patient has pain

THEN prescribe painkillers priority 10

IF patient has chest pain

THEN treat for heart disease priority 100

Here, it is clear that treating possible heart problems is more important than just curing the pain.

An alternative method is the **longest-matching strategy**. This method involves firing the conclusion that was derived from the longest rule.

For example:

IF patient has pain

THEN prescribe painkiller

IF patient has chest pain

AND patient is over 60

AND patient has history of heart conditions

THEN take to emergency room

Here, if all the antecedents of the second rule match, then this rule's conclusion should be fired rather than the conclusion of the first rule because it is a more specific match.

Meta Rules

In designing an expert system, it is necessary to select the conflict resolution method that will be used, and quite possibly it will be necessary to use different methods to resolve different types of conflicts.

For example, in some situations it may make most sense to use the method that involves firing the most recently added rules.

This method makes most sense in situations in which the timeliness of data is important. It might be, for example, that as research in a particular field of medicine develops, and new rules are added to the system that contradicts some of the older rules.

It might make most sense for the system to assume that these newer rules are more accurate than the older rules.

It might also be the case, however, that the new rules have been added by an expert whose opinion is less trusted than that of the expert who added the earlier rules.?

In this case, it clearly makes more sense to allow the earlier rules priority.

This kind of knowledge is called **meta knowledge**—knowledge about knowledge. The rules that define how conflict resolution will be used, and how other aspects of the system itself will run, are called **meta rules**.

The knowledge engineer who builds the expert system is responsible for building appropriate meta knowledge into the system (such as "expert A is to be trusted more than expert B" or "any rule that involves drug X is not to be trusted as much as rules that do not involve X").

Meta rules are treated by the expert system as if they were ordinary rules but are given greater priority than the normal rules that make up the expert system.

Q10. Explain backward chaining.

Ans :

Backward Chaining

Forward chaining applies a set of rules and facts to deduce whatever conclusions can be derived, which is useful when a set of facts are present, but you do not know what conclusions you are trying to prove.

Forward chaining can be inefficient because it may end up proving a number of conclusions that are not currently interesting.

In such cases, where a single specific conclusion is to be proved, **backward chaining** is more appropriate.

In backward chaining, we start from a conclusion, which is the **hypothesis** we wish to prove, and we aim to show how that conclusion can be reached from the rules and facts in the database.

The conclusion we are aiming to prove is called a **goal**, and so reasoning in this way is known as **goal-driven reasoning**.

Backward chaining is often used in formulating plans.

A plan is a sequence of actions that a program decides to take to solve a particular problem.

Backward chaining can make the process of formulating a plan more efficient than forward chaining.

Backward chaining in this way starts with the goal state, which is the set of conditions the agent wishes to achieve in carrying out its plan. It now examines this state and sees what actions could lead to it.

For example, if the goal state involves a block being on a table, then one possible action would be to place that block on the table.

This action might not be possible from the start state, and so further actions need to be added before this action in order to reach it from the start state.

In this way, a plan can be formulated starting from the goal and working back toward the start state.

In this kind of situation, it can be very inefficient to attempt to formulate a plan using forward chaining because it involves examining every possible action, without paying any attention to which action might be the best one to lead to the goal state.

Backward chaining ensures that each action that is taken is one that will definitely lead to the goal, and in many cases this will make the planning process far more efficient.

Compare Forward and Backward chaining with an example

Comparing Forward and Backward Chaining

Let us use an example to compare forward and backward chaining. In this case, we will revert to our use of symbols for logical statements, in order to clarify the explanation, but we could equally well be using rules about elevators or the weather.

Rules:

Rule 1 $A \wedge B \rightarrow C$

Rule 2 $A \rightarrow D$

Rule 3 $C \wedge D \rightarrow E$

Rule 4 $B \wedge E \wedge F \rightarrow G$

Rule 5 $A \wedge E \rightarrow H$

Rule 6 $D \wedge E \wedge H \rightarrow I$

Facts:

Fact 1 A

Fact 2 B

Fact 3 F

Goal

Our goal is to prove H.

- First let us use forward chaining. As our conflict resolution strategy, we will fire rules in the order they appear in the database, starting from Rule 1.
- In the initial state, Rules 1 and 2 are both triggered. We will start by firing Rule 1, which means we add C to our fact database. Next, Rule 2 is fired, meaning we add D to our fact database.
- We now have the facts A, B, C, D, F, but we have not yet reached our goal, which is G.
- Now Rule 3 is triggered and fired, meaning that fact E is added to the database.
- As a result, Rules 4 and 5 are triggered. Rule 4 is fired first, resulting in Fact G being added to the database, and then Rule 5 is fired, and Fact H is added to the database.
- We have now proved our goal and do not need to go on any further.

- This deduction is presented in the following table:

Facts	Rules triggered	Rule fired
A, B, F	1, 2	1
A, B, C, F	2	2
A, B, C, D, F	3	3
A, B, C, D, E, F	4, 5	4
A, B, C, D, E, F, G	5	5
A, B, C, D, E, F, G, H	6	STOP

Now we will consider the same problem using backward chaining. To do so, we will use a goals database in addition to the rule and fact databases.

In this case, the goals database starts with just the conclusion, H, which we want to prove. We will now see which rules would need to fire to lead to this conclusion.

Rule 5 is the only one that has H as a conclusion, so to prove H, we must prove the antecedents of Rule 5, which are A and E.

Fact A is already in the database, so we only need to prove the other antecedent, E. Therefore, E is added to the goal database. Once we have proved E, we now know that this is sufficient to prove H, so we can remove

H from the goals database.

So now we attempt to prove Fact E. Rule 3 has E as its conclusion, so to prove E, we must prove the antecedents of Rule 3, which are C and D.

Neither of these facts is in the fact database, so we need to prove both of them. They are both therefore added to the goals database. D is the conclusion of Rule 2 and Rule 2's antecedent, A, is already in the fact database, so we can conclude D and add it to the fact database.

Similarly, C is the conclusion of Rule 1, and Rule 1's antecedents, A and B, are both in the fact database. So, we have now proved all the goals in the goal database and have therefore proved H and can stop.

This process is represented in the table below:

Facts	Goals	Matching rules
A, B, F	H	5
A, B, F	E	3
A, B, F	C, D	1
A, B, C, F	D	2
A, B, C, D, F		STOP

In this case, backward chaining needed to use one fewer rule. If the rule database had had a large number of other rules that had A, B, and F as their antecedents, then forward chaining might well have been even more inefficient.

In general, backward chaining is appropriate in cases where there are few possible conclusions (or even just one) and many possible facts, not very many of which are necessarily relevant to the conclusion.

Forward chaining is more appropriate when there are many possible conclusions.

The way in which forward or backward chaining is usually chosen is to consider which way an expert would solve the problem. This is particularly appropriate because rule-based reasoning is often used in **expert systems**.

4.1.9 Frame-based Expert Systems

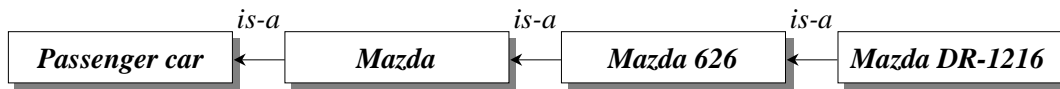
Q11. Explain frame-based expert systems.

Ans :

- A frame is a data structure with typical knowledge about a particular object or concept.
- Each frame has its own name and a set of attributes associated with it. *Name, weight, height and age* are slots in the frame *Person*. *Model, processor, memory and price* are slots in the frame *Computer*. Each attribute or slot has a value attached to it.
- Frames provide a natural way for the structured and concise representation of knowledge.

<ul style="list-style-type: none"> ➤ A frame provides a means of organising knowledge in slots to describe various attributes and characteristics of the object. ➤ Frames are an application of object-oriented programming for expert systems. ➤ Object-oriented programming is a programming method that uses <i>objects</i> as a basis for analysis, design and implementation. ➤ In object-oriented programming, an object is defined as a concept, abstraction or thing with crisp boundaries and meaning for the problem at hand. All objects have identity and are clearly distinguishable, 	<p>member of the class <i>Computer</i>, which in turn might belong to the class <i>Hardware</i>.</p> <ol style="list-style-type: none"> 3. Slot value : A slot value can be symbolic, numeric or Boolean. For example, the slot <i>Name</i> has symbolic values, and the slot <i>Age</i> numeric values. Slot values can be assigned when the frame is created or during a session with the expert system 3. Default slot value : The default value is taken to be true when no evidence to the contrary has been found. For example, a car frame might have four wheels and a chair frame four legs as default values in the corresponding slots.
<ul style="list-style-type: none"> ➤ An object combines both data structure and its behaviour in a single entity. This is in sharp contrast to conventional programming, in which data structure and the program behaviour have concealed or vague connections. ➤ When an object is created in an object-oriented programming language, we first assign a name to the object, then determine a set of attributes to describe the object's characteristics, and at last write procedures to specify the object's behaviour. ➤ A knowledge engineer refers to an object as a frame 	<ol style="list-style-type: none"> 4. Range of the slot value : The range of the slot value determines whether a particular object complies with the stereotype requirements defined by the frame. For example, the cost of a computer might be specified between \$750 and \$1500. 5. Procedural information : A slot can have a procedure attached to it, which is executed if the slot value is changed or needed. 6. Frame-based expert systems also provide an extension to the slot-value structure through the application of <i>facets</i>. 7. A facet is a means of providing extended knowledge about an attribute of a frame.
<p>Frames as a knowledge Representation Technique</p> <ol style="list-style-type: none"> 1. The concept of a frame is defined by a collection of slots. Each slot describes a particular attribute or operation of the frame. 2. Slots are used to store values. A slot may contain a default value or a pointer to another frame, a set of rules or procedure by which the slot value is obtained. 	<p>Facets are used to establish the attribute value, control end-user queries, and tell the inference engine how to process the attribute.</p> <p>Class Inheritance in Frame-based Systems</p> <p>Frame-based systems support class inheritance.</p>
<p>Typical Information included in a Slot</p> <ol style="list-style-type: none"> 1. Frame name. 2. Relationship of the frame to the other frames: The frame <i>IBM Aptiva S35</i> might be a 	<p>The fundamental idea of inheritance is that attributes of the class-frame represent things that are <i>typically</i> true for all objects in the class. However, slots in the instance-frames can be filled with actual data uniquely specified for each instance.</p>

Relations of the Car Frames



Inheritance of Slot Values

CLASS: <i>Passenger car</i>	CLASS: <i>Mazda</i>
[C] Engine type <i>In-line 4 cylinder:</i> <i>V6:</i>	Superclass: <i>Passenger car</i>
[N] Horsepower:	[C] Engine type <i>In-line 4 cylinder:</i> <i>V6:</i>
[C] Drivetrain type <i>Rear wheel drive:</i> <i>Front wheel drive:</i> <i>Four wheel drive:</i>	[N] Horsepower:
[C] Transmission type <i>5-speed manual:</i> <i>4-speed automatic:</i>	[C] Drivetrain type <i>Rear wheel drive:</i> <i>Front wheel drive:</i> <i>Four wheel drive:</i>
[N] Fuel consumption (mpg):	[C] Transmission type <i>5-speed manual:</i> <i>4-speed automatic:</i>
[N] Seating capacity:	[N] Fuel consumption (mpg):
	[N] Seating capacity:
	[Str] Country of manufacture: <i>Japan</i>

Methods and Demons

Expert systems are required not only to store the knowledge but also to validate and manipulate this knowledge. To add actions to our frames, we need methods and demons.

1. A method is a procedure associated with a frame attribute that is executed whenever requested.
2. We write a method for a specific attribute to determine the attribute's value or execute a series of actions when the attribute's value changes.
3. Most frame-based expert systems use two types of methods:
WHEN CHANGED and WHEN NEEDED.

Interaction of Frames and Rules

Most frame-based expert systems allow us to use a set of rules to evaluate information contained in frames.

1. In a rule-based expert system, the inference engine links the rules contained in the knowledge base with data given in the database.
2. When the goal is set up, the inference engine searches the knowledge base to find a rule that has the goal in its consequent.
3. If such a rule is found and its IF part matches data in the database, the rule is fired and the specified object, the goal, obtains its value. If no rules are found that can derive a value for the goal, the system queries the user to supply that value.
4. In a frame-based system, the inference engine also searches for the goal.

But:

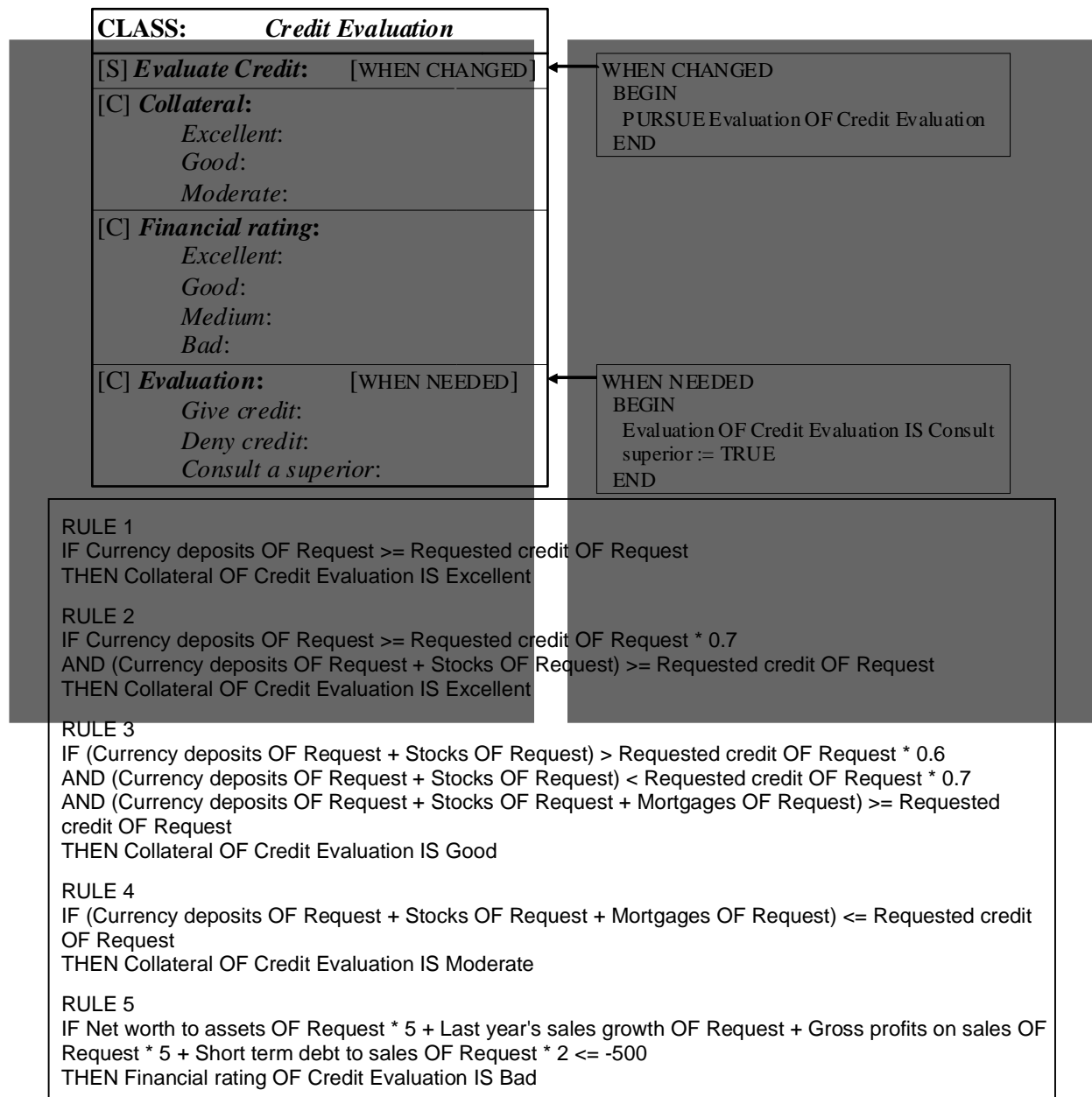
5. In a frame-based system, rules play an auxiliary role. Frames represent here a major source of knowledge, and both methods and demons are used to add actions to the frames.
6. Thus, the goal in a frame-based system can be established either in a method or in a demon.

Example

Suppose we want to evaluate the credit request selected by the user. The expert system is expected to begin the evaluation when the user clicks the *Evaluate Credit* push-button on the input display.

This pushbutton is attached to the attribute Evaluate Credit of the class Credit Evaluation.

The *Credit Evaluation* class, WHEN CHANGED and WHEN NEEDED methods



- Based on the set of rules provided for credit evaluation, the inference engine cannot establish the value of the attribute Evaluation in some cases.
 - We can use the WHEN NEEDED method to establish the attribute value.
 - The WHEN NEEDED method is attached to the attribute Evaluation. The inference engine executes this method when it needs to determine the value of Evaluation. When the WHEN NEEDED method is executed, the attribute Evaluation receives the value Consult a superior.
-

4.1.10 Uncertainty Management in Expert Systems

Q12. Explain uncertainty management in expert systems.

Ans:

The information available to humans is often imperfect. An expert can cope with defects.

- Classical logic permits only exact reasoning
- IF A is true THEN A is \neg false and
- IF B is false THEN B is \neg true
- Most real-world problems do not provide exact information. It can be inexact, incomplete or even immeasurable.

Uncertainty is defined as the lack of the exact knowledge, that would enable us to reach a perfectly reliable conclusion.

- Classical logic permits only exact reasoning. It assumes that perfect knowledge always exists and the law of the excluded middle can always be applied.

\neg TRUE = FALSE.

Sources of uncertain knowledge

- Weak implications:
- Domain experts and knowledge engineers must establish concrete correlations between **IF and THEN parts of the rules.**
- Therefore, expert systems need to have the ability to handle vague associations, for example by accepting the degree of correlations as numerical certainty factors.
- Imprecise Language: Our natural language is ambiguous and imprecise. As a result, it can be difficult to express knowledge in the precise.

IF-THEN form of production rules. Expert systems need quantified measures.

Unknown Data

When the data is incomplete or missing, the only solution is to accept the value "unknown" and proceed to an approximate reasoning with this value.

- Disagreement among experts: Weighting associated to different expert opinions.

Examine the Uncertainty

- Probabilistic reasoning
- Probability Theory Basics
- Bayesian Reasoning

Basics of Probability Theory

When examining uncertainty, we adopt probability as a model to predict future events.

$$P_{(\text{success})} = P = \frac{\text{Number of successes}}{\text{Number of possible outcomes}}$$

$$= \frac{S}{S+f} \quad \dots (1)$$

And likewise for failures, q. Now let A be some event and B be some other event. These are not mutually exclusive.

The **conditional probability that event A will occur**, GIVEN that B has occurred is $P(A|B)$.

$$P(A|B) = \frac{\text{Number of times both A and B can occur}}{\text{Number of times B can occur}} \quad \dots (2)$$

Bayesian Rule

The probability of both A and B both occurring, denoted $P(A \cap B)$ is the joint probability.

$$P(A \cap B) = p(A|B) \times p(B) \quad \dots (3)$$

and is commutative (i.e., $P(A \cap B) = P(B \cap A)$)

This allows us to derive the famous Bayesian Rule.

$$p(A|B) = \frac{p(B|A) \times p(A)}{p(B)} \quad \dots (4)$$

➤ If A is conditionally dependent on n other mutually exclusive events then:

$$p(A) = \sum_{i=1}^n p(A|B_i) \times p(B_i) \quad \dots (5)$$

Dependent Events that are Mutually Exclusive

We shall now consider the case where A depends on two mutually exclusive events, B and $\neg B$. From Equation 5

$$p(B) = (p(B|A) \times p(A)) + (p(B|\neg A) \times p(\neg A))$$

and substituting this into Bayesian Rule (Equation 4) gives

$$p(A|B) = \frac{p(B|A) \times p(A)}{(p(B|A) \times p(A)) + (p(B|\neg A) \times p(\neg A))} \quad \dots (6)$$

➤ Equation 6 is used in the management of uncertainty in expert systems.

Reasoning in Expert Systems

Armed with Equation 6, we shall try to manage uncertainty in expert systems.

➤ Suppose rules in a knowledge base are represented as follows:

Uncertain Rules

IF H is true THEN E is true, with probability p

1. If event E has occurred, then probability of occurrence of H can be obtained by
2. Equation 6, replacing for A and B. In this case, H is the hypothesis and E is the evidence. Rewriting Eq. 6 in terms of H and E:

$$p(H|E) = \frac{p(E|H) \times p(H)}{(p(E|H) \times p(H)) + (p(E|\neg H) \times p(\neg H))} \quad \dots (7)$$

Generalising to m hypotheses and n Evidences

Single evidence E and m hypotheses imply:

$$p(H_i|E) = \frac{p(E|H_i) \times p(H_i)}{\sum_{k=1}^m (p(E|H_k) \times p(H_k))} \quad \dots (8)$$

Suppose the expert, given multiple (n) evidences, cannot distinguish between m hypotheses:

$$p(H_i|E_1 \dots E_n) = \frac{p(E_1 \dots E_n | H_i) \times p(H_i)}{\sum_{k=1}^m p(E_1 \dots E_n | H_k) \times p(H_k)} \quad \dots (9)$$

An application of Eq. 9 requires us to obtain the conditional probabilities of all possible combinations of evidences for all hypotheses! This grows exponentially. Therefore, assume conditional independence if possible.

Probabilistic Reasoning in Expert Systems

Let the **posterior probability of hypothesis H_i upon observing** evidences $E_1 \dots E_n$ be,

$$p(H_i|E_1 \dots E_n) = \frac{p(E_1 | H_i) \times p(E_n | H_i) \times p(H_i)}{\sum_{k=1}^m p(E_1 | H_k) \times \dots \times p(E_n | H_k) \times p(H_k)} \quad \dots (10)$$

- This is a far more tractable solution and assumes conditional independence among different evidences.
- Users provide information about the evidence observed and the expert system computes $p(H|E)$ for hypothesis H in light of the user-supplied evidence E.

Probability $p(H|E)$ is called the **posterior probability** of hypothesis H upon observing evidence E.

4.1.11 Expert System and DSS

Q13. Distinguish between expert systems and DSS.

Ans :

The differences between Expert systems and DSS are the following:

(a) Terminology

The terminology in both fields, ES and DSS, is confusing. For example, if the term "Decision Support System" is taken literally, most computer programs would in fact be DSSs: one way or another they

·support decision· since they provide certain kinds of information used in a decision process. Then there is a host of names implying somehow that DSS and ES are indeed very similar concepts, e.g. ESM (Expert Systems for Management), suggesting that management (or substantial parts of it) can be done by ESs; IMS (Intelligent Management System), which suggests that management can be automated by applying techniques from Artificial Intelligence (e.g. ES), etc.

Thus, the terminology in these fields should never be taken literally.

(b) Software Dominance

With the advent of ES a new programming methodology has emerged, namely the one of declarative specifications as opposed to procedural programs. Declarative specification is exemplified in production system languages (e.g. OPS5), and logic programming languages such as PROLOG. This has led a number of authors to identify ES with logic programming. Any program employing logic programming (or other kinds of declarative) techniques is called an ES, a point of view the authors clearly reject. But the fact that DSS and ES may use a similar programming methodology does not imply that they are equal.

4.1.12 Pros and Cons of Expert Systems

Q14. Write about the advantages and disadvantages of expert systems.

Ans :

The Advantages of Using Expert System

1. Providing Consistent Solutions

It can provide consistent answers for repetitive decisions, processes and tasks. As long as the rule base in the system remains the same, regardless of how many times similar problems are being tested, the final conclusions drawn will remain the same.

2. Provides Reasonable Explanations

It has the ability to clarify the reasons why the conclusion was drawn and be why it is

considered as the most logical choice among other alternatives. If there are any doubts in concluding a certain problem, it will prompt some questions for users to answer in order to process the logical conclusion.

3. Overcome Human Limitations

It does not have human limitations and can work around the clock continuously. Users will be able to frequently use it in seeking solutions. The knowledge of experts is an invaluable asset for the company. It can store the knowledge and use it as long as the organization needs.

4. Easy to Adapt to New Conditions

Unlike humans who often have troubles in adapting in new environments, an expert system has high adaptability and can meet new requirements in a short period of time. It also can capture new knowledge from an expert and use it as inference rules to solve new problems.

The Disadvantages of Using Expert System

1. Lacks Common Sense

It lacks common sense needed in some decision making since all the decisions made are based on the inference rules set in the system. It also cannot make creative and innovative responses as human experts would in unusual circumstances.

2. High Implementation and Maintenance Cost

The implementation of an expert system will be a financial burden for smaller organizations since it has high development cost as well as the subsequent recurring costs to upgrade the system to adapt in new environment.

3. Difficulty in Creating Inference Rules

Domain experts will not be able to always explain their logic and reasoning needed for the knowledge engineering process. Hence, the task of codifying out the knowledge is highly complex and may require high

4. May Provide Wrong Solutions

It is not error-free. There may be errors occurred in the processing due to some logic mistakes made in the knowledge base, which it will then provide the wrong solutions.

4.1.13 Case Study

Q15. Explain the case study of expert systems

Ans :

Case Study: A Course Advisor Expert System

The Problem Domain

The present University course description Web-based system exhibits some limitations. Occasionally, subject pages display erroneous or stale information. Students may be unintentionally misled in their enrolments / exams etc which could adversely affect the image of the teaching institution. Also, the present system does not verify that a student may enrol in subjects in a way that may contradict University policies.

The advisory expert system is intended to provide preliminary assistance and advice for prospective postgraduates, extracting and inferring the necessary information from a current knowledge base within a consultation session. This should provide a realistic assessment of the alternatives, requirements and expectations for a particular student, and thus help prevent enrolment errors.

In the current teaching system, subjects are considered to be an indivisible entity, non-customisable, with a fixed number of points awarded on completion. Even if a student has prior knowledge covering part of the course, he / she still has to enrol in the *whole* course and will be awarded *all* of the points allocated to the course if successful. In order to enable and support the proposed expert system, the teaching system should allow course *modularity*, with points allocated to course components, separately assessed.

Developing the Expert System

As previously stated, in developing the expert system it is desirable to start with a *prototype* of the complete expert system, which would provide information on the feasibility of the project without full financial or resource commitment. This prototype may then be submitted for evaluation to users / stakeholders and so obtain their feedback and commitment. User and host institution acceptance is a multidimensional aspect, which ultimately decides the usefulness of the entire system development effort. It must be also emphasized that

the prototype needed the elicitation of knowledge from at least one domain expert. The development of the prototype could not be left to the knowledge engineer alone.

Life Cycle Models

Several life cycle paradigms have been considered such as waterfall, incremental, linear, spiral,, etc. A modified *spiral* paradigm may be successfully employed, providing that some guidelines are followed regarding the newly added facts and rules. Such constraints may be: maintain integrity (must not contradict the existing facts and rules), avoid redundancy (must not represent knowledge already existent in the rule base), prevent scattering the knowledge over an excessive amount of rules/facts, etc. Thus, a balance must be struck between the facts /rules complexity, expressively and number.

Concepts of the Expert System

The expert system will be based on several concepts aiming to improve the subject selection and education processes. These concepts are *modularity* of subjects, *prerequisites* and *outcomes* for subject modules and *credit* for previous studies, applied to modules rather than subjects.

Hence, the aim is to establish *modules* within the subjects, with their own prerequisites, outcomes and credit points awarded on completion. The number of modules within a subject must maintain a balance between flexibility and the amount of processing and development time required.

User Requirements for the Expert System Prototype

The user will provide a preferred type of occupation (targeted set of skills) and previous knowledge, usable to satisfy some of the prerequisites for the modules composing the study program. The system will provide a study course to achieve the targeted occupation and may also give correction advice if existing user skills are stated as too limited or too high.

Design of the Expert System Prototype

System Requirements

The system requirements represent a translation of the user requirements into the from:

system domain. For this particular case they may take the form:

1. The expert system must rely on the user's tacit knowledge rather than trying to model it;
2. Modules are seen as *objects* having *interfaces* - which are in fact the prerequisites and outcomes of a module. Prerequisites and outcomes are items contained in *special lists*, which are further contained within *module facts*;
3. A module prerequisite may be satisfied by one (and only one) outcome, be it of another module or declared as 'known' by the user (previously acquired skills);
4. The consultation starts with a set of initial facts, asserted at run-time according to the user's answers to 'job domain' and 'job type' queries. These facts provide the initial list of unsatisfied prerequisites.
5. The system attempts to match these unsatisfied prerequisites: first, against outcomes declared 'known' by the user, and then (if unsuccessful) against the outcomes lists of the other module facts in the knowledge base;
6. When a matching outcome is found for a prerequisite, the module owning the outcome is marked as 'needed' and its *prerequisites* list is then in its turn scanned for matches with other outcomes. The prerequisite for which the match has been found is marked 'done'.
7. Any prerequisites found to be not either declared 'known' by the user, or already satisfied by other modules' outcomes will trigger *additional questions* for the user;
8. According to the user's answers, the prerequisites are either declared 'known' (if the user happens to have the required skill), or added to the list of unsatisfied
9. Prerequisites; the process is repeated until all prerequisites are satisfied - either by other modules' outcomes ('done') or by previous skills of the user ('known').
10. The list of all modules marked as 'needed' (that is, modules the user will need to enrol

in) is printed out. **NB** some special modules are automatically added, such

11. As the project modules (which will be present regardless of the user's prior knowledge)

Knowledge Representation and Method of Inference

Design decisions had to be made regarding the formalism to be used in representing the knowledge contained in the teaching system. A first criterion is related to the type of knowledge to be represented, namely the components of the various courses composing current study programs. As such, this knowledge matched the form of a collection of isolated facts, rather than a structured set of knowledge or a concise theory. Therefore, rules / assertions were preferred to frames or algorithms in a first decision round. Production rules have been chosen in preference to logic and semantic nets / frames, since they are more suitable for solving design and planning problems. Bottom-up reasoning / inference, whereby a starting fact set (provided by the user) is matched against the conditions of the production rules (in the rule base, constructed upon the rules governing the study programs within the University) will also be used. The use of bottom-up inference leads to forward chaining as a preferred model of conflict resolution. However, prioritisation may also be involved by assigning production rules appropriate weights.

Knowledge Acquisition

The preferred methods of knowledge acquisition for the prototype have been chosen to be the use of questionnaires and structured interview. This decision owes to several factors such as the size of the prototype, the nature of problem domain and the availability of domain experts. Questionnaires are well suited to future automated processing which is likely to benefit the knowledge acquisition process. The design of both questionnaire and interview have acknowledged the gap between the descriptions of domain specialists (subject conveners) and the resulting computational models and also the social issues underlying knowledge creation. The interview design has mainly followed the COMPASS procedure.

Design Constraints

Constraints are necessary in order to enable a finite solution to be produced.

Examples

- The outcomes of a module must be distinct; two modules may not produce the same outcome: doing so would produce a redundancy in the teaching structure which needs to be resolved
- All the module prerequisites contained in the knowledge base are satisfied by outcomes of other modules in the base.
- Nil prerequisites for modules are allowed; however, they still require basic graduate knowledge, such as maths, physics, etc (the tacit knowledge);
- Cyclic dependencies between any two modules are disallowed (e.g. if module A has a prerequisite satisfied by module B, module B must not have a prerequisite satisfied only by module A). Should this situation occur, the offending modules must be reassessed with regards to their prerequisites / outcomes;

Further constraints may be added in the developing the expert system, e.g.:

- Maximum number of year **n** modules: may conflict with the concept of a Conversion Course and limit the flexibility of the expert system.
- Maximum number of modules per Semester. The present prototype does not obey this constraint. In real life, subjects tend to be equally distributed in all Semesters.

Balanced number of modules in each Semester: see previous.

The Expert System Conceptual Model

Once the decision to build the prototype as a production rule-based expert system has been taken, the main elements of the system may be further specified. The knowledge base should contain *facts* and *rules* referring to the prerequisites and outcomes of modules of the subjects offered in the University¹⁸. The facts are either 'fixed' (such as the modules information) or run-time asserted (e.g. the user's answers to the expert systems' questions). The inference engine must be chosen to match previous requirements and design decisions. The user interface, preferably graphical and integratable with currently and commonly used operating systems and enabling technology infrastructure (e.g. Internet), will preferably be implemented in the same language as the inference engine.

A Unified Modelling Language model of the expert system is presented in **Figure**. In this figure, the user-expert system interaction occurs through the user interface, which sends the problem conditions (answers to questions) to the work area that holds all the temporary data. The inference engine uses the knowledge base for the rules and fixed facts, and the work area for the dynamic facts (asserted at run-time). The knowledge base contains the fixed facts and the rules. Finally, the solution is delivered to the user interface. The classes shown in the figure will be further specified in the Implementation section.

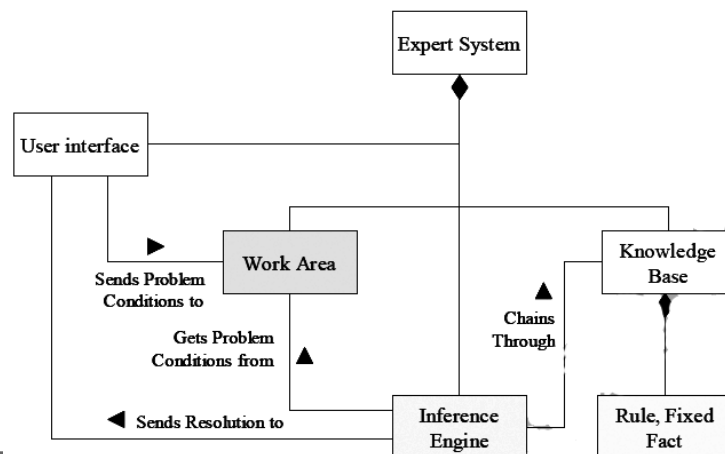
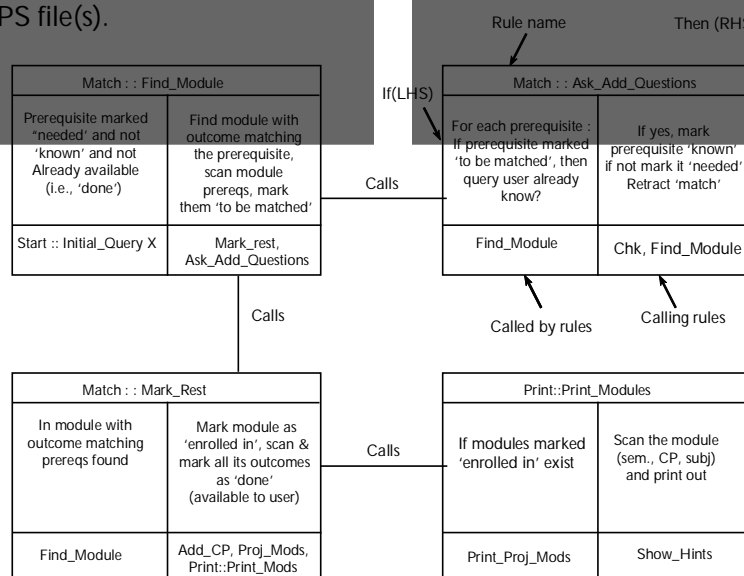


Fig. : Object Diagram of Expert System

Implementation

Several decisions had to be made in the implementation of the expert system. The virtual machine hierarchy described in provides a good guide in establishing the options. Several web-enabled expert system shells have been considered for the specific problem domain. While the shell may assist in the effort of knowledge representation, it has to be matched to the task. Most shells impose a particular production rule formalism, chaining and structure to the rule set. Considering the size of the prototype, the resources available and the problem domain, the choice has been an expert system shell written in Java, which emulates the CLIPS language, together with a simple graphical user interface. The Java implementation of CLIPS is called JESS - The Java Expert System Shell. This provides the user with the power of the CLIPS language and the flexibility of the Java cross-platform concept. Most of the data structures provided by Java are available in JESS, including the AWT (Abstract Window Toolkit), which enables the user to construct graphical user interfaces to the JESS (CLIPS) engine. Consult has been used as an off-the-shelf basic JESS graphical user interface. The inference engine is indirectly provided CLIPS. The CLIPS language uses forward-chaining and the RETE fast pattern-matching algorithm. CLIPS uses lists to process information, very similar to the LISP language. In view of the implementation decision, **Figure** may now be further explained. Thus, the Work Area is the Java applet, the user interface is the applet's window, the inference engine is provided by the Java CLIPS implementation and the knowledge base resides in CLIPS file(s).



4.2 PATTERN RECOGNITION

4.2.1 Machine Perception And Pattern Recognition

Q16. What is Machine Perception?

Ans :

Machine perception is the capability of a computer system to interpret data in a manner that is similar to the way humans use their senses to relate to the world around them. The basic method that the computers take in and respond to their environment is through the attached hardware. Until recently input was limited to a keyboard, or a mouse, but advances in technology, both in hardware and software, have allowed computers to take in sensory input in a way similar to humans.

Machine perception allows the computer to use this sensory input, as well as conventional computational means of gathering information, to gather information with greater accuracy and to present it in a way that is more comfortable for the user. These include computer vision, machine hearing, and machine touch.

The end goal of machine perception is to give machines the ability to see, feel and perceive the world as humans do and therefore for them to be able to explain in a human way why they are making their decisions, to warn us when it is failing and more importantly, the reason why it is failing.

Machine Vision

Computer vision is a field that includes methods for acquiring, processing, analyzing, and understanding images and, in general, high-dimensional data from the real world in order to produce numerical or symbolic information, e.g., in the forms of decisions. Computer vision has many applications already in use today such as facial recognition, geographical modeling, and even aesthetic judgment.

Machine Hearing

Machine hearing, also known as machine listening or computer audition, is the ability of a computer or machine to take in and process sound data such as music or speech. This area has a wide range of application including music recording and

compression, speech synthesis, and speech recognition. Moreover, this technology allows the machine to replicate the human brain's ability to selectively focus in a specific sound against many other competing sounds and background noise. This particular ability is called "auditory scene analysis". The technology enables the machine to segment several streams occurring at the same time. Many commonly used devices such as a smart phones, voice translators, and cars make use of some form of machine hearing.

Machine Touch

Machine touch is an area of machine perception where tactile information is processed by a machine or computer. Applications include tactile perception of surface properties and dexterity whereby tactile information can enable intelligent reflexes and interaction with the environment.

Q17. What is Pattern Recognition and Explain about the approaches of pattern recognition.

Ans :

Pattern recognition can be defined as the categorization of input data into identifiable classes via the extraction of significant features or attributes of the data from a background of irrelevant detail.

The field of pattern recognition is concerned mainly with the description and analysis

of measurements taken from physical or mental processes. It consists of acquiring raw data and taking actions based on the "class" of the patterns recognized in the data.

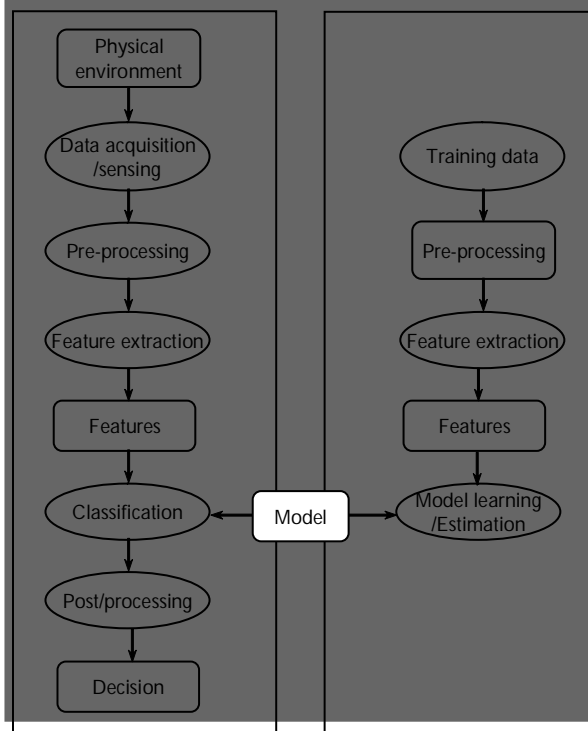
The design of a pattern recognition system essentially involves the following three aspects: data representation, Classification and finally, Prototyping. The problem domain dictates the choice of sensors, pre-processing techniques, representation scheme, and decision making model.

- i) Representation : It describes the patterns to be recognized;
- ii) Classification : It recognizes the "category" to which the patterns provided belong to;
- iii) Prototyping : It is the mechanism used for developing the prototypes or models.

Prototypes are used for representing the different classes to be recognized.

A general pattern recognition system is shown in the Figure.

In the first step data is acquired and preprocessed, this step is followed by feature extraction, feature reduction and grouping of features, and finally the features are classified. In the classification step, the trained classifier assigns the input pattern to one of the pattern classes based on the measured features. The training set used during construction of the classifier is different from the test set which is used for evaluation. This ensures different performance environment.



Pattern Recognition Approaches

Patterns generated from the raw data depend on the nature of the data. Patterns may be generated based on the statistical feature of the data.

The four best-known approaches for the pattern recognition are:

1. Template matching
2. Statistical classification
3. Syntactic matching
4. Neural networks

Template Matching

One of the simplest and earliest approaches to pattern recognition is based on template matching. Matching is carried out to determine the similarity between two entities such as points, curves, or shapes of the same type. In template matching, a template or a prototype of the pattern to be recognized is available. The pattern to be recognized is matched against the stored template while taking into account all allowable operations such as translation, rotation and scale changes.

Statistical Pattern Recognition

The statistical pattern recognition approach assumes statistical basis for classification of data. It generates random parameters that represent the properties of the pattern to be recognized. The main goal of statistical pattern classification is to find to which category or class a given sample belongs. Statistical methodologies such as statistical hypothesis testing, correlation and Bayes classification are used for implementing this method. The effectiveness of the representation is determined by how well pattern from different classes are well separated.

Syntactic Pattern Recognition

In many situations there exist interrelationship or interconnection between the features associated with a pattern. In such circumstances it is appropriate to assume a hierarchical relationship where a pattern is viewed as being consist of simple sub patterns which are themselves built with yet another sub pattern. This is the basis of Syntactic pattern recognition. In this method symbolic data structures such as arrays, strings, trees, or graphs are used for pattern representation.

Neural Network

Neural computing is based on the way by which biological neural system store and manipulates information. It can be viewed as parallel computing environment consisting of interconnection of large number of simple processors. Neural network have been successfully applied in many tasks of pattern recognition and machine learning systems. The structure of neural system is drawn from analogies with biological neural systems.

Neural network models uses a network of weighted directed graphs in which the nodes are

artificial neurons and directed edges are connections between neuron outputs and neuron inputs. The neural networks have the ability to learn complex nonlinear input-output relationships, use sequential training procedures, and adapt themselves to the data.

4.2.2 Feature Extraction

Q18. Write about Feature Extraction process in Pattern recognition.

Ans :

Feature selection is the process of choosing input to the pattern recognition system. Many methods can be used to extract the features. The feature selected is such that it is relevant to the task at hand. These features can be obtained from the mathematical tools or by applying feature extraction algorithm or operator to the input data. The level at which these features are extracted determines the amount of necessary preprocessing and may influence the amount of error introduced into the feature extracted. Features may be represented as continuous, discrete, or discrete binary variables. During the features extraction phase of the recognition process objects are measured. A measurement is the value of some quantifiable property of an object. A feature is a function of one or more measurements, computed so that it quantifies some significant characteristic of the object. This process produces a set of features that, taken together, forms the feature vector.

A number of transformations can be used to generate features. The basic idea is to transform a given set of measurements to a new set of features. Transformation of features can lead to a strong reduction of information as compared with the original input data. In most of the situations relatively small number of features is sufficient for correct recognition.

Obviously feature reduction is a sensitive procedure since if the reduction is done incorrectly the whole recognition system may fail or may not produce the expected results

Feature extraction also depends on application in hand and may use different techniques such as moment-based features, chain codes, and parametric models to obtain required features.

4.2.3 Classification

Q19. What are known as classifiers in expert system.

Ans :

Classifiers are designed to perform the classification stage of the pattern recognition system. A Classifier partitions the feature space into different regions. The border of each decision region is a decision boundary. The determination of region to which the feature vector belongs to is a challenging task. There are many approaches for the design of the classifier in a pattern recognition system and they can be grouped in three classes: classifiers based on Bayes decision theory, linear and nonlinear classifiers.

The first approach builds upon probabilistic arguments stemming from the statistical nature of the generated features. This is due to the statistical variation of the patterns as well as to possible noise obtained in the signal acquisition phase. The objective of this type of design is to classify an unknown pattern in the most probable class as deduced from the estimated probability density functions. Even though linear classifiers are more restricted in their use.

The major advantage is their simplicity and computational demand in solving problems which do not require more sophisticated nonlinear model. Examples of linear classifiers are the perceptrons algorithm and least squares methods. For problems that are not linearly separable and for which the design of a linear classifier, even in an optimal way, does not lead to satisfactory performance, the use of nonlinear classifier are mandatory.

4.2.4 Object Recognition

Q20. What Is Object Recognition? Explain object recognition techniques.

Ans :

Object recognition is a process for identifying a specific object in a digital image or video. Object recognition algorithms rely on matching, learning, or pattern recognition algorithms using appearance-based or feature-based techniques.

Object recognition is useful in applications such as video stabilization, advanced driver assistance systems (ADAS), and disease identification in bio-imaging. Common techniques include deep learning based approaches such as convolutional neural networks, and feature-based approaches using edges, gradients, histogram of oriented gradients (HOG), Haar wavelets, and linear binary patterns.

Object Recognition Techniques

You can recognize objects using a variety of models, including:

- Feature extraction and machine learning models
- Deep learning models such as CNNs
- Bag-of-words models with features such as SURF and MSER
- Gradient-based and derivative-based matching approaches
- The Viola-Jones algorithm, which can be used to recognize a variety of objects, including faces and upper bodies
- Template matching
- Image segmentation and blob analysis

4.2.5 Speech Recognition

Q21. What is speech recognition?

Ans :

Speech recognition is the process of extracting text transcriptions or some form of meaning from speech input.

Speech analytics can be considered as the part of the voice processing, which converts human speech into digital forms suitable for storage or transmission computers.

Speech synthesis function is essentially reverse speech analysis-they convert speech data from digital form to one that is similar to the original entry and is suitable for playback.

Speech analysis processes can also be called digital speech coding (or encoding) and The high variability due to local scale as shown in processing of time signals requires devices with memory. This issue calls the problem of temporary structures.

Q22. Explain about the Basic concepts of speech recognition.

Ans :

Basic Concepts of Speech Recognition

1. Structure of speech
2. Recognition process
3. Models
4. Other used concepts

1. Structure of Speech

Speech is a continuous audio stream where rather stable states mix with dynamically changed states. In this sequence of states, one can define more or less similar classes of sounds, or **phones**. Words are understood to be built of phones, but this is certainly not true. The acoustic properties of a waveform corresponding to a phone can vary greatly depending on many factors - phone context, speaker, style of speech and so on. The so-called coarticulation makes phones sound very different from their "canonical" representation. Next, since transitions between words are more informative than stable regions, developers often talk about **diphones** - parts of phones between two consecutive phones.

Next, phones build subword units, like syllables. Sometimes, syllables are defined as "reduction-stable entities". For instance, when speech becomes fast, phones often change, but syllables remain the same. Also, syllables are related to an intonational contour. There are other ways to build subwords - morphologically-based (in morphology-rich languages) or phonetically-based. Subwords are often used in open vocabulary speech recognition.

Subwords form words. Words are important in speech recognition because they restrict combinations of phones significantly. If there are 40 phones and an average word has 7 phones, there must be 40^7 words. Luckily, even people with a rich vocabulary rarely use more than 20k words in practice, which makes recognition way more feasible.

Words and other non-linguistic sounds, which we call **fillers** (breath, um, uh, cough), form **utterances**. They are separate chunks of audio between pauses. They don't necessarily match sentences, which are more semantic concepts.

Recognition Process

The common way to recognize speech is the following: we take a waveform, split it at utterances by silences and then try to recognize what's being said in each utterance. To do that, we want to take all possible combinations of words and try to match them with the audio. We choose the best matching combination.

There are some important concepts in this matching process. First of all it's the concept of **features**. Since the number of parameters is large, we are trying to optimize it. Numbers that are calculated from speech usually by dividing the speech into frames. Then for each frame, typically of 10 milliseconds length, we extract 39 numbers that represent the speech. That's called a **feature vector**. The way to generate the number of parameters is a subject of active investigation, but in a simple case it's a derivative from the spectrum.

Second, it's the concept of the **model**. A model describes some mathematical object that gathers common attributes of the spoken word.

Models

According to the speech structure, three models are used in speech recognition to do the match.

An **acoustic model** contains acoustic properties for each sen one. There are context-independent models that contain properties (the most probable feature vectors for each phone) and context-dependent ones (built from senones with context).

A **phonetic dictionary** contains a mapping from words to phones. This mapping is not very effective. For example, only two to three pronunciation variants are noted in it. However, it's practical enough most of the time. The dictionary is not the only method for mapping words to phones. You could also use some complex function learned with a machine learning algorithm.

A **language model** is used to restrict word search. It defines which word could follow previously recognized words (remember that matching is a sequential process) and helps to significantly restrict the matching process by stripping words that are not probable. The most common language models are n-gram language models—these contain statistics of word sequences—and finite state language models—these define speech sequences by finite state automation, sometimes with weights

Other used Concepts

A **Lattice** is a directed graph that represents variants of the recognition. Often, getting the best match is not practical. In that case, lattices are good intermediate formats to represent the recognition result.

N-best lists of variants are like lattices, though their representations are not as dense as the lattice ones.

Word confusion networks (sausages) are lattices where the strict order of nodes is taken from lattice edges.

Speech database - a set of typical recordings from the task database. If we develop dialog system it might be dialogs recorded from users. For dictation system it might be reading recordings. Speech databases are used to train, tune and test the decoding systems.

Text databases - sample texts collected for e.g. language model training. Usually, databases of texts are collected in sample text form. The issue with such a collection is to put present documents (like PDFs, web pages, scans) into a spoken text form. That is, you need to remove tags and headings, to expand numbers to their spoken form and to expand abbreviations.

4.2.6 Pattern Mining

Q23. What is pattern mining? Explain about it.

Ans :

Pattern mining consists of using/developing **data mining algorithms** to discover interesting, unexpected and useful patterns in databases.

Pattern mining algorithms can be applied on **various types of data** such as transaction databases, sequence databases, streams, strings, spatial data, graphs, etc.

Pattern mining algorithms can be designed to discover **various types of patterns**: subgraphs, associations, indirect associations, trends, periodic patterns, sequential rules, lattices, sequential patterns, high-utility patterns, etc.

Examples

1. Discovering Frequent Itemsets

The most popular algorithm for pattern mining is without a doubt **Apriori** (1993). It is designed to be applied on a transaction database to discover patterns in transactions made by customers in stores. But it can also be applied in several other applications. A transaction is defined a set of distinct items (symbols). **Apriori** takes as input (1) a *minsupthres* hold set by the user and (2) a transaction database containing a set of transactions. **Apriori** outputs all **frequent itemsets**, i.e. groups of items shared by no less than *minsuptrans* actions in the input database. For example, consider the following transaction database containing four transactions. Given a *minsup* of two transactions, frequent itemsets are “bread, butter”, “bread milk”, “bread”, “milk” and “butter”.

T1: bread, butter, spinach

T2: butter, salmon

T3: bread, milk, butter

T4: cereal, bread milk

Fig. : A Transaction Database

2. Discovering Sequential Rules

The second example that I will give is to **discover sequential rules** in a **sequence database**. A sequence database is defined as a set of sequences. A sequence is a list of transactions (as previously defined). For example in the left part of the following figure a sequence database containing four sequences is shown. The first sequence contains item *a* and *b* followed by *c*, followed

by *f*, followed by *g*, followed by *e*. A **sequential rule** has the form $X \rightarrow Y$ where *X* and *Y* are two distinct non empty sets of items. The meaning of a rule is that if the items of *X* appears in a sequence in any order, they will be followed by the items of *Y* in any order. The **support of a rule** is the number of sequence containing the rule divided by the total number of sequences. The **confidence of a rule** is the number of sequence containing the rule divided by the number of sequences containing its antecedent. The goal of **sequential rule mining** is to discover all **sequential rules** having a support and confidence no less than two thresholds given by the user named “minsup” and “minconf”. For example, on the right part of the following figure some sequential rules are shown for minsup = 0.5 and minconf = 0.5, discovered by the **Rule Growth algorithm**.

4.3 GAME PLAYING

4.3.1 Important Concepts of Game Theory

Q24. What is game theory? Explain the basic concepts in game theory.

Ans :

Game theory is the process of modeling the strategic interaction between two or more players in a situation containing set rules and outcomes.

Game theory attempts to reduce these transaction to mathematical terms, to deduce which actions, by the parties, are better or worse, based on the desired outcome.

Game Theory Definitions

1. **Game:** Any set of circumstances that has a result dependent on the actions of two or more decision-makers (players)
2. **Players :** A strategic decision-maker within the context of the game
3. **Strategy :** A complete plan of action a player will take given the set of circumstances that might arise within the game
4. **Payoff :** The payout a player receives from arriving at a particular outcome. The payout can be in any quantifiable form, from dollars to utility.

5. **Information set** : The information available at a given point in the game. The term information set is most usually applied when the game has a sequential component.
6. **Equilibrium** : The point in a game where both players have made their decisions and an outcome is reached.
-

4.3.2 Game Playing and Knowledge Structure

Q25. Explain the role of knowledge representation in game playing.

Ans :

Knowledge representation is the task of understanding how the world around us works and how difficult that will make it for a rational agent to act within it.

Part 1: Select a Game

We can begin by select an existing game that you are familiar with. This can be a board game, card game or video game. Write down key information such as:

- A quick summary of what the game is about: this can be expressed by themes, narratives etc.
- How many players can participate?
- How players participate in the game?
- What are the rules of the game?
- What conditions allow for a player to win that game?

Part 2: Understand Your Game

Now that we have selected a game and identified what it is and how it works. Let's begin assessing it for the purposes of an AI agent. Take your game and assess the following criteria:

Determinism

If we take a particular action in a game, do we always know the outcome? This is complicated by game elements such as dice, given that – while they only have one of six outcomes – we cannot predict with 100% confidence what the dice roll would be.

Observability

Can we see everything that is happening in the game? Is there any information that is hidden from us? This will have an impact on our decisions given we can only make educated guesses as to what is currently happening. Card games are often guilty of this, given that players will each be holding cards that only they can see. Even something as simple as *Snap* is partially-observable because while we can guess what cards are still to be revealed, we do not know who is actually holding them.

Single/Multi-Agent

Are there any other agents that can have an impact on our performance? Is it solely the player who is in control, or are there other agents in play who can instigate change in the world?

Discrete

Is the game discretised? Or is it a continuous event? Of course if it is a video game then on some level it is discretised given that it will run on a fixed number of frames per second. However, consider the reality of the problem rather than the methods of its execution through programming.

4.3.3 Game As Search Problem

Q26. Explain about game as search problem.

Ans :

The behaviour / actions of the opponent are unpredictable, therefore search for a “worst-case”-plan.

Time limit, therefore complete search is not feasible and an approximation is needed

- Algorithm for perfect play
- Finite horizon, approximate evaluation
- Pruning search tree

Games vs Search Problems

- “Unpredictable” opponent : specifying a move for every possible opponent reply.
- Time limits : unlikely to find goal, must approximate

Game Playing Strategy

- Maximize winning possibility assuming that opponent will try to minimize (Minimax Algorithm)
- Ignore the unwanted portion of the search tree (Alpha Beta Pruning)
- Evaluation(Utility) Function

A measure of winning possibility of the player

Tic-Tac-Toe

X	O		

$$e(p) = 6 - 5 = 1$$

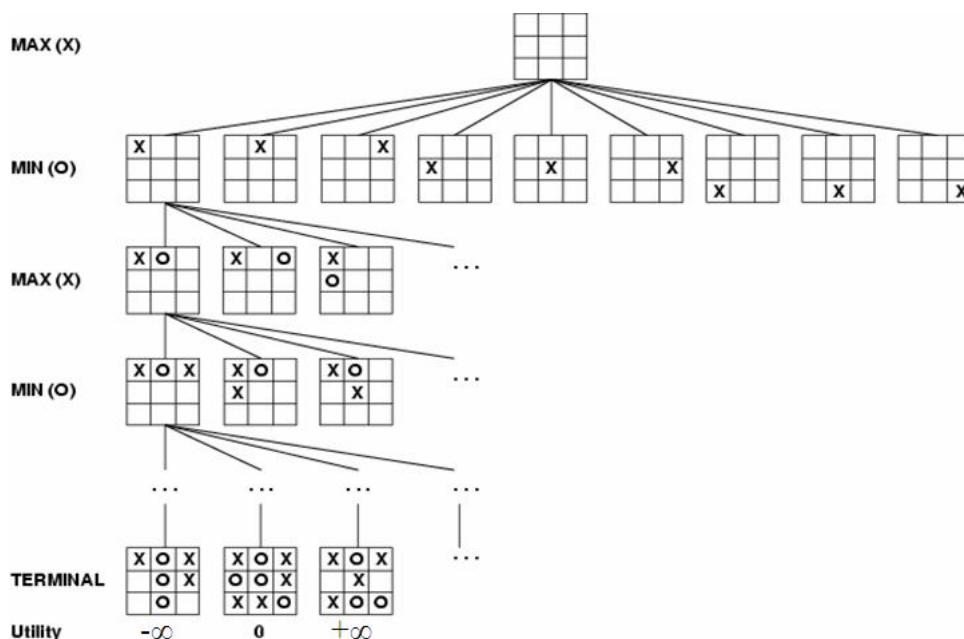
- Initial State: Board position of 3x3 matrix with 0 and X.
- Operators: Putting 0's or X's in vacant positions alternatively
- Terminal test: Which determines game is over
- Utility function:

$$e(p) = (\text{No. of complete rows, columns or diagonals are still open for player}) - (\text{No. of complete rows, columns or diagonals are still open for opponent})$$

Minimax Algorithm

- Generate the game tree
- Apply the utility function to each terminal state to get its value
- Use these values to determine the utility of the nodes one level higher up in the search tree
 - From bottom to top
 - For a max level, select the maximum value of its successors.
 - For a min level, select the minimum value of its successors.
 - From root node select the move which leads to highest value.

Game tree for Tic-Tac-Toe



4.3.4 Alpha-beta Pruning

Q27. Explain about ALPHA-BETA pruning

Ans :

ALPHA-BETA pruning is a method that reduces the number of nodes explored in Minimax strategy. It reduces the time required for the search and it must be restricted so that no time is to be wasted searching moves that are obviously bad for the current player.

The exact implementation of alpha-beta keeps track of the best move for each side as it moves throughout the tree. We proceed in the same (preorder) way as for the minimax algorithm.

For the MIN nodes, the score computed starts with $+\infty$ and decreases with time.

For MAX nodes, scores computed starts with $-\infty$ and increase with time.

The efficiency of the Alpha-Beta procedure depends on the order in which successors of a node are examined. If we were lucky, at a MIN node we would always consider the nodes in order from low to high score and at a MAX node the nodes in order from high to low score. In general it can be shown that in the most favorable circumstances the alpha-beta search opens as many leaves as minimax on a game tree with double its depth.

Here is an example of Alpha-Beta search

Alpha-Beta algorithm. The algorithm maintains two values, alpha and beta, which represent the minimum score that the maximizing player is assured of and the maximum score that the minimizing player is assured of respectively. Initially alpha is negative infinity and beta is positive infinity. As the recursion progresses the "window" becomes smaller.

When beta becomes less than alpha, it means that the current position cannot be the result of best play by both players and hence need not be explored further.

Pseudocode for the alpha-beta algorithm is given below.

evaluate (node, alpha, beta)

```

if node is a leaf
return the heuristic value of node
if node is a minimizing node
for each child of node
    beta = min (beta, evaluate (child, alpha, beta))
    if beta <= alpha
        return beta
return beta
if node is a maximizing node
for each child of node
    alpha = max (alpha, evaluate (child, alpha, beta))
    if beta <= alpha
        return alpha
return alpha

```

4.3.5 Game Theory Problems

Q28. Explain Prisoner's Dilemma Problem.

Ans :

Prisoner's Dilemma Problem

The prisoner's dilemma refers to a situation, wherein an individual has to choose between self-interest and mutual interest. The **prisoner's dilemma** is a standard example of a game analyzed in game theory that shows why two completely "rational" individuals might not cooperate, even if it appears that it is in their best interests to do so.

Further examples will make it clear why above stated statement is TRUE in most of the cases.

Two men are arrested by the police on suspicion of committing the same crime. They are questioned by the police in separate rooms. To convict them, the police need testimony from at least one of them. Both are rational, and value their personal freedom more than the other's. They have two options to confess or remain silent. If one confesses and the other remains silent, he (who remained silent) will have to serve the full tenure of punishment. On the other hand, if both confess and accuse the other to be a culprit, they'll share the sentence of imprisonment, that will be lesser than the full term. However, if both remain silent, due to a lack of evidence, the police will have to sentence both to a much lesser period.

The offer is

1. If A and B each betray the other, each of them serves 2 years in prison.
2. If A betrays B but B remains silent, A will be set free and B will serve 3 years in prison (and vice versa).
3. If A and B both remain silent, both of them will only serve 1 year in prison (on the lesser charge).

	Prisoner B stays silent (cooperates)	Prisoner B betrays (defects)
Prisoner A stays silent (Cooperates)	Each serves 1 year	Prisoner A : 3 years Prisoner B : goes free
Prisoner A betrays (defects)	Prisoner A: goes free Prisoner B : 3 years	Each serves 2 years

Thus, the best option for suspects is to remain silent and not testify against the other. However, neither of them know what the other will say, and lack of trust and confidence in other accomplice may compel one of them to testify rather than remain silent. They're faced with this dilemma, since there is a risk of the other partner testifying against if the other one remains mum. If they have mutual trust, it will be easy for them to have a win-win situation by staying mum.

Q29. Explain Rock-Paper- Scissors Problem.

Ans :

Rock-Paper- Scissors Problem

This rock-paper-scissors game illustrates the basic principles of an adaptive artificial intelligence technology.

General Statement

A tournament is being held for champion players of the game Rock, Paper, and Scissors. For Player A and Player B, determine who wins each game and who wins the overall tournament.

Input

The first line in the data set is an integer that represents the number of datapairs that follow. The data begins on the second line. R represents rock, P represents paper, and S represents scissors.

Output

The output is to be formatted exactly like that for the sample output given below.

Assumptions

The only letters in the input will be upper case R, P, and S. The first letter in the pair is the choice for player A and the second letter is the choice for player B.

Example

Input: 7

RRRSSRSPPPSRP

etc.

Output:

DRAW

A WINS

B WINS

A WINS

DRAW

B WINS

B WINS

B WINS TOURNAMENT

etc.

Discussion

The game is a draw if both players choose the same item. Paper wins over rock because paper covers rock. Scissors wins over paper because scissors cuts paper. Rock wins over scissors because rock breaks scissors.

How Does it Work?

The computer keeps track of the conditional probabilities of you picking each of the three objects given the object you picked last. The computer always picks the object that beats the one that it thinks you are most likely to choose. Although it knows what you have actually picked (you press one of the buttons to make your choice), it is honest and doesn't cheat!

You can observe the computer learning by picking a strategy and sticking to it for a while. Here are a few things to try:

1. Pick Rock, then Paper, then Scissors, then Rock again and keep that pattern up. See how quickly the computer learns to beat you every time?
2. Having done that a few times, change strategy and pick Paper 5 times in a row. See how the computer spots your change of strategy and alters its play?
3. Pick any strategy of your own and see if the computer can spot the pattern.
4. See if you can be perfectly random in your choices and beat the computer.

4.3.6 Robotics

Q30. Write the brief introduction of robotics.

Ans :

A **ROBOT** is a mechanical or virtual artificial agent, usually an electro-mechanical machine that is guided by a computer program or electronic circuitry.

Robots can be autonomous or semi-autonomous. A Robot may convey a sense of intelligence or thought of its own.

A **Mechatronic Device** is a degenerate robot with these components:

1. Sensors, which detect the state of the environment
2. Actuators, which modify the state of the environment
3. A Control System, which controls the actuators based on the environment as depicted by the sensors

A **Robot** is a mechatronic device which also includes **resourcefulness** or **autonomy**. A device with **autonomy** does its thing "on its own" without a human directly guiding it moment-by-moment. Some authors would contend that all mechatronic devices are robots, and that this book's restriction on **robot** entails only specialized software.

Various types of robots are usually classified by their capabilities. Two examples will be used to capture most of what we see as a "robot".

1. Machine Pet

A machine, capable of moving in some way, that can sense its surroundings and can act on what it senses autonomously. Most of these robots have no real useful purpose, other than to entertain and challenge. These are also commonly used for experimenting with sensors, artificial intelligence, actuators and more. Most of this book covers this type of robot.

2. Autonomous Machine

A machine with sensors and actuators that can do some sort of work "on its own". This

includes things like robotic lawnmowers and vacuum cleaners, and also self-operating construction machines such as CNC cutters. Most industrial and commercial robots fall in this category.

Applications of Robotics

1. Outer Space Applications

Robots are playing a very important role for outer space exploration. The robotic unmanned spacecraft is used as the key of exploring the stars, planets...etc.

2. Military Applications

In today's modern army robotics is an important factor which is researched and developed day by day. Already remarkable success has been achieved with unmanned aerial vehicles like the Predator drone, which are capable of taking surveillance photographs, and even accurately launching missiles at ground targets, without a pilot. There are many advantages in robotic technology in warfare however, as outlined by Major Kenneth Rose of the US Army's Training and Doctrine Command: "Machines don't get tired. They don't close their eyes. They don't hide under trees when it rains and they don't talk to their buddies. A human's attention to detail on guard duty drops dramatically in the first 30 minutes ... Machines know no fear."

3. Intelligent Home Applications

We can monitor home security, environmental conditions and energy usage with intelligent robotic home systems. Door and windows can be opened automatically and appliances such as lighting and air conditioning can be pre programmed to activate. This assists occupants irrespective of their state of mobility.

Industry

From the beginning of the industrial revolution robotics and automation becomes the most important part of manufacturing. Robotic arms which are able to perform multiple tasks such as welding, cutting, lifting, sorting and bending are used in fabrics.

The most commonly used configurations of the industrial robots are

Articulated Robots: An articulated robot is one which uses rotary joints to access its work space. Articulated robots can range from simple two-jointed structures to systems with 10 or more interacting joints. The six-axis, articulated robot is the most versatile industrial robot which allows for a high level of freedom.

Cylindrical Coordinate Robots: These robots have three degrees of freedom and they move linearly only along the Y and Z axes with a cylindrical work envelope.

Scara Robots: It stands for Selective Compliant Assembly Robot Arm or Selective Compliant Articulated Robot Arm. SCARA robots usually have four axes as any X-Y-Z coordinate within their work envelope and a fourth axis of motion which is the wrist rotate (Theta-Z).

Spherical Coordinate Robots: The special arm, also known as polar coordinate robot arm, has one sliding motion and two rotational, around the vertical post and around a shoulder joint.

4.4 CONCEPTS AND TERMINOLOGY OF ANN

Q31. What is Artificial Neural Network?

Ans :

Artificial Neural Network (ANN) is an efficient computing system whose central theme is borrowed from the analogy of biological neural networks. ANNs are also named as "artificial neural systems," or "parallel distributed processing systems," or "connectionist systems." ANN acquires a large collection of units that are interconnected in some pattern to allow communication between the units. These units, also referred to as nodes or neurons, are simple processors which operate in parallel.

Every neuron is connected with other neuron through a connection link. Each connection link is

associated with a weight that has information about the input signal. This is the most useful information for neurons to solve a particular problem because the weight usually excites or inhibits the signal that is being communicated. Each neuron has an internal state, which is called an activation signal. Output signals, which are produced after combining the input signals and activation rule, may be sent to other units.

4.4.1 FEED-FORWARD NN

Q32. Explain about Feed Forward Neural Networks.

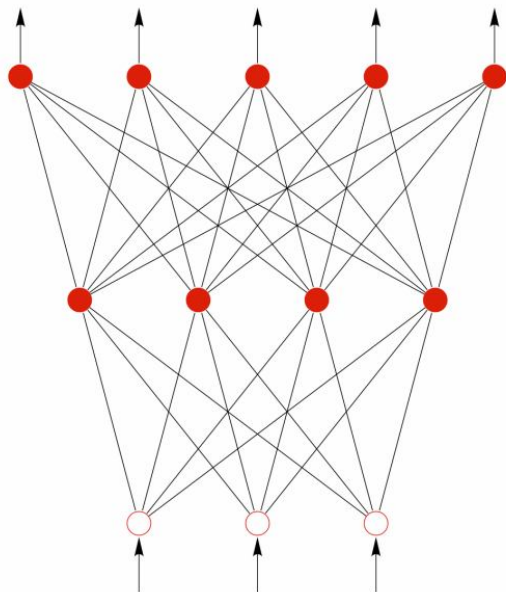
Ans :

Deep feedforward networks, also often called **feedforward neural networks**, or **multilayer perceptrons (MLPs)**, are the quintessential deep learning models. The goal of a feedforward network is to approximate some function f^* . For example, for a classifier, $y = f^*(\mathbf{x})$ maps an input \mathbf{x} to a category \mathbf{y} . A feedforward network defines a mapping $\mathbf{y} = f(\mathbf{x}; \boldsymbol{\theta})$ and learns the value of the parameters $\boldsymbol{\theta}$ that result in the best function approximation. (Reference)

These models are called feedforward because information flows through the function being evaluated from \mathbf{x} , through the intermediate computations used to define f , and finally to the output \mathbf{y} . There are no feedback connections in which outputs of the model are fed back into itself. When feedforward neural networks are extended to include feedback connections, they are called **recurrent neural networks** (we will see in later segment).

As we know the inspiration behind neural networks are our brains. So let's see the biological aspect of neural networks.

In the following figure we see an example of a 2-layered network with, from top to bottom: an output layer with 5 units, a *hidden* layer with 4 units, respectively. The network has 3 input units.



The 3 inputs are shown as circles and these do not belong to any layer of the network (although the inputs sometimes are considered as a virtual layer with layer number 0). Any layer that is not an output layer is a *hidden* layer. This network therefore has 1 hidden layer and 1 output layer. The figure also shows all the connections between the units in different layers. A layer only connects to the previous layer.

The operation of this network can be divided into two phases:

1. The Learning Phase

The FFNet uses a *supervised* learning algorithm: besides the input pattern, the neural net also needs to know to what category the pattern belongs. Learning proceeds as follows: a pattern is presented at the inputs. The pattern will be transformed in its passage through the layers of the network until it reaches the output layer. The units in the output layer all belong to a different category. The outputs of the network as they are now are compared with the outputs as they ideally would have been if this pattern were correctly classified: in the latter case the unit with the correct category would have had the largest output value and

the output values of the other output units would have been very small.

2. The Classification Phase

In the classification phase, the weights of the network are fixed.

A pattern, presented at the inputs, will be transformed from layer to layer until it reaches the output layer. Now classification can occur by selecting the category associated with the output unit that has the largest output value. For classification we only need to select an FFNet and a Pattern List together and choose To Categories.

In contrast to the learning phase classification is very fast.

4.4.2 Feedback Networks

Q33. Write about feed back networks

Ans :

Feedback (or recurrent or interactive) networks can have signals traveling in both directions by introducing loops in the network. Feedback networks are powerful and can get extremely complicated. Computations derived from earlier input are fed back into the network, which gives them a kind of memory. Feedback networks are dynamic; their 'state' is changing continuously until they reach an equilibrium point. They remain at the equilibrium point until the input changes and a new equilibrium needs to be found.

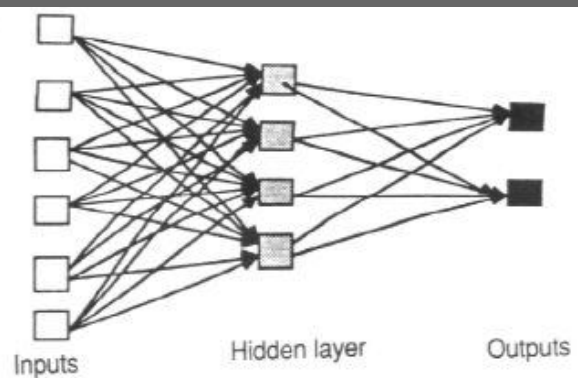


Fig.: Example of Simple Feed back Networks

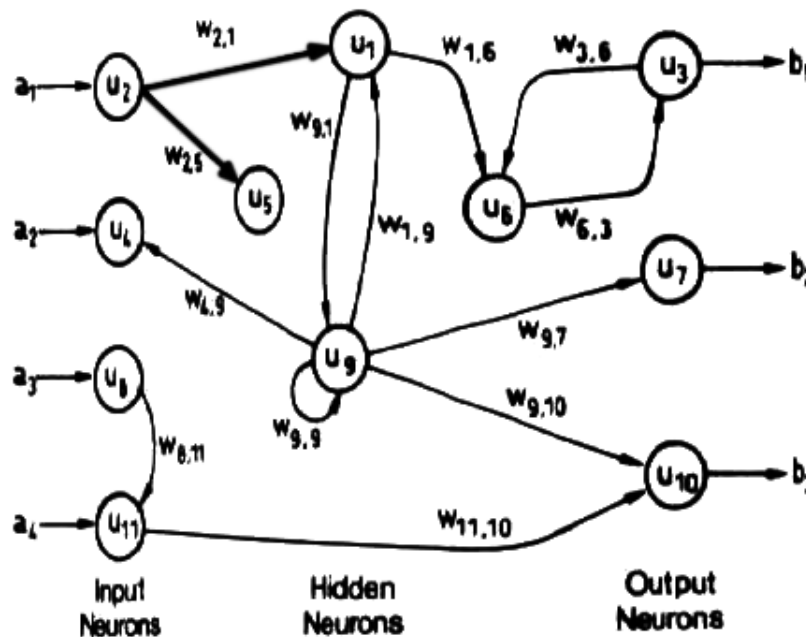


Fig: Complicated of Feedback Networks

As an example of feedback network, is 1 Hopfield's network. The main use of Hopfield's network is as associative memory. An associative memory is a device which accepts an input pattern and generates an output as the stored pattern which is most closely associated with the input. The function of the associate memory is to recall the corresponding stored pattern, and then produce a clear version of the pattern at the output. Hopfield networks are typically used for those problems with binary pattern vectors and the input pattern may be a noisy version of one of the stored patterns. In the Hopfield network, the stored patterns are encoded as the weights of the network.

4.4.3 Pattern Associative Networks

Q34. Explain different types of associative networks.

Ans :

These kinds of neural networks work on the basis of pattern association, which means they can store different patterns and at the time of giving an output they can produce one of the stored patterns by matching them with the given input pattern. These types of memories are also called **Content-Addressable Memory (CAM)**. Associative memory makes a parallel search with the stored patterns as data files.

Following are the two types of associative memories we can observe.

1. Auto Associative Memory
2. Hetero Associative memory

1. Auto Associative Memory

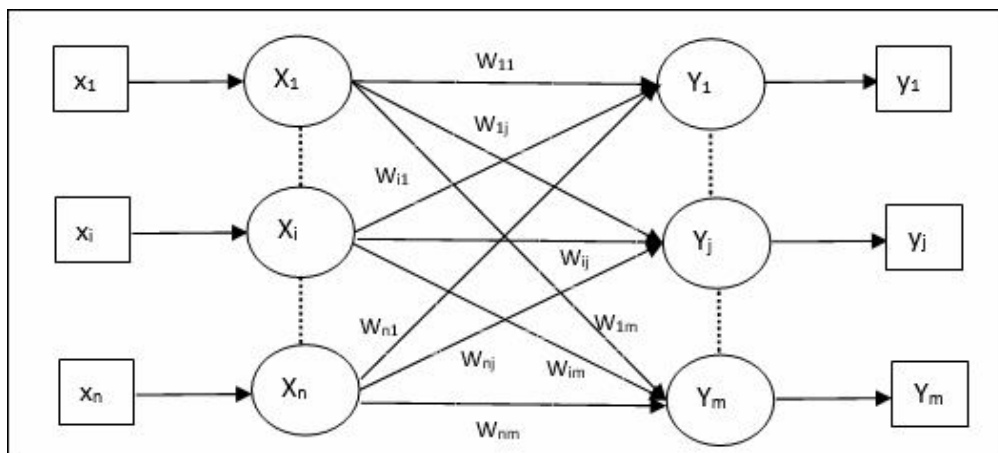
This is a single layer neural network in which the input training vector and the output target vectors are the same. The weights are determined so that the network stores a set of patterns.

Hetero Associative Memory

Similar to Auto Associative Memory network, this is also a single layer neural network. However, in this network the input training vector and the output target vectors are not the same. The weights are determined so that the network stores a set of patterns. Hetero associative network is static in nature, hence, there would be no non-linear and delay operations.

Architecture

As shown in the following figure, the architecture of Hetero Associative Memory network has ' n ' number of input training vectors and ' m ' number of output target vectors.



Training Algorithm

For training, this network is using the Hebb or Delta learning rule.

Step 1 : Initialize all the weights to zero as $w_{ij} = 0$ ($i = 1$ to n , $j = 1$ to m)

Step 2 : Perform steps 3-4 for each input vector.

Step 3 : Activate each input unit as follows.

$$x_i = s_i(i = 1 \text{ to } n)$$

Step 4 : Activate each output unit as follows.

$$y_j = s_j(j = 1 \text{ to } m)$$

Step 5 : Adjust the weights as follows.

$$w_{ij}(\text{new}) = w_{ij}(\text{old}) + x_i y_j$$

Testing Algorithm

Step 1 : Set the weights obtained during training for Hebb's rule.

Step 2 : Perform steps 3-5 for each input vector.

Step 3 : Set the activation of the input units equal to that of the input vector.

Step 4 : Calculate the net input to each output unit $j = 1$ to m ;

$$y_{inj} = \sum_{i=1}^n x_i w_{ij}$$

Step 5 : Apply the following activation function to calculate the output.

$$y_j = f(y_{inj}) = \begin{cases} +0 & \text{if } y_{inj} > 0 \\ 0 & \text{if } y_{inj} = 0 \\ -1 & \text{if } y_{inj} < 0 \end{cases}$$

Q35. Explain about Hopfield Network.

Ans :

Hopfield neural network was invented by Dr. John J. Hopfield in 1982. It consists of a single layer which contains one or more fully connected recurrent neurons. The Hopfield network is commonly used for auto-association and optimization tasks.

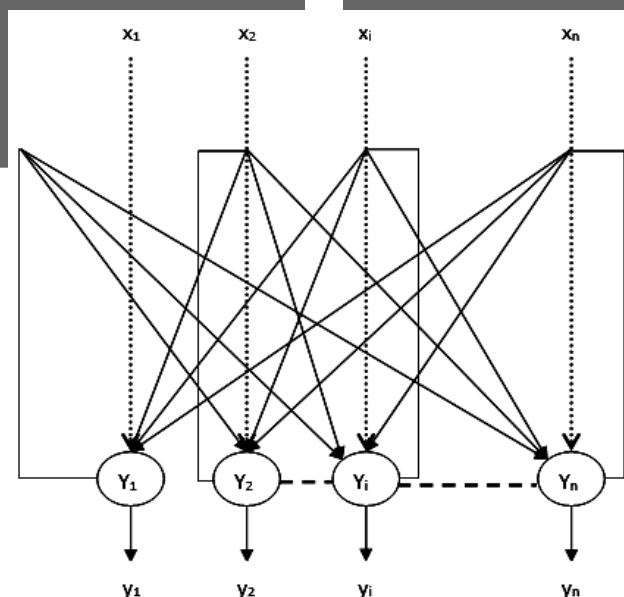
Discrete Hopfield Network

A Hopfield network which operates in a discrete line fashion or in other words, it can be said the input and output patterns are discrete vector, which can be either binary (0,1) or bipolar (+1, -1) in nature. The network has symmetrical weights with no self-connections i.e., $w_{ij} = w_{ji}$ and $w_{ii} = 0$.

Architecture

Following are some important points to keep in mind about discrete Hopfield network

- This model consists of neurons with one inverting and one non-inverting output.
- The output of each neuron should be the input of other neurons but not the input of self.
- Weight/connection strength is represented by w_{ij} .
- Connections can be excitatory as well as inhibitory. It would be excitatory, if the output of the neuron is same as the input, otherwise inhibitory.
- Weights should be symmetrical, i.e. $w_{ij} = w_{ji}$



The output from Y_1 going to Y_2 , Y_i and Y_n have the weights w_{12} , w_{1i} and w_{1n} respectively. Similarly, other arcs have the weights on them.

Training Algorithm

During training of discrete Hopfield network, weights will be updated. As we know that we can have the binary input vectors as well as bipolar input vectors. Hence, in both the cases, weight updates can be done with the following relation

Case 1 : Binary input patterns

For a set of binary patterns $s(p)$, $p = 1$ to P

Here, $s(p) = s_1(p), s_2(p), \dots, s_i(p), \dots, s_n(p)$

Weight Matrix is given by

$$\sum_{p=1}^P [2S_i(p) - 1][2s_j(p) - 1] \text{ for } i \neq j$$

Case 2 : Bipolar input patterns

For a set of binary patterns $s(p)$, $p = 1$ to P

Here, $s(p) = s_1(p), s_2(p), \dots, s_i(p), \dots, s_n(p)$

Weight Matrix is given by

$$w_{ij} = \sum_{p=1}^P [s_i(p)][s_j(p)] \text{ for } i \neq j$$

Testing Algorithm

Step 1 : Initialize the weights, which are obtained from training algorithm by using Hebbian principle.

Step 2 : Perform steps 3-9, if the activations of the network is not consolidated.

Step 3 : For each input vector X , perform steps 4-8.

Step 4 : Make initial activation of the network equal to the external input vector X as follows

$$y_i = x_i \text{ for } i = 1 \text{ to } n$$

Step 5 : For each unit Y_i , perform steps 6-9.

Step 6 : Calculate the net input of the network as follows.

$$y_{ini} = x_i + \sum_j y_j w_{ji}$$

Step 7 : Apply the activation as follows over the net input to calculate the output.

$$y_i = \begin{cases} 1 & \text{if } y_{inj} > \theta_i \\ y_i & \text{if } y_{inj} = \theta_i \\ 0 & \text{if } y_{inj} < \theta_i \end{cases}$$

Here θ_i is the threshold.

Step 8 : Broadcast this output y_i to all other units.

Step 9 : Test the network for conjunction.

Energy Function Evaluation

An energy function is defined as a function that is bounded and non-increasing function of the state of the system.

Energy function E_f , also called **Lyapunov function** determines the stability of discrete Hopfield network, and is characterized as follows.

$$E_f = -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n y_i y_j w_{ij} - \sum_{i=1}^n x_i y_i + \sum_{i=1}^n \theta_i y_i$$

Condition

In a stable network, whenever the state of node changes, the above energy function will decrease.

Suppose when node i has changed state from $y_i^{(k)}$ to $y_i^{(k+1)}$ then the Energy change ΔE_f is given by the following relation

$$\begin{aligned} \Delta E_f &= E_f(y_i^{(k+1)}) - E_f(y_i^{(k)}) \\ &= - \left(\sum_{j=1}^n w_{ij} y_j^{(k)} + x_i - \theta_i \right) (y_i^{(k+1)} - y_i^{(k)}) \\ &= - (\text{net}_i) \Delta y_i \end{aligned}$$

Here $\Delta y_i = y_i^{(k+1)} - y_i^{(k)}$

The change in energy depends on the fact that only one unit can update its activation at a time.

Continuous Hopfield Network

In comparison with Discrete Hopfield network, continuous network has time as a continuous variable. It is also used in auto association and optimization problems such as travelling salesman problem.

Model The model or architecture can be build up by adding electrical components such as amplifiers which can map the input voltage to the output voltage over a sigmoid activation function.

Energy Function Evaluation

$$E_f = \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n y_i y_j w_{ij} - \sum_{i=1}^n x_i y_i + \frac{1}{\lambda} \sum_{i=1}^n \sum_{j=1}^n w_{ij} g_{ri} \int_0^{y_i} a^{-1}(y) dy$$

Here λ is gain parameter and g_{ri} input conductance.

4.4.4 Competitive Learning

Q36. What is competitive learning? Explain it.

Ans :

Competitive learning is a form of unsupervised learning in artificial Neural Networks. The nodes compete for the right to respond to a subset of the input data. Competitive learning works by increasing the specialization of each node in the network. It is well suited to finding clusters within data.

Examples Include: Vector quantization and Kohonen maps (self-organizing maps)

Principles of Competitive Learning

There are three basic elements to a competitive learning rule:

1. A set of neurons that are all the same, except for some randomly distributed synaptic weights, which respond differently to a given set of input patterns
2. A limit which is imposed on the "strength" of each neuron.
3. A mechanism that permits the neurons to compete for the right to respond to a given subset of inputs, such that only one output neuron (or only one neuron per group), is active (i.e. "on") at a time. The neuron that wins the competition is called a "winner-take-all" neuron.

Individual neurons of the network learn to specialize on ensembles of similar patterns and become 'feature detectors' for different classes of input patterns.

The competitive networks recode sets of correlated inputs to one of a few output neurons essentially removes the redundancy in representation.

Architecture & Implementation:

Competitive Learning is usually implemented with Neural Networks that contain a hidden layer which is commonly known as "competitive layer".

Every competitive neuron is described by a vector of weights :

$w_i = (w_{i1}, \dots, w_{id})^T, i = 1, \dots, M$ and calculates the similarity measure between the input data

$x^n = (x_{n1}, \dots, x_{nd})^T \in R^d$ and the weight vector w_i .

For every input vector, the competitive neurons "compete" with each other to see which one of them is the most similar to that particular input vector. The winner neuron m sets its output to:

$$O_m = 1$$

All the other competitive neurons set their output to:

$$O_i = 0, i = 1, \dots, M, i \neq m.$$

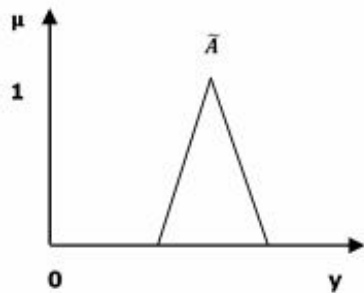
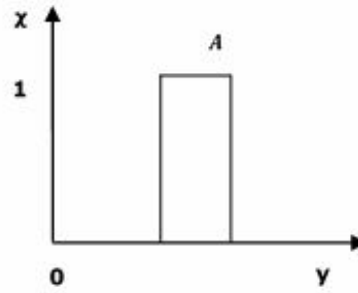
Usually, in order to measure similarity the inverse of the Euclidean distance is used.

4.4.5 Fuzzy Sets

Q37. What are Fuzzy Sets? Write about how to implement fuzzy sets.

Ans :

Fuzzy sets can be considered as an extension and gross oversimplification of classical sets. It can be best understood in the context of set membership. Basically it allows partial membership which means that it contain elements that have varying degrees of membership in the set. From this, we can understand the difference between classical set and fuzzy set. Classical set contains elements that satisfy precise properties of membership while fuzzy set contains elements that satisfy imprecise properties of membership.

Membership Function of Fuzzy set \tilde{A} 

Membership Function of classical set A

Mathematical Concept

A fuzzy set \tilde{A} in the universe of information U can be defined as a set of ordered pairs and it can be represented mathematically as

$$\tilde{A} = \{(y, \mu_{\tilde{A}}(y)) \mid y \in U\}$$

Here $\mu_{\tilde{A}}(y)$ = degree of membership of y in \tilde{A} , assumes values in the range from 0 to 1, i.e., $\mu_{\tilde{A}}(y) \in [0, 1]$.

Representation of fuzzy set

Let us now consider two cases of universe of information and understand how a fuzzy set can be represented.

Case 1

When universe of information U is discrete and finite

$$\begin{aligned}\tilde{A} &= \left\{ \frac{\mu_{\tilde{A}}(y_1)}{y_1} + \frac{\mu_{\tilde{A}}(y_2)}{y_2} + \frac{\mu_{\tilde{A}}(y_3)}{y_3} + \dots \right\} \\ &= \left\{ \sum_{i=1}^n \frac{\mu_{\tilde{A}}(y_i)}{y_i} \right\}\end{aligned}$$

Case 2

When universe of information U is continuous and infinite “

$$\tilde{A} = \left\{ \int \frac{\mu_{\tilde{A}}(y)}{y} \right\}$$

In the above representation, the summation symbol represents the collection of each element.

Operations on Fuzzy Sets

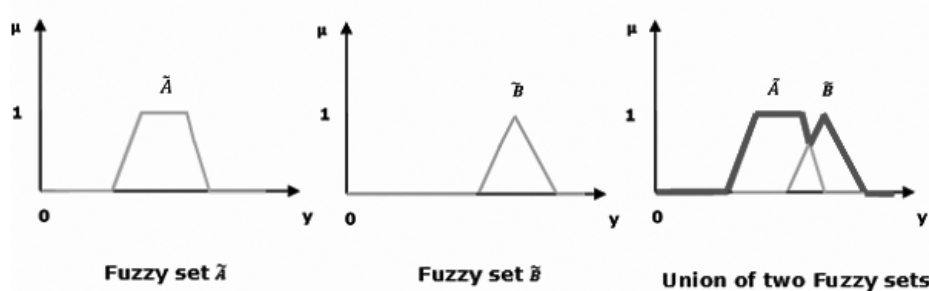
Having two fuzzy sets \tilde{A} and \tilde{B} , the universe of information U and an element y of the universe, the following relations express the union, intersection and complement operation on fuzzy sets.

Union/Fuzzy 'OR'

Let us consider the following representation to understand how the **Union/Fuzzy 'OR'** relation works.

$$\mu_{\bar{A} \cup \bar{B}}(y) = \mu_{\bar{A}} \vee \mu_{\bar{B}} \forall y \in U$$

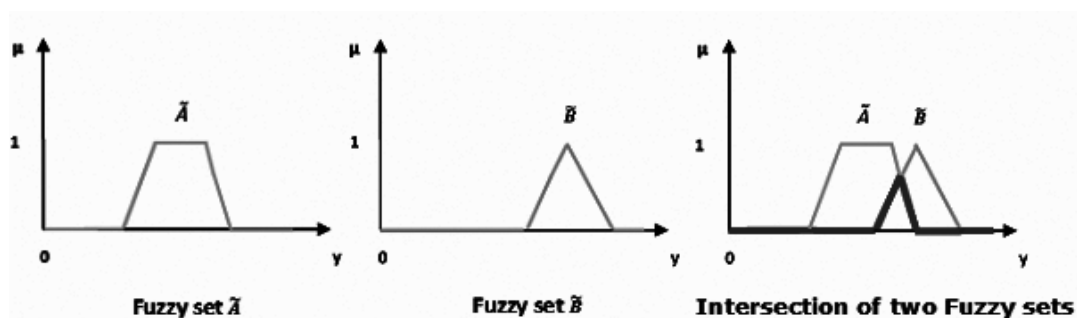
Here \vee represents the 'max' operation.

**Intersection/Fuzzy 'AND'**

Let us consider the following representation to understand how the **Intersection/Fuzzy 'AND'** relation works.

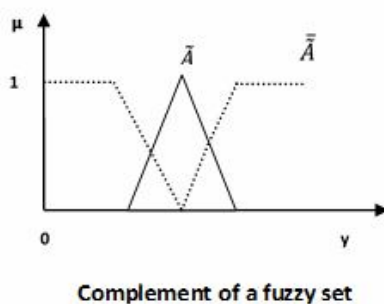
$$\mu_{\bar{A} \cap \bar{B}}(y) = \mu_{\bar{A}} \wedge \mu_{\bar{B}} \forall y \in U$$

Here \wedge represents the 'min' operation.

**Complement/Fuzzy 'NOT'**

Let us consider the following representation to understand how the **Complement/Fuzzy 'NOT'** relation works.

$$\mu_{\bar{A}} = 1 - \mu_A(y) \quad y \in U$$



Properties of Fuzzy Sets

Let us discuss the different properties of fuzzy sets.

Commutative Property

Having two fuzzy sets A^{\sim} and B^{\sim} , this property states.

$$\tilde{A} \cup \tilde{B} = \tilde{B} \cup \tilde{A}$$

$$\tilde{A} \cap \tilde{B} = \tilde{B} \cap \tilde{A}$$

Associative Property

Having three fuzzy sets A^{\sim} , B^{\sim} and C^{\sim} , this property states.

$$\tilde{A} \cup (\tilde{B} \cap \tilde{C}) = (\tilde{A} \cup \tilde{B}) \cap \tilde{C}$$

$$\tilde{A} \cap (\tilde{B} \cup \tilde{C}) = (\tilde{A} \cap \tilde{B}) \cup \tilde{C}$$

Distributive Property

Having three fuzzy sets A^{\sim} , B^{\sim} and C^{\sim} , this property states.

$$\tilde{A} \cup (\tilde{B} \cap \tilde{C}) = (\tilde{A} \cup \tilde{B}) \cap (\tilde{A} \cup \tilde{C})$$

$$\tilde{A} \cap (\tilde{B} \cup \tilde{C}) = (\tilde{A} \cap \tilde{B}) \cup (\tilde{A} \cap \tilde{C})$$

Idempotency Property

For any fuzzy set A^{\sim} , this property states

$$\tilde{A} \cup \tilde{A} = \tilde{A}$$

$$\tilde{A} \cap \tilde{A} = \tilde{A}$$

Idempotency Property

For fuzzy set A^{\sim} and universal set U , this property states.

$$\tilde{A} \cup \phi = \tilde{A}$$

$$\tilde{A} \cap U = \tilde{A}$$

$$\tilde{A} \cap \phi = \phi$$

$$\tilde{A} \cup U = U$$

Transitive Property

Having three fuzzy sets A^{\sim} , B^{\sim} and C^{\sim} , this property states

$$\text{If } \tilde{A} \subseteq \tilde{B} \subseteq \tilde{C}, \text{ then } \tilde{A} \subseteq \tilde{C}$$

Involution Property

For any fuzzy set A^{\sim} , this property states

$$\overline{\overline{\tilde{A}}} = \tilde{A}$$

De Morgan's Law

This law plays a crucial role in proving tautologies and contradiction. This law states.

$$\overline{\tilde{A} \cap \tilde{B}} = \overline{\tilde{A}} \cup \overline{\tilde{B}}$$

$$\overline{\tilde{A} \cup \tilde{B}} = \overline{\tilde{A}} \cap \overline{\tilde{B}}$$

4.4.6 Fuzzy Inference Process

Q38. Explain about Fuzzy Inference Process

Ans :

Fuzzy Inference System is the key unit of a fuzzy logic system having decision making as its primary work. It uses the "IF... THEN" rules along with connectors "OR" or "AND" for drawing essential decision rules.

Characteristics of Fuzzy Inference System

Following are some characteristics of FIS

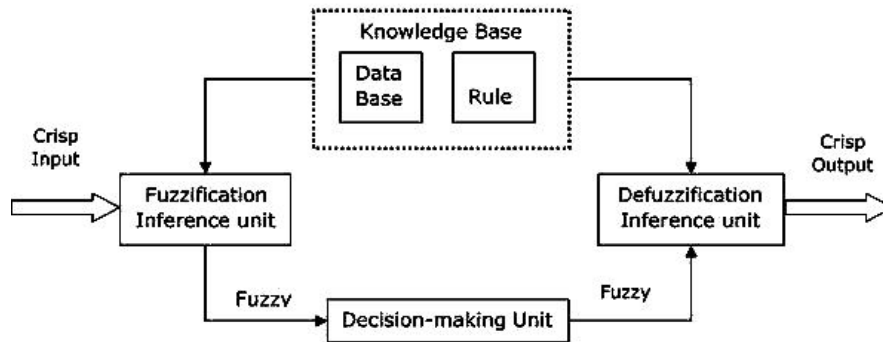
1. The output from FIS is always a fuzzy set irrespective of its input which can be fuzzy or crisp.
2. It is necessary to have fuzzy output when it is used as a controller.
3. A defuzzification unit would be there with FIS to convert fuzzy variables into crisp variables.

Functional Blocks of FIS

The following five functional blocks will help you understand the construction of FIS.

1. **Rule Base :** It contains fuzzy IF-THEN rules.
2. **Database :** It defines the membership functions of fuzzy sets used in fuzzy rules.
3. **Decision-making Unit :** It performs operation on rules.

4. **Fuzzification Interface Unit** : It converts the crisp quantities into fuzzy quantities.
5. **Defuzzification Interface Unit** : It converts the fuzzy quantities into crisp quantities. Following is a block diagram of fuzzy interference system.



Working of FIS

The working of the FIS consists of the following steps

1. A fuzzification unit supports the application of numerous fuzzification methods, and converts the crisp input into fuzzy input.
2. A knowledge base - collection of rule base and database is formed upon the conversion of crisp input into fuzzy input.
3. The defuzzification unit fuzzy input is finally converted into crisp output.

Methods of FIS

Let us now discuss the different methods of FIS. Following are the two important methods of FIS, having different consequent of fuzzy rules

1. Mamdani Fuzzy Inference System
2. Takagi-Sugeno Fuzzy Model (TS Method)

Mamdani Fuzzy Inference System

This system was proposed in 1975 by Ebhasim Mamdani. Basically, it was anticipated to control a steam engine and boiler combination by synthesizing a set of fuzzy rules obtained from people working on the system.

Steps for Computing the Output

Following steps need to be followed to compute the output from this FIS

Step 1 : Set of fuzzy rules need to be determined in this step.

Step 2 : In this step, by using input membership function, the input would be made fuzzy.

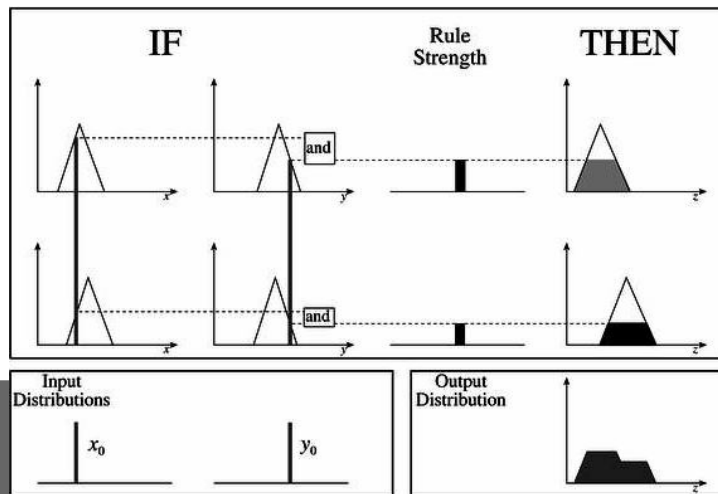
Step 3 : Now establish the rule strength by combining the fuzzified inputs according to fuzzy rules.

Step 4 : In this step, determine the consequent of rule by combining the rule strength and the output membership function.

Step 5 : For getting output distribution combine all the consequents.

Step 6 : Finally, a defuzzified output distribution is obtained.

Following is a block diagram of Mamdani Fuzzy Interface System.



4.4.7 Neuro-Fuzzy Systems

Q39. What are Neuro-Fuzzy Systems? Explain the layered approach of neuro fuzzy systems.

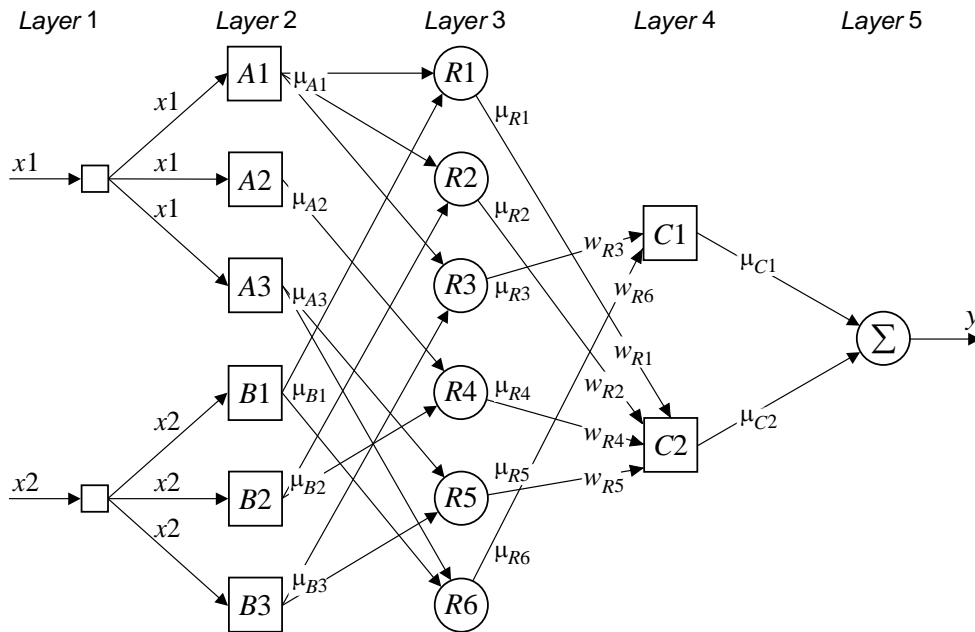
Ans :

- Fuzzy logic and neural networks are natural complementary tools in building intelligent systems.
- While neural networks are low-level computational structures that perform well when dealing with raw data, fuzzy logic deals with reasoning on a higher level, using linguistic information acquired from domain experts.
- However, fuzzy systems lack the ability to learn and cannot adjust themselves to a new environment.
- On the other hand, although neural networks can learn, they are opaque to the user.

Synergy of Neural and Fuzzy

- Integrated neuro-fuzzy systems can combine the parallel computation and learning abilities of neural networks with the human-like knowledge representation and explanation abilities of fuzzy systems.
- As a result, neural networks become more transparent, while fuzzy systems become capable of learning.
- A neuro-fuzzy system is a neural network which is functionally equivalent to a fuzzy inference model.
- Neuro-Fuzzy System can be trained to develop IF-THEN fuzzy rules and determine membership functions for input and output variables of the system.
- Expert knowledge can be incorporated into the structure of the neuro-fuzzy system.
- At the same time, the connection is structure avoids fuzzy inference, which entails a substantial computational burden.
- The structure of a neuro-fuzzy system is similar to a multi-layer neural network.
- In general, a neuro-fuzzy system has:
 - input and output layers,
 - and three hidden layers
 - that represent membership functions and fuzzy rules.

Layered approach of Neuro Fuzzy Systems



Each layer in the neuro-fuzzy system is associated with a particular step in the fuzzy inference process.

Layer 1 is the input layer.

Each neuron in this layer transmits external crisp signals directly to the next layer. That is,

$$y_i^{(1)} = x_i^{(1)}$$

Layer 2 is the fuzzification layer.

- Neurons in this layer represent fuzzy sets used in the antecedents of fuzzy rules.
- A fuzzification neuron receives a crisp input and determines the degree to which this input belongs to the neuron's fuzzy set.
- The activation function of a membership neuron is set to the function that specifies the neuron's fuzzy set.
- We use triangular sets, and therefore, the activation functions for the neurons in Layer 2 are set to the triangular membership functions.
- A triangular membership function can be specified by two parameters $\{a, b\}$ as follows:

$$y_i^{(2)} = \begin{cases} 0 & \text{if } x_i^{(2)} \leq a - \frac{b}{2} \\ 1 - \frac{2|x_i^{(2)} - a|}{b} & \text{if } a - \frac{b}{2} < x_i^{(2)} < a + \frac{b}{2} \\ 0 & \text{if } x_i^{(2)} \geq a + \frac{b}{2} \end{cases}$$

Layer 3 is the fuzzy rule layer.

1. Each neuron in this layer corresponds to a single fuzzy rule.
2. A fuzzy rule neuron receives inputs from the fuzzification neurons that represent fuzzy sets in the rule antecedents.
3. For instance, neuron $R1$, which corresponds to *Rule 1*, receives inputs from neurons $A1$ and $B1$.
4. In a neuro-fuzzy system, intersection can be implemented by the product operator.
5. Thus, the output of neuron i in *Layer 3* is obtained as:

$$y_i^{(3)} = x_{1i}^{(3)} \times x_{2i}^{(3)} \times \dots \times x_{ki}^{(3)}$$

$$y_{R1}^{(3)} = \mu_{A1} \times \mu_{B1} = \mu_{R1}$$

Layer 4 is the output membership layer.

Neurons in this layer represent fuzzy sets used in the consequent of fuzzy rules.

An output membership neuron combines all its inputs by using the fuzzy operation union.

This operation can be implemented by the probabilistic OR. That is,

$$y_i^{(4)} = x_{1i}^{(4)} \oplus x_{2i}^{(4)} \oplus \dots \oplus x_{ki}^{(4)}$$

$$y_{C1}^{(4)} = \mu_{R3} \oplus \mu_{R6} = \mu_{C1}$$

The value of μ_{C1} represents the integrated firing strength of fuzzy rule neurons $R3$ and $R6$.

Layer 5 is the defuzzification layer.

Each neuron in this layer represents a single output of the neuro-fuzzy system. It takes the output fuzzy sets clipped by the respective integrated firing strengths and combines them into a single fuzzy set.

Neuro-fuzzy systems can apply standard defuzzification methods, including the centroid technique.

We will use the sum-product composition method.

The sum-product composition calculates the crisp output as the weighted average of the centroids of all output membership functions.

For example, the weighted average of the centroids of the clipped fuzzy sets $C1$ and $C2$ is calculated as,

$$y = \frac{\mu_{C1} \times a_{C1} \times b_{C1} + \mu_{C2} \times a_{C2} \times b_{C2}}{\mu_{C1} \times b_{C1} + \mu_{C2} \times b_{C2}}$$

4.4.8 Range of AI Applications

Q40. Give the list of various areas where the Artificial Intelligence is used.

Ans :

Applications of Artificial Intelligence

1. Problem Solving
2. Game Playing
3. Theorem Proving
4. Natural Language Processing & Understanding
5. Perception General
 - Speech Reorganization
 - Pattern Reorganization
6. Image Processing
7. Expert System
8. Computer Vision
9. Robotics
10. Intelligent Computer Assisted Instruction
11. Automatic programming
12. Planning & Decision Support systems
13. Engineering Design & Comical Analysis
14. Neural Architecture.
15. Heuristic Classification.

4.4.9 AI Applications And Examples

Q41. Explain about various applications of AI.

Ans :

1 Problem Solving

This is the first application area of AI research., the objective of this particular area of research is how to implement the procedures on AI systems to solve the problems like Human Beings.

2. Game Playing

Much of early research in state space search was done using common board games such as checkers, chess and 8 puzzle. Most games are played using a well defined set of rules. This makes it easy to generate the search space and frees the researcher from many of the ambiguities and complexities inherent in less structured problems. The board Configurations used in playing these games are easily represented in computer, requiring none of complex formalisms. For solving large and complex AI problems it requires lots of techniques like Heuristics. We commonly used the term intelligence seems to reside in the heuristics used by Human beings to solve the problems.

3. Theorem Proving

Theorem proving is another application area of AI research.i.e.,. To prove Boolean Algebra theorems as a humans we first try to prove Lemma.,i.e it tell us whether the Theorem is having feasible solution or not. If the theorem having feasible solution we will try to prove it otherwise discard it., In the same way whether the AI system will react to prove Lemma before trying to attempting to prove a theorem., is the focus of this application area of research.

4. Natural Language understanding

The main goal of this problem is we can ask the question to the computer in our mother tongue the computer can receive that particular language and the system gave the response with in the same language. The effective use of a Computer has involved the use off a Programming Language of a set of Commands that we must use to Communicate with the Computer. The goal of natural language processing is to enable people and language such as English, rather than in a computer language.

It can be divided in to Two sub fields.

i) Natural Language Understanding

Which investigates methods of allowing the Computer to improve instructions given in ordinary English so that Computers can understand people more easily.

ii) Natural Language Generation

This aims to have Computers produce ordinary English language so that people can understand Computers more easily.

5. Perception

The process of perception is usually involves that the set of operations i.e. Touching , Smelling Listening , Tasting , and Eating. These Perceptual activities incorporation into Intelligent Computer System is concerned with the areas of Natural language Understanding & Processing and Computer Vision mainly. There are two major Challenges in the application area of Perception.

i) Speech Reorganization

ii) Pattern Reorganization

i) Speech Reorganization

The main goal of this problem is how the Computer System can recognize our Speeches. (Next process is to understand those Speeches and process them i.e. Encoding & Decoding i.e producing the result in the same language.) It is one is very difficult; Speech Reorganization can be described in two ways.

a) Discrete Speech Reorganization

Means People can interact with the Computer in their mother tongue. In such interaction whether they can insert time gap in between the two words or two sentences (In this type of Speech Reorganization the computer takes some time for searching the database).

b) Continuous Speech Reorganization

Means when we interact with the computer in our mother tongue we can not insert the time gap in between the two words or sentences , i.e. we can talk continuously with the Computer (For this purpose we can increase speed of the computer).

ii) Pattern Reorganization

To identify the regular shape objects, we can see that object from any angle; we can imagine the actual shape of the object (means to picture which part is light fallen) through this we can identify the total structure of that particular object.

-To identify the irregular shape things, we can see that particular thing from any angle; through this we cannot imagine the actual structure. With help of that we can attach the Camera to the computer and picture certain part of the light fallen image with the help of that whether the AI system can recognize the actual structure of the image or not? It is somewhat difficult compare to the regular shape things, till now the research is going on. This is related to the application area of Computer Vision.

A Pattern is a quantitative or structured description of an object or some other entity of interest of an Image. Pattern is found an arrangement of descriptors. Pattern recognition is the research area that studies the operation and design of systems that recognize patterns in data. It includes the discriminate analysis, feature extraction, error estimation, cluster analysis, and parsing (sometimes called syntactical pattern recognition).

Closely Related Areas Pattern Recognition

Artificial Intelligence

Expert systems and machine learning

Neural Networks

Computer Vision

Cognition

Perception

Image Processing

6. Image Processing

Where as in pattern reorganization we can catch the image of real world things with the help of Camera. The goal of Image Processing is to identify the relations between the parts of image.

It is a simple task to attach a Camera to a computer so that the computer can receive visual images. People generally use Vision as their primary means of sensing their environment. We generally see more than we here. i.e. how can we provide such perceptual facilities touch, smell, taste, listen, and eat to the AI System. The goal of Computer Vision research is to give computers this powerful facility for understanding their surroundings. Currently, one of the primary uses of Computer Vision is in the area of Robotics.

Example

We can take a Satellite image to identify the roots and forests; we can make digitize all the image and place on the disk. With the help of particular scale to convert the image in to dots form, later we can identify that particular image at any time. Its one is time consuming process. With the help of "image processing" how to reduce the time to process an image till now the AI research will be continuously going on.

In Image Processing the process of image recognition can be broken into the following main stages.

- a) Image capture
- b) Edge detection
- c) Segmentation
- d) Recognition and Analysis.

Image capturing can be performed by a simple Camera, which converts light signals from a scale of electrical signals., i.e., done by human visual system. We obtained these light signals in a set of 0's and 1's. Each pixel takes on one of a number of possible values often from 0 to 255. Color images are broken down in the same way, but with varying colors instead of gray scales. When a computer receives an image from sensor in form of set of pixels. These pixels are integrated to give the computer an understanding of what it is perceiving.

An image has been obtained, is to determine where the edges are in the image, the very first stage of analysis is called edge detection. Objects in the real world are almost all have solid edges of one kind or another, detecting those images is first step in the process of determining which objects are present in a scene.

Once the edges have been detected, in an image, this information can be used to Segment the image, into homogeneous areas. There are other methods available for segmenting an image, apart from using edge detection, like threshold method. This method involves finding the color of each pixel in an image and considering adjacent pixels to be in the same area as long as their color is similar enough.

A similar method for segmenting images is splitting and merging. Splitting involves taking an area that is not homogeneous and splitting it into two or more smaller areas, each of which is homogeneous. Merging involves taking two areas that are the same as each other, and adjacent to each other and combining them together into a large area. This provides a sophisticated interactive approach to segmenting an image.

Intermediate Level of processing

Low Level Processing High Level Processing

7. Expert System

Expert means the person who had complete knowledge in particular field, ie is called as an expert. The main aim of this problem is with the help of experts, to load their tricks on to the compute and make available those tricks to the other users. The expert can solve the problems with in the time.

The goal of this problem is how to load the tricks and ideas of an expert on to the computer, till now the research will be going on.

8. Computer Vision

It is a simple task to attach a camera to a computer so that the computer can receive visual images. People generally use vision as their primary means of sensing their environment. We generally see more than we here, feel, smell, or taste.

The goal of computer vision research is to give computers this powerful facility for understanding their surroundings. Currently, one of the primary uses of computer vision is in the area of Robotics.

9. Robotics

A robot is an electro – mechanical device that can be programmed to perform manual tasks. The robotics industries association formally defines to move a Robot as a “ Programmable multi-functional manipulator designed to move material, parts, tools, or specialized devices through variable programmed motions for the performance of variety of tasks”.

Not all robotics is considered to be part of AI. A Robot that performs only the actions that it is has been pre-programmed to perform is considered to be a “dumb” robot, includes some kind of sensory apparatus, such as a camera , that allows it to respond to changes in its environment , rather than just to follow instructions “mindlessly”.

10. Intelligent Computer – Assisted Instruction

Computer - Assisted Instruction (CAI) has been used in bringing the power of the computer to bear on the educational process. Now AI methods are being applied to the development of intelligent computerized “ Tutors” that shape their teaching techniques to fit the learning patterns of individual students.

11. Automatic Programming

Programming is the process of telling the computer exactly what we want to do . the goal of automatic programming is to create special programs that act as intelligent “Tools” to assist programmers and expedite each phase of the programming process. The ultimate aim of automatic programming is a computer system that could develop programs by itself, in response to an in according with the specifications of the program developer.

12. Planning and Decision Support System

When we have a goal, either we rely on luck and providence to achieve that goal or we design and implement a plan. The realization of a complex goal may require to construction of a formal and detailed plan. Intelligent planning programs are designed to provide active assistance in the planning process and are expected to be particularly helpful to managers with decision making responsibilities.

13. Engineering Design & Camical Analysis

Artificial Intelligence applications are playing major role in Engineering Drawings & Camical analysis to design expert drawings and Camical synthesis.

14. Neural Architecture

People are more intelligent than Computers,. But AI researchers are trying how make Computers Intelligent. Humans are better at interpreting noisy input, such as recognizing a face in a darkened room from an odd angle. Even where human may not be able to solve some problem, we generally can make a reasonable guess as to its solution. Neural architectures, because they capture knowledge in a large no. of units. Neural architectures are robust because knowledge is distributed somewhat uniformly around the network.

Neural architectures also provide a natural model for parallelism, because each neuron is an independent unit. This showdown searching the data base a massively parallel architecture like the human brain would not suffer from this problem.

15. Heuristic Classification

The term Heuristic means to Find & Discover., find the problem and discover the solution. For solving complex AI problems it's requires lots of knowledge and some represented mechanisms in form of Heuristic Search Techniques., i.e referred to known as Heuristic Classification.

4.5 CASE STUDY

4.5.1 Agricultural Domain – Farmer's Intelligent Assistant

Q42. Write a case study on how, agricultural domain uses artificial intelligence to increase the productivity.

Ans :

Agriculture is the industry that accompanied the evolution of humanity from pre-historic times to modern days and fulfilled faithfully one of its most basic needs: food supply. Today this still remains its core mission, but it's integrated in a more complex than ever mechanism driven by multiple sociological, economic and environmental forces.

Data generated by sensors or agricultural drones collected at farms, on the field or during transportation offer a wealth of information about soil, seeds, livestock, crops, costs, farm equipment or the use of water and fertilizer. Internet of Things technologies and advanced analytics help farmers analyze real time data like weather, temperature, moisture, prices or GPS signals and provide insights on how to optimize and increase yield, improve farm planning, make smarter decisions about the level of resources needed, when and where to distribute them in order to prevent waste.

Efficiency and productivity will increase in the next years as 'precision agriculture' grows bigger and farms become smarter and more connected. It is estimated that by 2020.

While the growing number of connected devices represents a big opportunity for food and agribusiness players, it also adds more complexity for farmers and organizations.

The use of cognitive technologies that help understand, learn, reason, interact and thus, increase efficiency.

1. Help IoT achieve its Maximum Potential

While digital transformation is disrupting the agricultural world and more data comes feed the systems, solutions like the Watson IoT platform enhance value by applying machine learning abilities to sensor or drone data, transforming management systems in real artificial intelligence systems. Cognitive IoT technologies allow many types of correlations of large amount of structured and unstructured data from multiple sources, such as historic weather data, social media posts, research notes, soil information, market place information, images, etc., to extract knowledge and provide organizations with richer insights and recommendations to take action and improve yields.

2. Image Recognition and Insight

Agricultural drones help already farmers scan fields, monitor crops and seeding or analyze plant health. Farm activities can become much more effective when drone data, IoT and computer vision technologies join forces to optimize strategies. These artificial intelligence systems will save time, increase safety and reduce potential human error while improving effectiveness. Agriculture could benefit greatly out of it.

3. Skills and Workforce

This growing urbanization will lead to a decrease of workforce in the rural areas. Innovative technologies using cognitive systems will help address this challenge by easing farmers' work, removing the need for large numbers of people to work the land. Many operations will be done remotely, processes will be automated, risks will be identified and issues solved before occurring. Farmers will be able to take more informed and rapid decisions. In the future, the right mix of skills will probably increasingly be technology and agricultural skills rather than pure agricultural.

4. Determine the best Options to Maximize Return on Crops

The use of cognitive technologies in agriculture could help determine the best crop choice or the best hybrid seed choices for a crop mix adapted to various objectives, conditions and better suited for farm's needs. Watson can use diverse capabilities to understand how seeds react to different soil types, weather forecasts and local conditions. By analyzing and correlating information about weather, type of seeds, types of soil or infestations in a certain area, probability of diseases, data about what worked best, year to year outcomes, marketplace trends, prices or consumer needs, farmers can make decisions to maximize return on crops.

5. Chatbots for Farmers

Chatbots are conversational virtual assistants who automate interactions with end users. Artificial intelligence powered chatbots, using machine learning techniques, understand natural language and interact with users in a personalized way. While it's still early days and chatbots are used mostly by retail, travel, media or insurance players, agriculture could also leverage this emerging technology by assisting farmers with answers to their questions, giving advice and recommendations on specific farm problems.