Syllabus

Course: USCS503	TOPICS (Credits : 03 Lectures/Week:03) Software Testing and Quality Assurance

Objectives:

To provide learner with knowledge in Software Testing techniques. To understand how testing methods can be used as an effective tools in providing quality assurance concerning for software.

To provide skills to design test case plan for testing software

Expected Learning Outcomes:

Understand various software testing methods and strategies. Understand a variety of softwar metrics, and identify defects and managing those defects for improvement in quality for give software. Design SQA activities, SQA strategy, formal technical review report for software quali control and assurance.

Unit	Details	Lectu
I	**Software Testing and Introduction to quality :** Introduction, Nature of errors, an example for Testing, Definition of Quality , QA, QC, QM and SQA , Software Development Life Cycle, Software Quality Factors **Verification and Validation :** Definition of V &V , Different types of V & V Mechanisms, Concepts of Software Reviews, Inspection and Walkthrough **Software Testing Techniques :** Testing Fundamentals, Test Case Design, White Box Testing and its types, Black Box Testing and its types **(Refer chapter 1)**	
II	**Software Testing Strategies :** Strategic Approach to Software Testing, Unit Testing, Integration Testing, Validation Testing, System Testing **Software Metrics :** Concept and Developing Metrics, Different types of Metrics, Complexity metrics **Defect Management :** Definition of Defects, Defect Management Process, Defect Reporting, Metrics Related to Defects, Using Defects for Process Improvement. **(Refer chapter 2)**	
III	**Software Quality Assurance :** Quality Concepts, Quality Movement, Background Issues, SQA activities, Software Reviews, Formal Technical Reviews, Formal approaches to SQA, Statistical Quality Assurance, Software Reliability, The ISO 9000 Quality Standards, , SQA Plan , Six sigma, Informal Reviews **Quality Improvement :** Introduction, Pareto Diagrams, Cause-effect Diagrams, Scatter Diagrams, Run charts **Quality Costs :** Defining Quality Costs, Types of Quality Costs, Quality Cost Measurement, Utilizing Quality Costs for Decision-Making **(Refer chapter 3**	

□□□

CHAPTER 1

Unit I

Artificial Intelligence

Syllabus

What Is AI : Foundations, History and State of the Art of AI.

Intelligent Agents : Agents and Environments, Nature of Environments, Structure of Agents.

Problem Solving by searching : Problem-Solving Agents, Example Problems Searching for Solutions, Uninformed Search Strategies, Informed (Heuristic) Search Strategies, Heuristic Functions.

Syllabus Topic : What is AI

1.1 Introduction to Artificial Intelligence

- John McCarthy who has coined the word "Artificial Intelligence" in 1956, has defined AI as "the science and engineering of making intelligent machines", especially intelligent computer programs.
- **Artificial Intelligence (AI)** is relevant to any intellectual task where the machine needs to take some decision or choose the next action based on the current state of the system, in short act intelligently or rationally. As it has a very wide range of applications, it is truly a universal field.
- In simple words, Artificial Intelligent System works like a Human Brain, where a machine or software shows intelligence while performing given tasks; such systems are called **intelligent systems** or **expert systems**. You can say that these systems can "think" while generating output!!!
- AI is one of the newest fields in science and engineering and has a wide variety of application fields. AI applications range from the general fields like learning, perception and prediction to the specific field, such as writing stories, proving mathematical theorems, driving a bus on a crowded street, diagnosing diseases, and playing chess.
- AI is the study of how to make machines do thing which at the moment people do better. Following are the four approaches to define AI.

Syllabus Topic : Foundations

1.2 Foundations and Mathematical Treatments

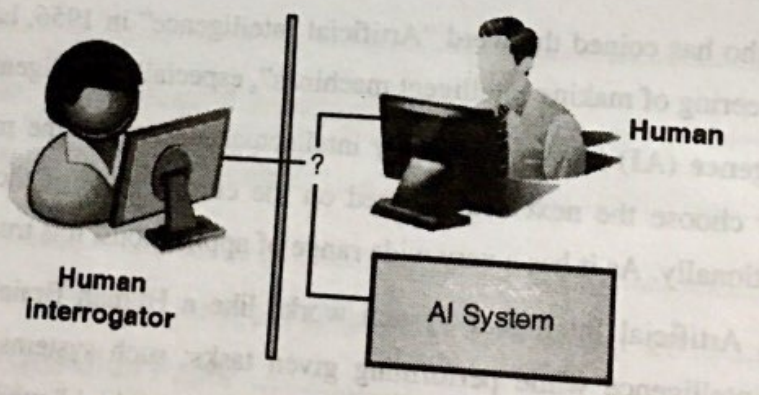
- In general, artificial intelligence is the study of how to make machines do things which at the moment human do better. Following are the four approaches to define AI.
- Historically, all four approaches have been followed by different group of people with different methods.

1.2.1 Acting Humanly : The Turing Test Approach

Definition 1 : "The art of creating machines that perform functions that requires intelligence when performed by people." (Kurzweil, 1990)

Definition 2 : "The study of how to make computers do things at which, at the moment, people are better." (Rich and Knight, 1991)

- To judge whether the system can act like a human, Sir Alan Turing had designed a test known as **Turing test**.
- As shown in Fig. 1.2.1, in Turing test, a computer needs to interact with a human interrogator by answering his questions in written format. Computer passes the test if a human interrogator, cannot identify whether the written responses are from a person or a computer. Turing test is valid even after 60 year of research.



hFig. 1.2.1 : Turing Test Environment

- For this test, the computer would need to possess the following capabilities :

➜ **1. Natural Language Processing (NLP)**

This unit enables computer to interpret the English language and communicate successfully.

➜ **2. Knowledge Representation**

This unit is used to store knowledge gathered by the system through input devices.

3. Automated Reasoning

This unit enables to analyze the knowledge stored in the system and makes new inferences to answer questions.

4. Machine Learning

This unit learns new knowledge by taking current input from the environment and adapts to new circumstances, thereby enhancing the knowledge base of the system.

- To pass total Turing test, the computer will also need to have **computer vision,** which is required to perceive objects from the environment and **Robotics,** to manipulate those objects.

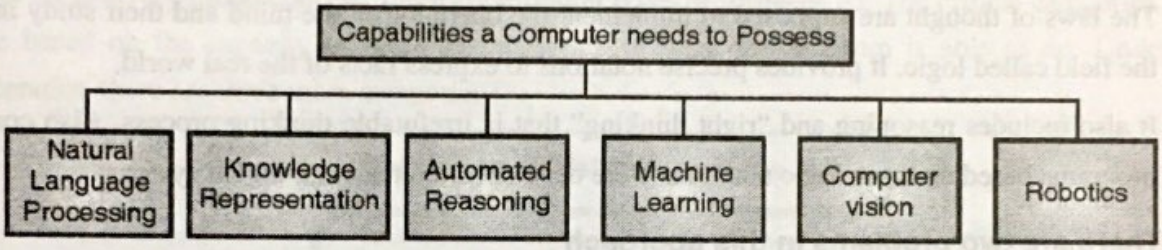


Fig. 1.2.2 : Capabilities a Computer needs to possess

- Fig. 1.2.2 lists all the capabilities a computer needs to have in order to exhibit artificial intelligence. Mentioned above are the six disciplines which implement most of the artificial intelligence.

1.2.2 Thinking Humanly : The Cognitive Modelling Approach

Definition 1 : "The exciting new effort to make computers think ... machines with minds, in the full and literal sense". (Haugeland, 1985)

Definition 2 : "The automation of activities that we associate with human thinking, activities such as decision making, problem solving, learning ..." (Hellman, 1978)

- **Cognitive science :** It is inter disciplinary field which combines computer models from Artificial Intelligence with the techniques from psychology in order to construct precise and testable theories for working of human mind.
- In order to make machines think like human, we need to first understand how human think. Research showed that there are three ways using which human's thinking pattern can be caught.
  1. **Introspection** through which human can catch their own thoughts as they go by.
  2. **Psychological experiments** can be carried out by observing a person in action.
  3. **Brain imaging** can be done by observing the brain in action.

- By catching the human thinking pattern, it can be implemented in computer system as a program and if the program's input output matches with that of human, then it can be claimed that the system can operate like humans.

1.2.3 Thinking Rationally : The "Laws of Thought" Approach

Definition 1 : "The study of mental faculties through the use of computational models". (Charniak and McDermott, 1985)

Definition 2 : "The study of the computations that make it possible to perceive, reason, and act".

- The laws of thought are supposed to implement the operation of the mind and their study initiated the field called logic. It provides precise notations to express facts of the real world.
- It also includes reasoning and "right thinking" that is irrefutable thinking process. Also computer programs based on those logic notations were developed to create intelligent systems.

☞ **There are two problems in this approach :**

1. This approach is not suitable to use when 100% knowledge is not available for any problem.
2. As vast number of computations was required even to implement a simple human reasoning process; practically, all problems were not solvable because even problems with just a few hundred facts can exhaust the computational resources of any computer.

1.2.4 Acting Rationally : The Rational Agent Approach

Definition 1 : "Computational Intelligence is the study of the design of intelligent agents". (Poole et at, 1998)

Definition 2 : "Al ... is concerned with intelligent behaviour in artifacts". (Nilsson, 1998)

☞ **Rational Agent**

- Agents perceive their environment through sensors over a prolonged time period and adapt to change to create and pursue goals and take actions through actuators to achieve those goals. A rational agent is the one that does "right" things and acts rationally so as to achieve the best outcome even when there is uncertainty in knowledge.
- The rational-agent approach has two advantages over the other approaches

1. As compared to other approaches this is the more general approach as, rationality can be achieved by selecting the correct inference from the several available.

2. Rationality has specific standards and is mathematically well defined and completely general and can be used to develop agent designs that achieve it. Human behavior, on the other hand, is very subjective and cannot be proved mathematically.

- The two approaches namely, thinking humanly and thinking rationally are based on the reasoning expected from intelligent systems while; the other two acting humanly and acting rationally are based on the intelligent behaviour expected from them.
- In our syllabus we are going to study acting rationally approach.

1.2.5 Categorization of Intelligent Systems

As AI is a very broad concept, there are different types or forms of AI. The critical categories of AI can be based on the capacity of intelligent program or what the program is able to do. Under this consideration there are three main categories :

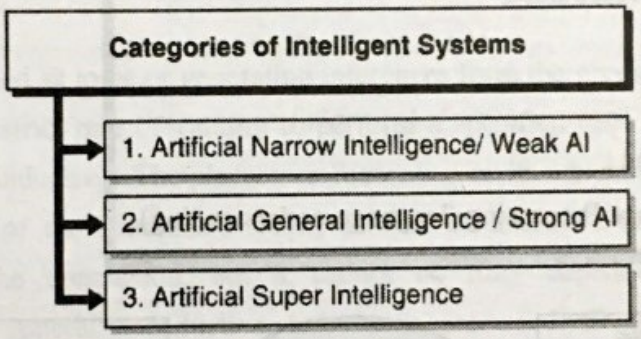


Fig. 1.2.3

➜ **1. Artificial Narrow Intelligence/ Weak AI**

Weak AI is AI that specializes in one area. It is not a general purpose intelligence. An intelligent agent is built to solve a particular problem or to perform a specific task is termed as narrow intelligence or weak AI. For example, it took years of AI development to be able to beat the chess grandmaster, and since then we have not been able to beat the machines at chess. But that is all it can do, which is does extremely well.

➜ **2. Artificial General Intelligence / Strong AI**

Strong AI or general AI refers to intelligence demonstrated by machines in performing any intellectual task that human can perform. Developing strong AI is much harder than developing weak AI. Using artificial general intelligence machines can demonstrate human abilities like reasoning, planning, problem solving, comprehending complex ideas, learning from self experiences, etc. Many companies, corporations' are working on developing a general intelligence but they are yet to complete it.

3. Artificial Super Intelligence

As defined by a leading AI thinker Nick Bostrom, "Super intelligence is an intellect that is much smarter than the best human brains in practically every field, including scientific creativity, general wisdom and social skills." Super intelligence ranges from a machine which is just a little smarter than a human to a machine that is trillion times smarter. Artificial super intelligence is the ultimate power of AI.

1.2.6 Components of AI

AI is a vast field for research and it has got applications in almost all possible domains. By keeping this in mind, components of AI can be identified as follows : Fig. 1.2.4

1. Perception
2. Knowledge representation
3. Learning
4. Reasoning
5. Problem Solving
6. Natural Language Processing (language-understanding).

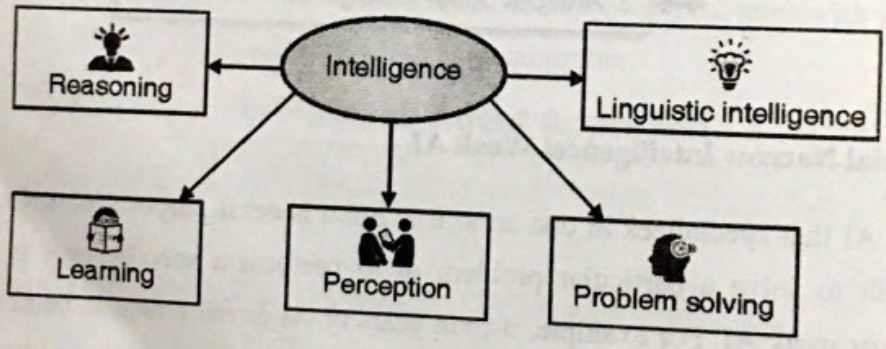


Fig. 1.2.4 : Components of AI

1. Perception

In order to work in the environment, intelligent agents need to scan the environment and the various objects in it. Agent scans the environment using various sense organs like camera, temperature sensor, etc. This is called as perception. After capturing various scenes, perceiver analyses the different objects in it and extracts their features and relationships among them.

2. Knowledge representation

The information obtained from environment through sensors may not be in the format required by the system. Hence, it need to be represented in standard formats for further processing like

learning various patterns, deducing inference, comparing with past objects, etc. There are various knowledge representation techniques like Prepositional logic and first order logic.

3. Learning

Learning is a very essential part of AI and it happens in various forms. The simplest form of learning is by trial and error. In this form the program remembers the action that has given desired output and discards the other trial actions and learns by itself. It is also called as unsupervised learning. In case of rote learning, the program simply remembers the problem solution pairs or individual items. In other case, solution to few of the problems is given as input to the system, basis on which the system or program needs to generate solutions for new problems. This is known as supervised learning.

4. Reasoning

Reasoning is also called as logic or generating inferences form the given set of facts. Reasoning is carried out based on strict rule of validity to perform a specified task. Reasoning can be of two types, deductive or inductive. The deductive reasoning is in which the truth of the premises guarantees the truth of the conclusion while, in case of inductive reasoning, the truth of the premises supports the conclusion, but it cannot be fully dependent on the premises. In programming logic generally deductive inferences are used. Reasoning involves drawing inferences that are relevant to the given problem or situation.

5. Problem-solving

AI addresses huge variety of problems. For example, finding out winning moves on the board games, planning actions in order to achieve the defined task, identifying various objects from given images, etc. As per the types of problem, there is variety of problem solving strategies in AI. Problem solving methods are mainly divided into general purpose methods and special purpose methods. General purpose methods are applicable to wide range of problems while, special purpose methods are customized to solve particular type of problems.

6. Natural Language Processing

Natural Language Processing, involves machines or robots to understand and process the language that human speak, and infer knowledge from the speech input. It also involves the active participation from machine in the form of dialog i.e. NLP aims at the text or verbal output from the machine or robot. The input and output of an NLP system can be speech and written text respectively.

1.2.7 Computational Intelligence Vs Artificial Intelligence

Computational Intelligence (CI)	Artificial Intelligence (AI)
Computational Intelligence is the study of the design of intelligent agents	Artificial Intelligence is study of making machines which can do things which at presents human do better.
CI involves numbers and computations.	AI involves designs and symbolic knowledge representations.
CI constructs the system starting from the bottom level computations, hence follows bottom-up approach.	AI analyses the overall structure of an intelligent system by following top down approach.
CI concentrates on low level cognitive function implementation.	AI concentrates of high level cognitive structure design.

Syllabus Topic : History of Artificial Intelligence

1.3 History of Artificial Intelligence

- The term **Artificial Intelligence (AI)** was introduced by John McCarthy, in 1955. He defined artificial intelligence as "The science and engineering of making intelligent machines".
- Mathematician Alan Turing and others presented a study based on logic driven computational theories which showed that any computer program can work by simply shuffling "0" and "1" (i.e. electricity off and electricity *on*). Also, during that time period, research was going on in the areas like Automations, Neurology, Control theory, Information theory, etc.
- This inspired a group of researchers to think about the possibility of creating an electronic brain. In the year 1956 a conference was conducted at the campus of Dartmouth College where the field of artificial intelligence research was founded.
- This conference was attended by John McCarthy, Marvin Minsky, Allen Newell and Herbert Simon, etc., who are supposed to be the pioneers of artificial intelligence research for a very long time. During that time period, Artificial Intelligence systems were developed by these researchers and their students.

☞ **Let's see few examples of such artificial intelligent systems :**

1. **Game - Checkers :** Computer played as an opponent,
2. **Education - Algebra :** For solving word problems,
3. **Education - Math :** Proving logical theorems,

4. **Education - Language :** Speaking English, etc.

- During that time period these founders predicted that in few years machines can do any work that a man can do, but they failed to recognize the difficulties which can be faced.
- Meanwhile we will see the ideas, viewpoints and techniques which Artificial Intelligence has inherited from other disciplines. They can be given as follows :

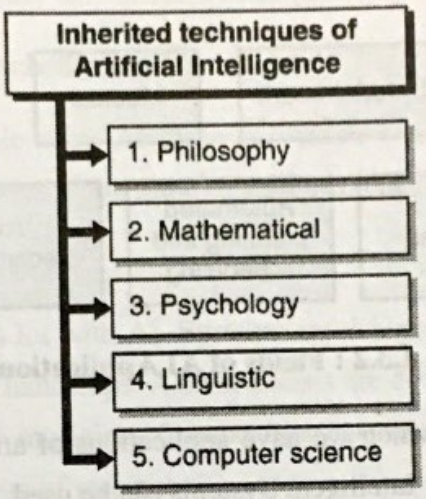


Fig. 1.3.1

➜ **1. Philosophy**

Theories of reasoning and learning have emerged, along with the view port that the mind is constituted by the operation of a physical system.

➜ **2. Mathematical**

Formal theories of logic, probability, decision making and computation have emerged.

➜ **3. Psychology**

Psychology has emerged tools to investigate the human mind and a scientific language which are used to express the resulting theories.

➜ **4. Linguistic**

Theories of the structure and meaning of language have emerged.

➜ **5. Computer science**

The tools which can make artificial intelligence a reality has emerged.

1.3.1 Applications of Artificial Intelligence

- You must have seen use of Artificial Intelligence in many SCI-FI movies. To name a few we have I Robot, Wall-E, The Matrix Trilogy, Star Wars, etc. movies. Many a times these movies show

positive potential of using AI and sometimes also emphasize the dangers of using AI. Also there are games based on such movies, which show us many probable applications of AI.

- Artificial Intelligence is commonly used for problem solving by analyzing or/and predicting output for a system. AI can provide solutions for constraint satisfaction problems. It is used in wide range of fields for example in diagnosing diseases, in business, in education, in controlling a robots, in entertainment field, etc.

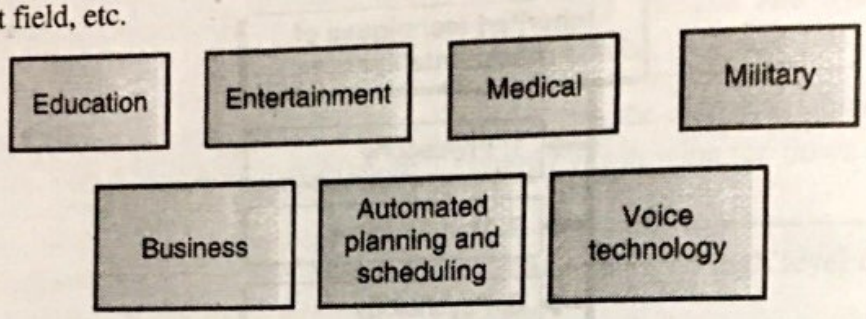


Fig. 1.3.2 : Fields of AI Application

- Fig. 1.6.1 shows few fields in which we have applications of artificial intelligence. There can be many fields in which Artificially Intelligent Systems can be used.

➜ **1. Education**

Training simulators can be built using artificial intelligence techniques. Software for pre-school children are developed to enable learning with fun games. Automated grading, Interactive tutoring, instructional theory are the current areas of application.

➜ **2. Entertainment**

Many movies, games, robots are designed to play as a character. In games they can play as an opponent when human player is not available or not desirable.

➜ **3. Medical**

AI has applications in the field of cardiology (CRG), Neurology (MRI), Embryology (Sonography), complex operations of internal organs, etc. It can be also used in organizing bed schedules, managing staff rotations, store and retrieve information of patient. Many expert systems are enabled to predict the decease and can provide with medical prescriptions.

➜ **4. Military**

Training simulators can be used in military applications. Also areas where human cannot reach or in life stacking conditions, robots can be very well used to do the required jobs. When decisions have to be made quickly taking into account an enormous amount of information, and when lives are at stake, artificial intelligence can provide crucial assistance. From developing intricate flight

plans to implementing complex supply systems or creating training simulation exercises, AI is a natural partner in the modern military.

➜ **5. Business and Manufacturing**

Latest generation of robots are equipped well with the performance advances, growing integration of vision and an enlarging capability to transform manufacturing.

➜ **6. Automated planning and scheduling**

Intelligent planners are available with AI systems, which can process large datasets and can consider all the constraints to design plans satisfying all of them.

➜ **7. Voice Technology**

Voice recognition is improved a lot with AI. Systems are designed to take voice inputs which are very much applicable in case of handicaps. Also scientists are developing an intelligent machine to emulate activities of a skillful musician. Composition, performance, sound processing, music theory are some of the major areas of research.

➜ **8. Heavy Industry**

Huge machines involve risk in operating and maintaining them. Human robots are better replacing human operators. These robots are safe and efficient. Robot are proven to be effective as compare to human in the jobs of repetitive nature, human may fail due to lack of continuous attention or laziness.

1.3.2 Sub Areas/ Domains of Artificial Intelligence

– AI Applications can be roughly classified based on the type of tools/approaches used for inoculating intelligence in the system, forming sub areas of AI. Various sub domains/ areas in intelligent systems can be given as follows; Natural Language Processing, Robotics, Neural Networks and Fuzzy Logic. Fig. 1.3.3 shows these areas in Intelligent Systems.

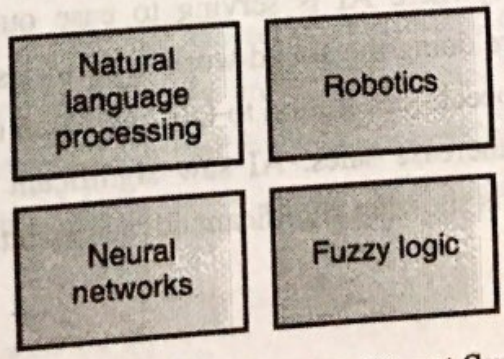


Fig. 1.3.3 : Sub-areas in Intelligent Systems

1. Natural language processing

One of the application of AI is in field of Natural Language Processing (NLP). NLP enables interaction between computers and human (natural) language. Practical applications of NLP are in machine translation (e.g. Lunar System), information retrieval, text categorization, etc. Few more applications are extracting 3D information using vision, speech recognition, perception, image formation.

2. Robotics

One more major application of AI is in Robotics. Robot is an active agent whose environment is the physical world. Robots can be used in manufacturing and handling material, in medical field, in military, etc. for automating the manual work.

3. Neural networks

Another application of AI is using Neural Networks. Neural Network is a system that works like a human brain/nervous system. It can be useful for stock market analysis, in character recognition, in image compression, in security, face recognition, handwriting recognition, Optical Character Recognition (OCR), etc.

4. Fuzzy logic

Apart from these AI systems are developed with the help of Fuzzy Logic. Fuzzy Logic can be useful in making approximations rather than having a fixed and exact reasoning for a problem. You must have seen systems like AC, fridge, washing machines which are based on fuzzy logic (they call it "6th sense technology!").

Syllabus Topic : State of the art of AI

1.4 State of the Art of Artificial Intelligence

Artificial Intelligence has touched each and every aspect of our life. From washing machine, Air conditioners, to smart phones everywhere AI is serving to ease our life. In industry, AI is doing marvellous work as well. Robots are doing the sound work in factories. Driverless cars have become a reality. WiFi-enabled Barbie uses speech-recognition to talk and listen to children. Companies are using AI to improve their product and increase sales. AI saw significant advances in machine learning. Following are the areas in which AI is showing significant advancements.

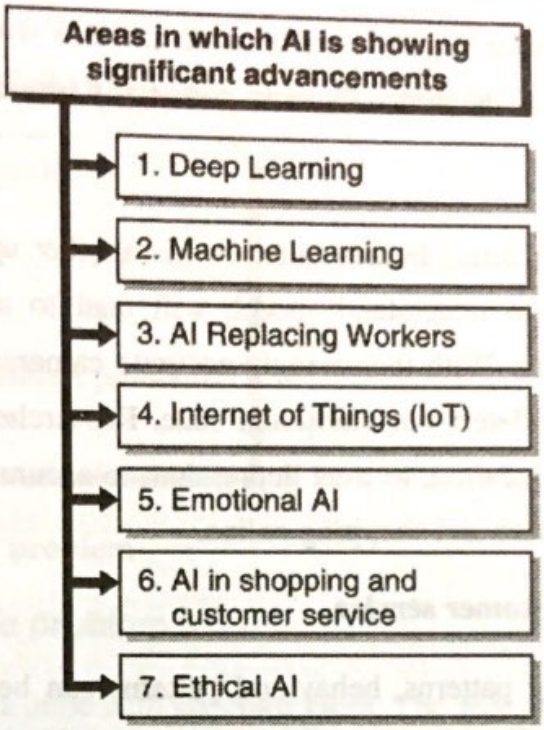


Fig. 1.4.1

➜ 1. Deep Learning

Convolutional Neural Networks enabling the concept of deep learning is the top most area of focus in Artificial intelligence in todays' era. Many problems and applications areas of AI like, natural language and text processing, speech recognition, computer vision, information retrieval, and multimodal information processing empowered by multi-task deep learning.

➜ 2. Machine Learning

The goal of machine learning is to program computers to use example data or past experience to solve a given problem. Many successful applications of machine learning include systems that analyse past sales data to predict customer behaviour, optimize robot behaviour so that a task can be completed using minimum resources, and extract knowledge from bioinformatics data.

➜ 3. AI Replacing Workers

In industry where there are safety hazards, robots are doing a good job. Human resources are getting replaced by robots rapidly. People are worries to see that the white color jobs of data processing are being done exceedingly well by intelligent programs. A study from The National Academy of Sciences brought together technologists and economists and social scientists to figure out what's going to happen.

➜ 4. Internet of Things (IoT)

The concepts of smarter homes, smarter cars and smarter world is evolving rapidly with the invention of internet of things. The future is no far when each and every object will be wirelessly

connected to something in order to perform soma smart actions without any human instructions or interference. The worry is how the mined data can potentially be exploited.

5. Emotional AI

Emotional AI, where AI can detect human emotions, is another upcoming and important area of reserch. Computers' ability to understand speech will lead to an almost seamless interaction between human and computer. With increasingly accurate cameras, voice and facial recognition, computers are better able to detect our emotional state. Researchers are exploring how this new knowledge can be used in education, to treat depression, to accurately predict medical diagnoses, and to improve customer service and shopping online.

6. AI in shopping and customer service

Using AI, customers' buying patterns, behavioral patterns can be studied and systems that can predict the purchase or can help customer to figure out the perfect item. AI cab be used to find out what will make the customer happy or unhappy. For example, if a customer is shopping online, like a dress pattern but needs dark shades and thick material, computer understand the need and brings out new set of perfectly matching clothing for him.

7. Ethical AI

With all the evolution happening in technology in every walk of life, ethics must be considered at the forefront of research. For example, in case of driverless car, while driving, if the decision has to be made between weather to dash a cat or a lady having both in an uncontrollable distance in front of the car, is an ethical decision. In such cases how the programming should decide who is more valuable, is a question. These are not the problems to be solved by computer engineers or research scientists but someone has to come up with an answer.

1.5 Problem Solving with Artificial Intelligence

1.5.1 Problems

- Early work in the field of AI focused on formal tasks, like game playing, proving theorems, etc.
- Games like chess, checkers received good deal of attention, because in case of machine opponents, system was using the experience gained in previous games to improve the moves in next game.
- **General Problem Solving (GPS)** was developed for commonsense reasoning problems.
- GPS used less amount of knowledge for performing simple tasks. When the knowledge base was scaled up for the problems which needed perception through vision and/or speech, natural language understanding, problem solving for medical diagnosis, chemical analysis, etc. it was difficult because it involved analog signals which are generally very noisy compared to digital signals.

– Following are some of the famous search problems in artificial intelligence.

1.5.1.1 Classic Artificial Intelligence Search Problems

1. 3*3*3 Rubik's cube problem
2. 8-Puzzle
3. N-queen problem
4. Missionaries and cannibals problem
5. The river problem

➜ **1. 3*3*3 Rubik's cube problem**

☞ **Do I need to explain the problem ?**

In Rubik's cube, we have a cube with six color faces. The goal is to arrange all the cuboids in such a way that each face of cube will show distinct color as shown in the Fig. 1.5.1.

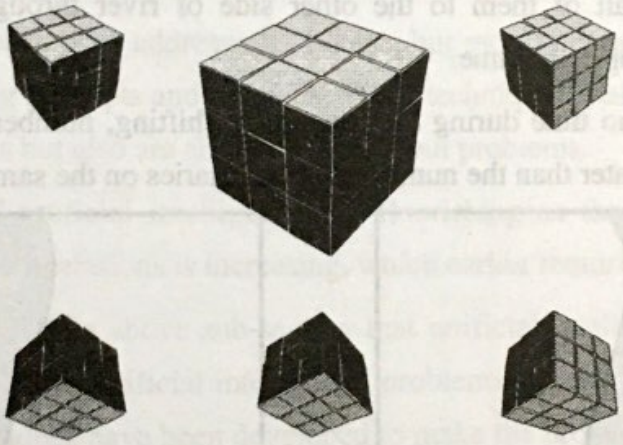


Fig. 1.5.1 : Rubik's cube Problem

➜ **2. 8-Puzzle**

7	–	4
5	2	6
8	3	1

Initial State

–	1	2
3	4	5
6	7	8

Goal State

Fig. 1.5.2

In 8-puzzle there are 8 tiles need to be arranged in a way showed in the goal state. The condition is only the blank tile can be moved to immediate up down, right or left positions and the goal state is to be attained in minimum number of moves.

3. N-queen problem

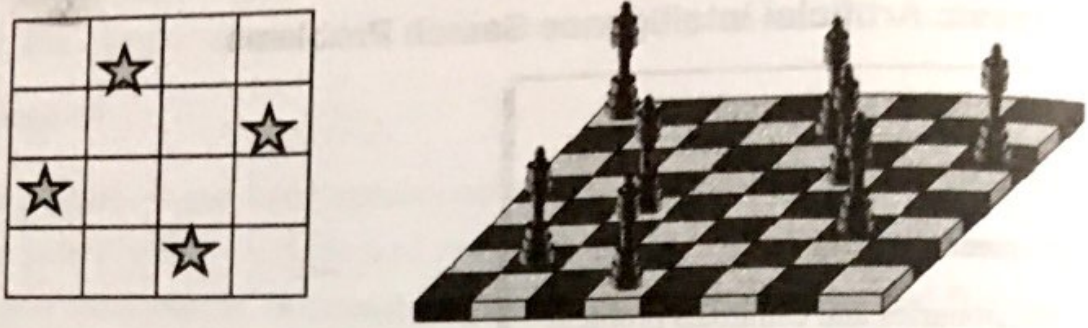


Fig. 1.5.3 : N-Queen Problem

In n-queen, the queens need to be placed on the n*n board, in such a way that no queen can dash the other queen, horizontally, vertically or diagonally.

4. Missionaries and cannibals problem

- In this problem, there are three missionaries and three cannibals on the same side of a river.
- We need to get all of them to the other side of river through a canoe which can hold maximum two people at a time.
- The condition is no time during the process of shifting, number of cannibals on any of the side should be greater than the number of missionaries on the same side.

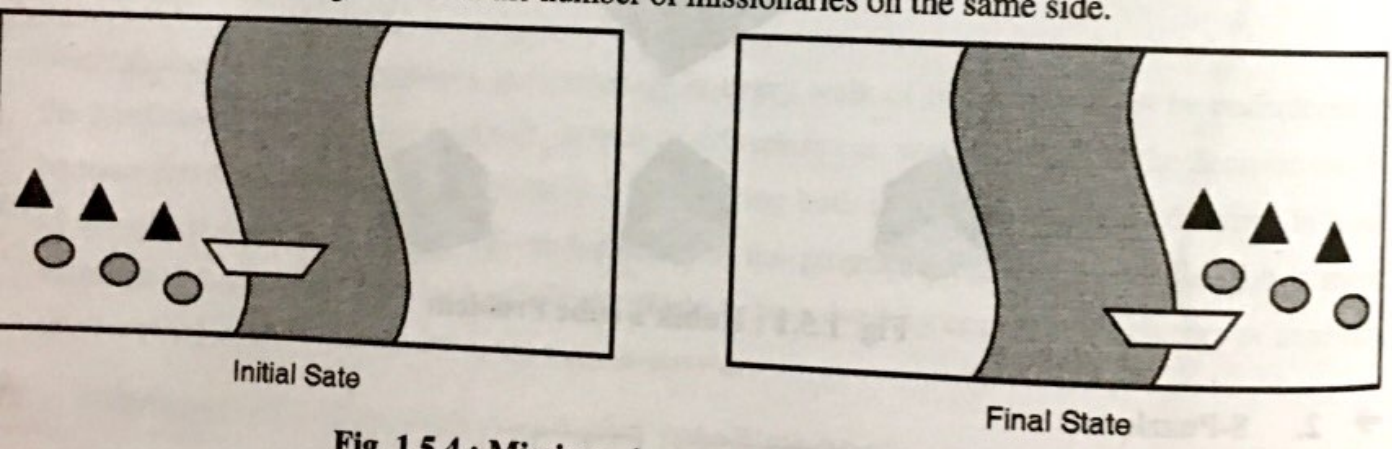


Fig. 1.5.4 : Missionaries and cannibals Problem

5. The river problem

- In this problem, a farmer needs to carry a wolf, a duck and corn across a river. The farmer has a small rowing boat, which can carry at most the farmer and one other thing. The problem is that, the wolf will eat the duck and the duck will eat the corn, if they are at the same side. How can the farmer safely transport the wolf, the duck and the corn to the opposite shore?

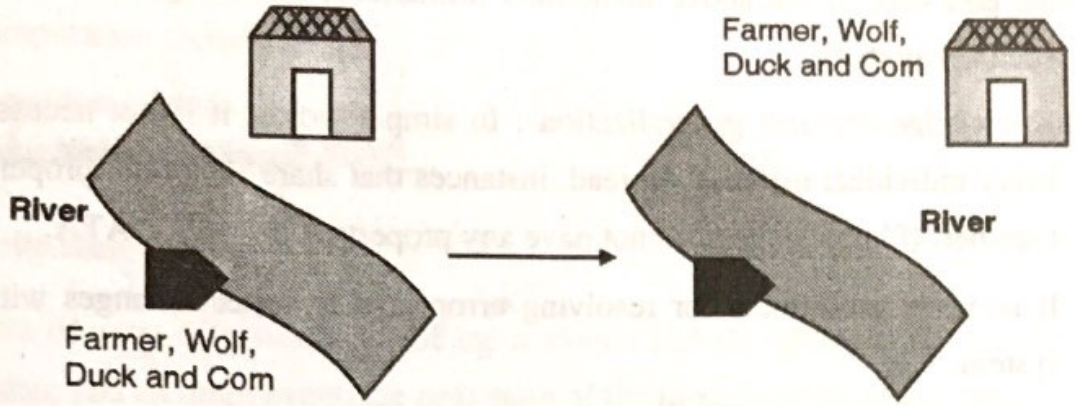


Fig. 1.5.5 : The river problem

- From the above examples, it must be clear that AI problems are the one in which there are few conditions specified and the aim is to not only generate the solution but also to improve performance of the system; because that is where the intelligence of the system gets challenged. The term intelligence includes many cognitive skills, like the ability to solve problems, learn, interpret and understand language.
- Artificial intelligence does address all of these, but more progress has been made in the area of problem solving concepts and methods. These techniques enable to build programs that not only find solutions but also are able to reason about problems.
- Problem areas of artificial intelligence are flourishing as the need for expert systems to perform day to day operations is increasing, which earlier required human expertise.
- It can be observed from above sub-section that artificial intelligence problems have a very large scope. Also those artificial intelligence problems are vivid and hard to solve. Over the decades few techniques have been developed to make the problem solving simple and easy to tackle.
- Research showed that intelligence requires knowledge and knowledge holds some less desirable properties like :
  - It is enormous.
  - It is tough to characterize precisely.
  - It is dynamic.
  - It is structured in a way that matches to the ways it will be used.
- This information is useful for developing Artificial Intelligence techniques. As artificial intelligence technique makes use of knowledge to develop solutions.

- To take care of the above mentioned limitations, knowledge should be represented in such a way that :
- Knowledge captures generalization : In simple words, it is not necessary to represent every individual instance. Instead, instances that share important properties are grouped together. If knowledge does not have any property it is called DATA.
- It is easily modifiable for resolving errors and to reflect changes with respect to the system.
- It should be feasible for **more number of instances.**

1.5.2 Artificial Intelligence Techniques

– Following are the artificial intelligence techniques that behave intelligently and can handle all the limitations of knowledge.

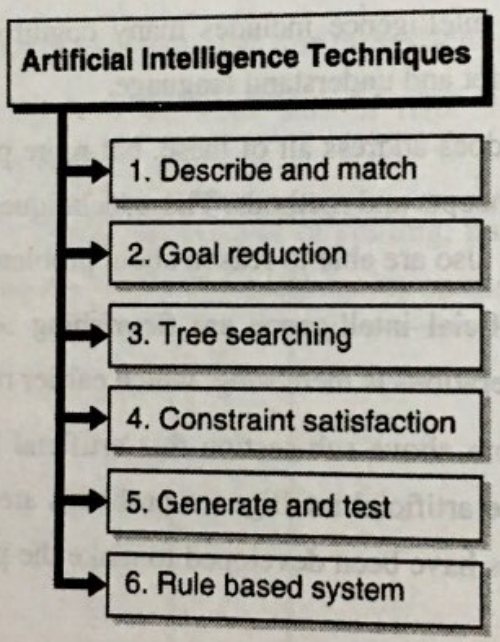


Fig. 1.5.6

– Biological-inspired AI Techniques :
  - Neural networks
  - Genetic algorithms
  - Reinforcement learning

Let us understand them one by one.

➜ **1. Describe and match :**

In this technique, system's behaviour is explained in terms of a finite state model and computation model.

1. Finite state model
2. Computation model
3. Transition relation

➜ 1. Finite state model

It consists of a set of states, a set of input events and the relations between them. Based on the current state and an input event, the next state of the model can be determined.

➜ 2. Computation model

It is a finite state machine which includes a set of states, a start state, a transition function and an input alphabet. The transition function provides mapping between input symbols and current states to a next state.

➜ 3. Transition relation

If a pair of states (S, S') is such that one move takes the system from S to S', then the transition relation is represented by S → S'. State-transition system is called deterministic if every state has at most one successor; it is called non-deterministic if at least one state has more than one successor.

- Representation of the computational system includes start and end state descriptions and a set of possible transition rules that might be applied. Problem is to find the appropriate transition rules.
- Example : Problem of Towers of Hanoi with two discs. The state transitions are shown for the same.

☞ Problem Definition

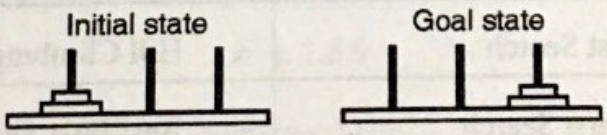


Fig. 1.5.7

- Move the disks from the leftmost post to the rightmost post with following conditions :
  - A large disk can never be put on top of a smaller one;
  - Only one disk can be moved at a time, from one peg to another;
  - The middle post is only to be used for intermediate storage.
- Complete the task in the smallest number of moves possible.
- Possible state transitions in the Towers of Hanoi puzzle with 2 disks.

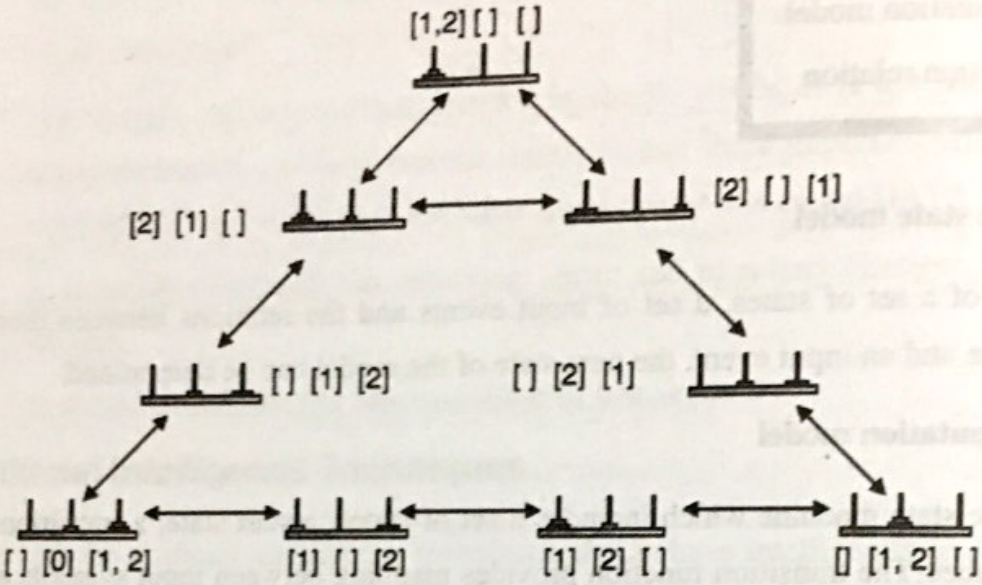


Fig. 1.5.8

- In this problem, the optimal solution is the sequence of transitions from the start state downward to the extreme left branch of the transition tree.

2. Tree searching

- Tree searching is a very commonly used technique to solve many problems. Goal reduction, constraint networks are few of the areas. Tree is searched through many nodes to obtain the goal node and it gives the path from source node to destination node.
- If each node of the entire tree is explored while search for the goal node, it is called as exhaustive search. All the searching techniques of AI are broadly classified as uninformed searching and informed searching.

Un-informed Searching Techniques	Informed Searching techniques
– Depth First Search	– Hill Climbing
– Breadth First Search	– Best First Search
– Uniform Cost Search	– A* Search
– Depth Limited Search	– Iterative deepening A*
– Iterative Deepening DFS	– Beam Search
– Bidirectional Search	– AO* Search

3. Goal reduction

- In goal reduction technique, the main goal is divided into sub goals. It requires some procedures to follow. Goal reduction procedures are an alternative to declarative and logic based representations.
- Goal reduction process includes hierarchical sub division of goals into sub goals is carried out until the sub goals have an immediate solution. An AND-OR tree/graph structure can represent relations between goals and sub-goals, alternative sub-goals and conjoint sub-goals.
- There are different goal levels. Higher-level goals are higher in the tree, and lower level goals are lower in the tree. There are directed arcs from a higher level to lower level nodes, representing the reduction of higher level goal to lower level sub goals. Nodes are at the leaf nodes of the tree. These goals cannot be divided further.

Example

An AND-OR tree structure to represent facts such as "enjoyment", "earning/save money", "old age" etc.

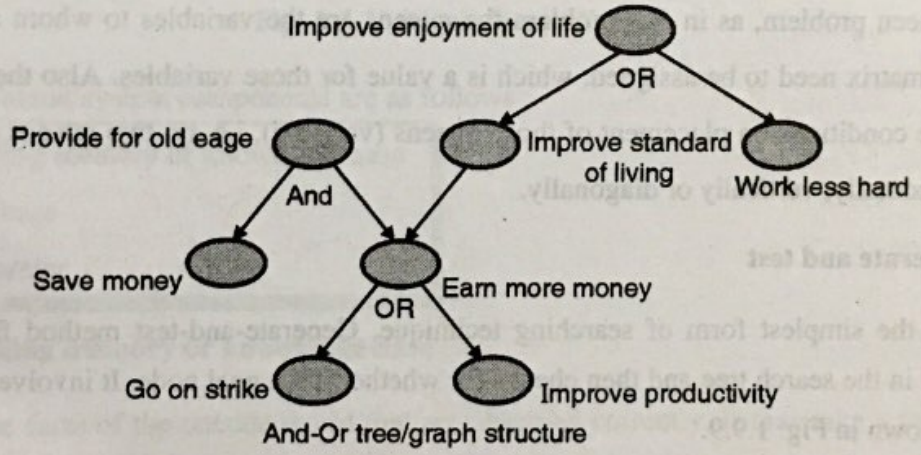


Fig. 1.5.9

The AND-OR tree structure describes following things :

- Hierarchical relationships between goals and sub-goals
- The "Earn more money", is a sub-goal of "Improve standard of living", which in turn is a sub-goal of "Improve enjoyment of life".
- Alternative ways of trying to solve a goal
- The "Go on strike" and "Improve productivity" are alternative ways of trying to "Earn more money".
- Conjoint sub-goals

- These are the goal which depends on more than one sub-goals.
- To "Provide for old age", not only need to "Earn more money", but also need to "Save money".

4. Constraint satisfaction

- Constraint satisfaction is a process of generating solution that satisfies all the specified constraints for a given problem. There are variable which needs to get assigned values from a specific domain.
- To generate the solution it uses backtracking mechanism. The optimal solution is the one which is generated in minimum number of backtracks and satisfies all the constraints, thereby assigning proper values to each of the variable.
- There are multiple fields having application of constraint satisfaction. Artificial Intelligence, Programming Languages, Symbolic Computing, and Computational Logic are few of them to name.
- Example :

N-queen problem, as in this problem the queens are the variables to whom a position in an n*n matrix need to be assigned, which is a value for those variables. Also the problem states some conditions on placement of those queens (variable), i.e. no two queens can clash either horizontally, vertically or diagonally.

5. Generate and test

- It is the simplest form of searching technique. Generate-and-test method first generates a node in the search tree and then checks for whether it's a goal node. It involves two processes as shown in Fig. 1.9.9.
  1. Generate the successor node.
  2. Test to match it with each of the proposed goal node.

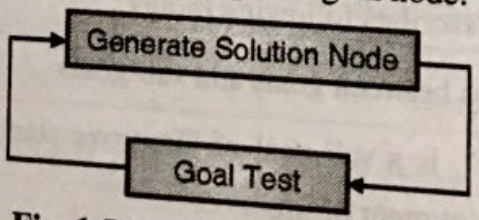


Fig. 1.5.10 : Generate and Test

- As there are many unfruitful nodes gets generated during this process, it's not a very efficient technique.

- Example

Problem of opening a combination lock by trial and error technique without knowing the combination.

➜ 6. Rule based system

- Rule based systems are the simplest and most useful systems. In its simplest form they have set of rules, an interpreter and input from the environment. Rules are of the form IF <condition> THEN <action>.

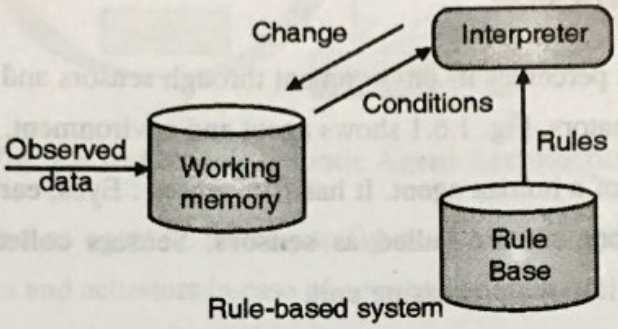


Fig. 1.5.11 : Rule Based System

- Rule based system components are as follows :

1. Working memory or knowledge base
2. Rule base
3. Interpreter

➜ 1. Working memory or knowledge base

It stores the facts of the outside world that are observed currently. It may take a form of a triplet< Object, Attribute, Value>. For example : To represent the fact that "The colour of my car is black", the knowledge base will store<car, colour black> triplet.

➜ 2. Rule base

Rule base is the basic part of the system. It contains all the rules. Rules are generated from the domain knowledge and can be used or modified based on the current observations.

For example : IF <(temperature, over, 20)>

THEN <add (ocean, swimmable, yes)>

While using a particular rule, first the conditions are matched to the current observations and if satisfied, then rule may be fired.

3. Interpreter

It performs the matching of the rule with respect to the current precepts observed in the environment. It has three repetitive tasks to follow : Retrieval of the matching rule, refinement of the rule and execution of the rule by performing the corresponding action.

Syllabus Topic : Intelligent Agents, Agents and Environments

1.6 Intelligent Agents

1.6.1 What is an Agent ?

- **Agent** is something that perceives its environment through sensors and acts upon that environment through effectors or actuators. Fig. 1.6.1 shows agent and environment.
- Take a simple example of a human agent. It has five senses : Eyes, ears, nose, skin, tongue. These senses sense the environment are called as **sensors**. Sensors collect percepts or inputs from environment and passes it to the processing unit.
- Actuators or effectors are the organs or tools using which the agent acts upon the environment. Once the sensor senses the environment, it gives this information to nervous system which takes appropriate action with the help of actuators.
- In case of human agents we have hands, legs as actuators or effectors.

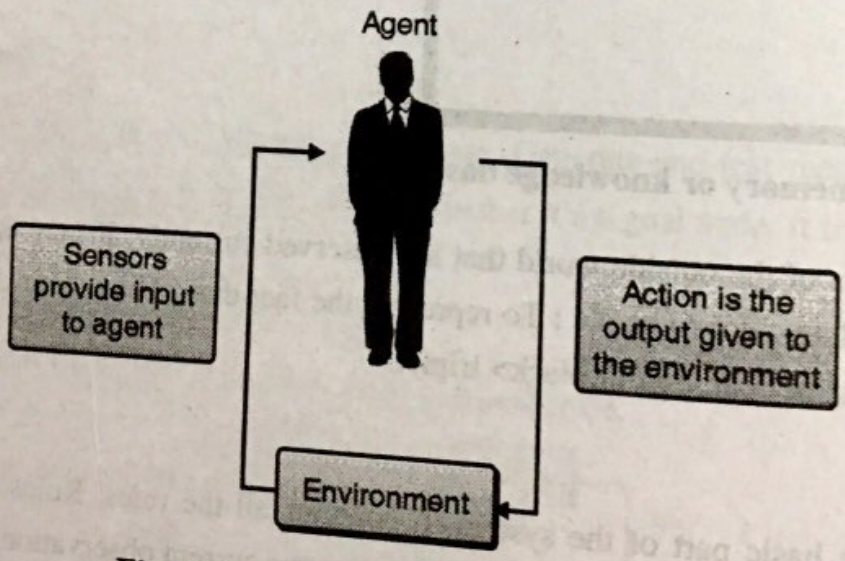


Fig. 1.6.1 : Agent and Environment

- Fig. 1.6.2 shows generic robotic agent structure.

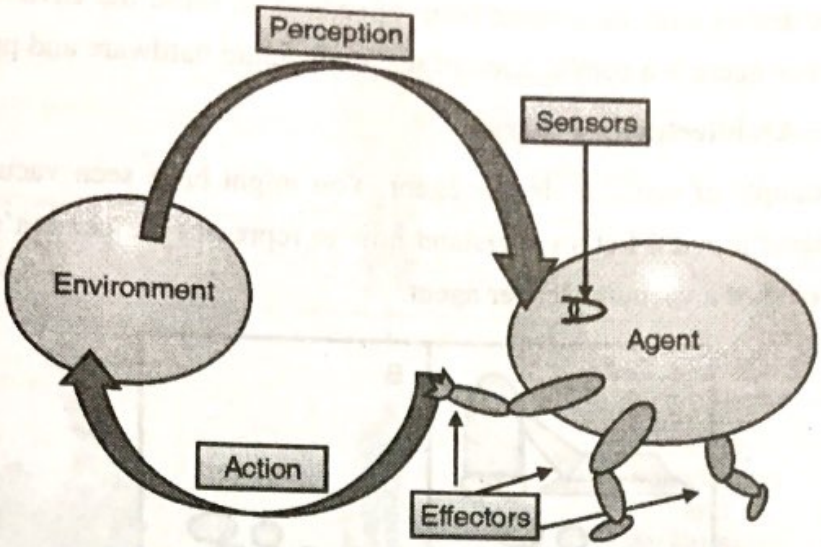


Fig. 1.6.2 : Generic Robotic Agent Architecture

- After understanding what an agent is, let's try to figure out sensor and actuator for a robotic agent, can you think of sensors and actuators in case of a robotic agent?
- The robotic agent has cameras, infrared range finders, scanners, etc. used as **sensors**, while various types of motors, screen, printing devices, etc. used as **actuators** to perform action on given input.

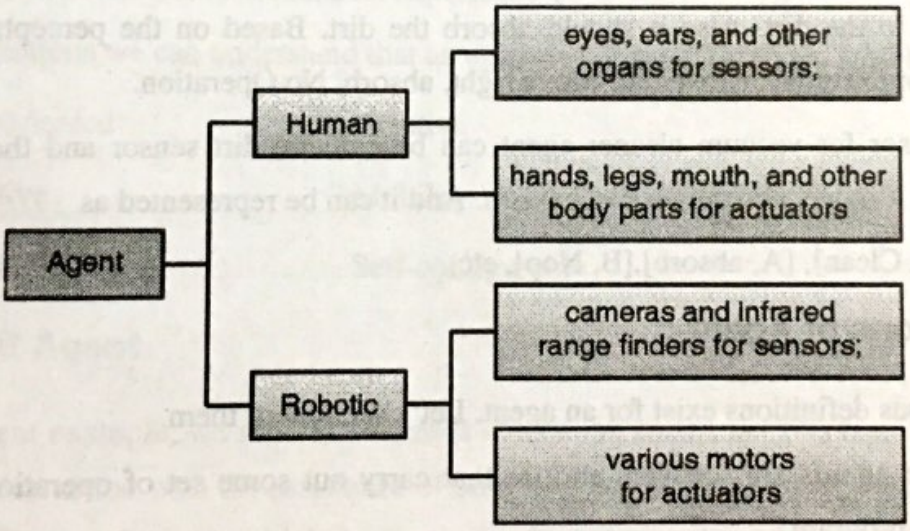


Fig. 1.6.3 : Sensors and Actuators in Human and Robotic Agent

- The **agent function** is the **description of what all functionalities** the agent is supposed to do. The agent function provides mapping between percept sequences to the desired actions. It can be represented as [f : P* ⇒ A]
- **Agent program is a computer program that** implements agent function in an architecture suitable language. Agent programs needs to be installed on a device in order to run the device

accordingly. That device must have some form of sensors to sense the environment and actuators to act upon it. Hence agent is a combination of the architecture hardware and program software.

Agent = Architecture + Program

Take a simple example of vacuum cleaner agent. You might have seen vacuum cleaner agent in "WALL-E"(animated movie). Let's understand how to represent the percept's (input) and actions (outputs) used in case of a vacuum cleaner agent.

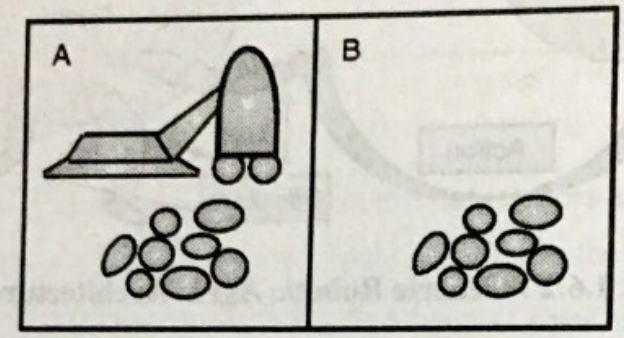


Fig. 1.6.4 : Vacuum cleaner Agent

As shown in Fig. 1.6.4, there are two blocks A and B having some dirt. Vacuum cleaner agent supposed to sense the dirt and collect it, thereby making the room clean. In order to do that the agent must have a camera to see the dirt and a mechanism to move forward, backward, left and right to reach to the dirt. Also it should absorb the dirt. Based on the percepts, actions will be performed. For example : Move left, Move right, absorb, No Operation.

Hence the sensor for vacuum cleaner agent can be camera, dirt sensor and the actuator can be motor to make it move, absorption mechanism. And it can be represented as

[A, Dirty], [B, Clean], [A, absorb],[B, Nop], etc.

1.6.2 Definitions of Agent

There are various definitions exist for an agent. Let's see few of them.

IBM states that **agents** are software entities that carry out some set of operations on behalf of a user or another program.

FIPA : Foundation for Intelligent Physical Agents (FIPA) terms that, an **agent** is a computational process that implements the autonomous functionality of an application.

Another definition is given as "An agent is anything that can be viewed as perceiving its environment through sensors and acting upon the environment through effectors".

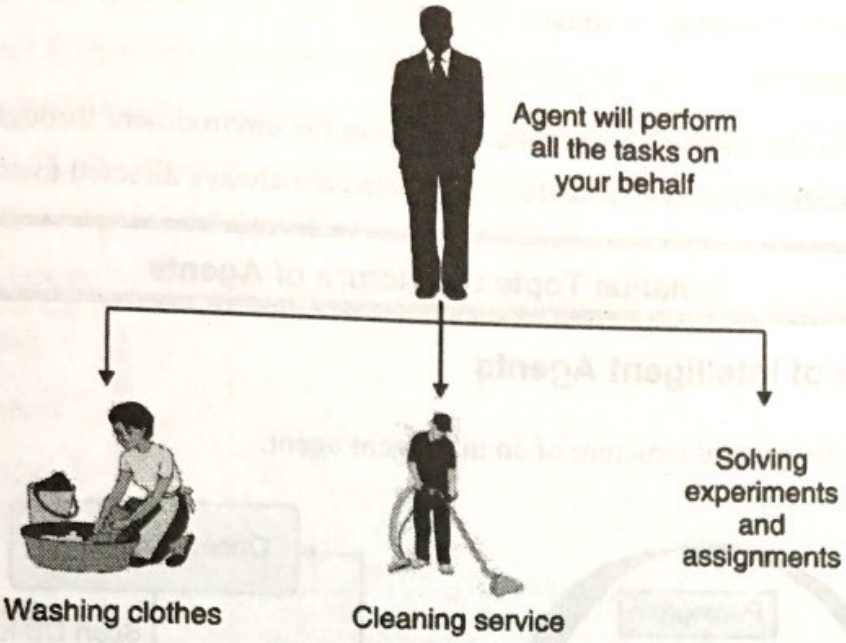


Fig. 1.6.5 : Interactive Intelligent Agent

- By Russell and Norvig, F. Mills and R. Stufflebeam's definition says that "**An agent** is anything that is capable of acting upon information it perceives. An intelligent agent is an agent capable of making decisions about how it acts based on experience".
- From above definitions we can understand that an agent is : (As per Terziyan, 1993)

Goal-oriented	Creative
Adaptive	Mobile
Social	Self-configurable

1.6.3 Intelligent Agent

- In the human agent example, we read that there is something called as **"Nervous System"** which helps in deciding an action with the assistance of effectors, based on the input given by sensors. In robotic agent, we have software's which demonstrates the functionality of nervous system.
- **Intelligent agent** is the one which can take input from the environment through its sensors and act upon the environment through its actuators. Its actions are always directed to achieve a goal.
- The basic abilities of an intelligent agent are to exist to be self-governed, responsive, goal-oriented, etc.
- In case of intelligent agents, the software modules are responsible for exhibiting intelligence. Generally observed capabilities of an intelligent agent can be given as follows :
  - Ability to remain autonomous (Self-directed)

- Responsive
- Goal-Oriented

– Intelligent agent is the one which can take input from the environment through its sensors and act upon the environment through its actuators. Its actions are always directed to achieve a goal.

Syllabus Topic : Structure of Agents

1.6.3.1 Structure of Intelligent Agents

– Fig. 1.6.6 shows the general structure of an intelligent agent.

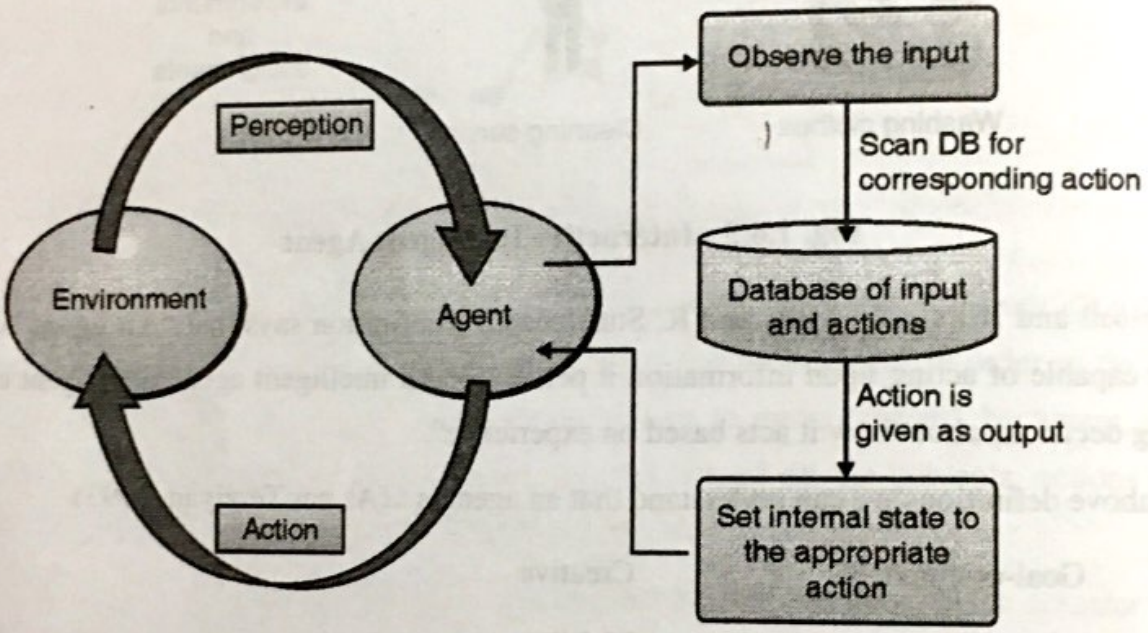


Fig. 1.6.6 : General Structure of Intelligent Agent

– From Fig. 1.6.6 it can be observed how agent and environment interact with each other. Every time environment changes the agent first observes the environment through its sensors and get the input, then scans the database of input and actions for the corresponding action for given input and lastly sets the internal state to the appropriate action.

– Let's understand this working with a real life example. Consider you are an agent and your surroundings is an environment. Now, take a situation where you are cooking in kitchen and by mistake you touch a hot pan. We will see what happens in this situation step by step. Your touch sensors take input from environment (i.e. you have touched some hot element), then it asks your brain if it knows "what action should be taken when you go near hot elements?" Now the brain will inform your hands (actuators) that you should immediately take it away from the hot element otherwise it will burn. Once this signal reaches your hand you will take your hand away from the hot pan.

- The agent keeps taking input from the environment and goes through these states every time. In above example, if your action takes more time then in that case your hand will be burnt.
- So the new task will be to find solution if the hand is burnt. Now, you think about the states which will be followed in this situation. As per Wooldridge and Jennings, "An intelligent agent is one that is capable of taking flexible self-governed actions".
- They say for an intelligent agent to meet design objectives, flexible means three things :

1. Reactiveness
2. Pro-activeness
3. Social ability

➜ 1. Reactiveness

It means giving reaction to a situation in a stipulated time frame. An agent can perceive the environment and respond to the situation in a particular time frame. In case of reactiveness, reaction within situation time frame is more important. You can understand this with above example, where, if an agent takes more time to take his hand away from the hot pan then agents hand will be burnt.

➜ 2. Pro-activeness

It is controlling a situation rather than just responding to it. Intelligent agent show goal-directed behavior by taking the initiative. For example : If you are playing chess then winning the game is the main objective. So here we try to control a situation rather than just responding to one-one action which means that killing or losing any of the 16 pieces is not important, whether that action can be helpful to checkmate your opponent is more important.

➜ 3. Social ability

Intelligent agents can interact with other agents (also humans). Take automatic car driver example, where agent might have to interact with other agent or a human being while driving the car.

- Following are few more features of an intelligent agent :

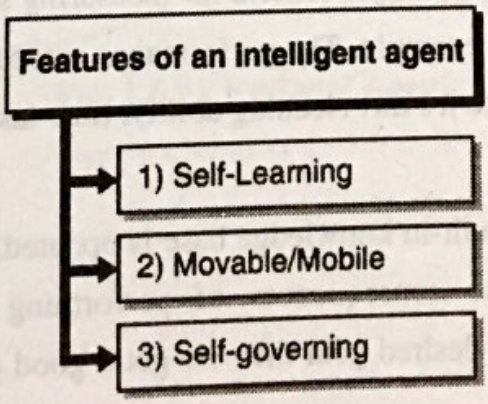


Fig. 1.6.7

➜ **1. Self-Learning**

An intelligent agent changes its behaviour based on its previous experience. This agent keeps updating its knowledge base all the time.

➜ **2. Movable/Mobile**

An Intelligent agent can move from one machine to another while performing actions.

➜ **3. Self-governing**

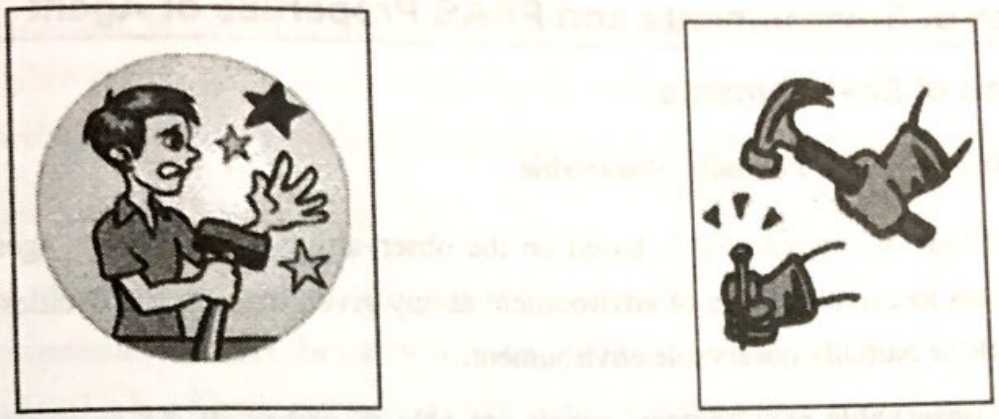
An Intelligent agent has control over its own actions.

1.6.4 Rational Agent

- For problem solving, if an agent makes a decision based on some logical reasoning, then, the decision is called as a "Rational Decision". The way humans have ability to make right decisions, based on his/her experience and logical reasoning; an agent should also be able to make correct decisions, based on what it knows from the percept sequence and actions which are carried out by that agent from its knowledge.
- Agents perceive their environment through sensors over a prolonged time period and adapt to change to create and pursue goals and take actions through actuators to achieve those goals. A **rational agent** is the one that does "right" things and acts rationally so as to achieve the best outcome even when there is uncertainty in knowledge.
- A rational agent is an agent that has clear preferences, can model uncertainty via expected values of variables or functions of variables, and always chooses to perform the action with the optimal expected outcome for itself from among all feasible actions. A rational agent can be anything that makes decisions, typically a person, a machine, or software program.
- **Rationality** depends on four main criteria: First is the performance measure which defines the criterion of success for an agent, second is the agent's prior knowledge of the environment, and third is the action performed by the agent and the last one is agent's percept sequence to date.
- Performance measure is one of the major criteria for measuring success of an agent's performance. Take a vacuum-cleaner agent's example. The performance measure of a vacuum-cleaner agent can depend upon various factors like it's dirt cleaning ability, time taken to clean that dirt, consumption of electricity, etc.

For every percept sequence a built-in knowledge base is updated, which is very useful for decision making, because it stores the consequences of performing some particular action. If the consequences direct to achieve desired goal then we get a good performance measure factor, else,

if the consequences do not lead to desired goal state, then we get a poor performance measure factor.



(a) Agent's finger is hurt while using Nail and hammer

(b) Agent is using Nail and hammer efficiently

Fig. 1.6.8

- **For example**, see Fig. 1.6.8. If agent hurts his finger while using nail and hammer, then, while using it for the next time agent will be more careful and the probability of not getting hurt will increase. In short agent will be able to use the hammer and nail more efficiently.
- Rational agent can be defined as an agent who makes use of its percept sequence, experience and knowledge to maximize the performance measure of an agent for every probable action. It selects the most feasible action which will lead to the expected results optimally.

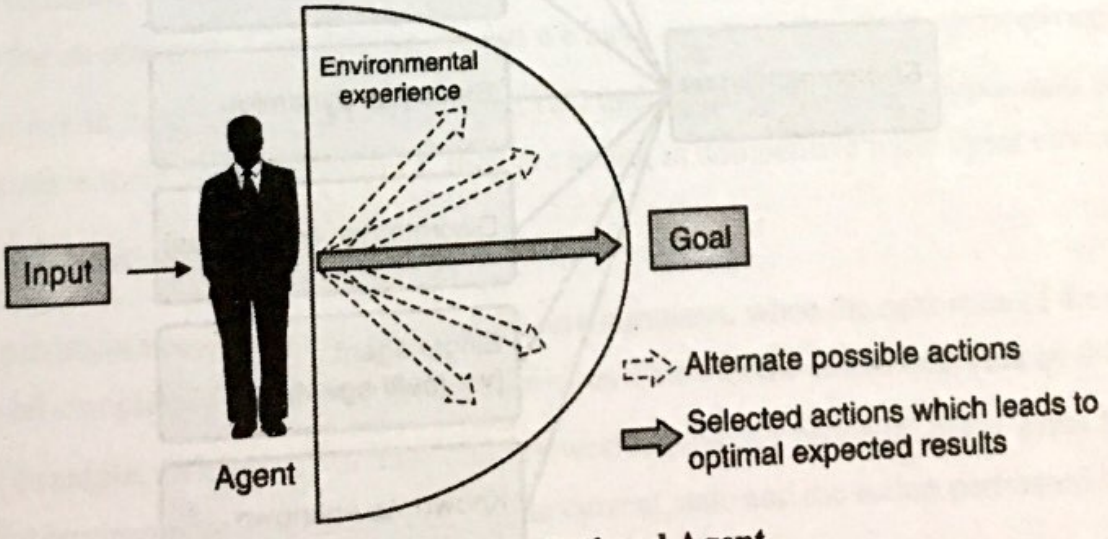


Fig. 1.6.9 : Rational Agent

1.7 Natures of Environments and PEAS Properties of Agent

1.7.1 Natures of Environments

1. Fully observable vs. Partially observable

- The first type of environment is based on the observability. Whether the agent sensors can have access to complete state of environment at any given time or not, decides if it is a fully observable or partially observable environment.
- In **Fully observable** environments agents are able to gather all the necessary information required to take actions. Also in case of fully observable environments agents don't have to keep records of internal states. For example, Word-block problem, 8-puzzle problem, Sudoku puzzle, etc. in all these problem worlds, the state is completely visible at any point of time.

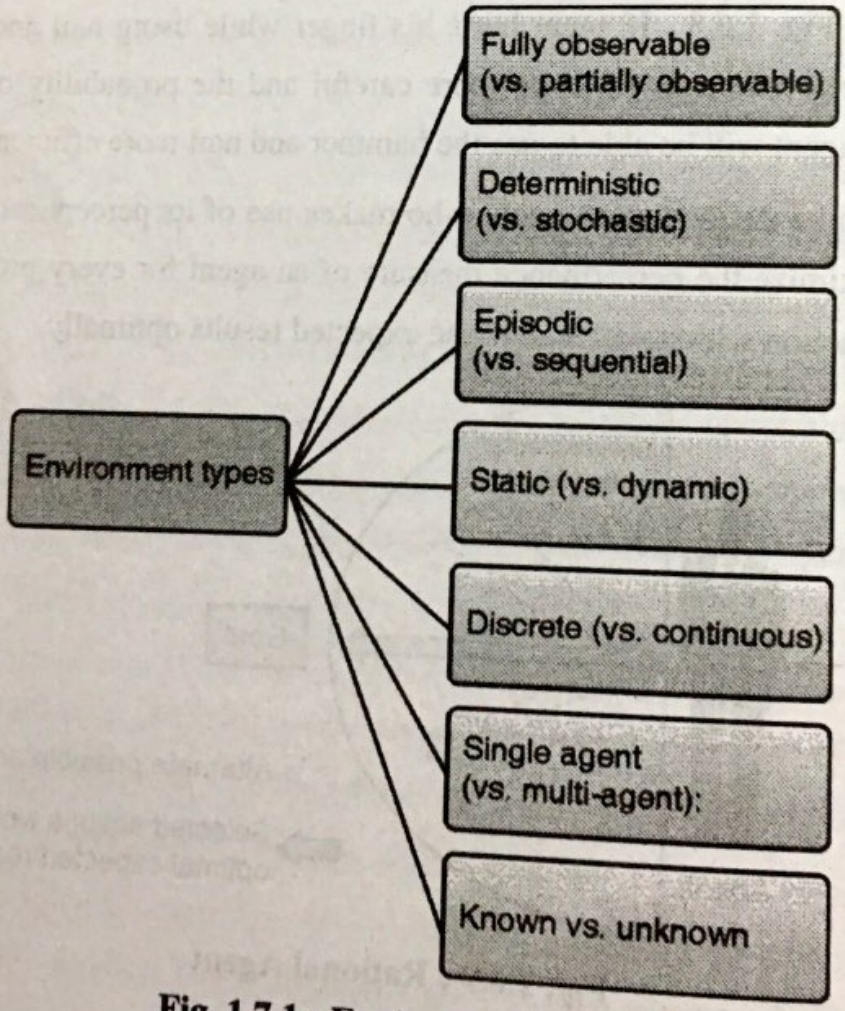


Fig. 1.7.1 : Environment types

- Environments are called **partially observable** when sensors cannot provide errorless information at any given time for every internal state, as the environment is not seen completely at any point of time.

- Also there can be unobservable environments where the agent sensors fail to provide information about internal states.
- For example, In case of an **automated car driver system**, automated car cannot predict what the other drivers are thinking while driving cars. Only because of the sensor's information gathering expertise it is possible for an automated car driver to take the actions.

➜ **2. Single agent vs. Multi-agent**

- The second type of an environment is based on the number of agents acting in the environment. Whether the agent is operating on its own or in collaboration with other agents decides if it is a **Single agent** or a multi-agent environment.
- For example : An agent playing Tetris by itself can be a single agent environment, whereas we can have an agent playing checkers in a two-agent environment. Or in case of vacuum cleaner world, only one machine is working, so it's a single agent while in case of car driving agent, there are multiple agents driving on the road, hence it's a **multi-agent environment**.
- Multi-agent environment is further classified as Co-operative multi-agent and Competitive multi-agent. Now, you might be thinking in case of an automated car driver system which type of agent environment do we have?
- Let's understand it with the help of an automated car driving example. For a car driving system 'X', other car say 'Y' is considered as an Agent. When 'Y' tries to maximize its performance measure and the input taken by car 'Y' depends on the car 'X'. Thus it can be said that for an automated car driving system we have a cooperative multi-agent environment.
- Whereas in case of "chess game" when two agents are operating as opponents, and trying to maximize their own performance, they are acting in competitive multi agent environment.

➜ **3. Deterministic vs. Stochastic**

- An environment is called **deterministic environment**, when the next state of the environment can be completely determined by the previous state and the action executed by the agent.
- For example, in case of vacuum cleaner world, 8-puzzle problem, chess game the next state of the environment solely depends on the current state and the action performed by agent.
- **Stochastic environment** generally means that the indecision about the actions is enumerated in terms of probabilities. That means environment changes while agent is taking action, hence the next state of the world does not merely depends on the current state and agent's action.
- And there are few changes happening in the environment irrespective of the agent's action. An automated car driving system has a stochastic environment as the agent cannot control the traffic conditions on the road.

- In case of checkers we have a multi-agent environment where an agent might be unable to predict the action of the other player. In such cases if we have partially observable environment then the environment is considered to be stochastic.
- If the environment is deterministic except for the actions of other agents, then the environment is **strategic.** That is, in case of game like chess, the next state of environment does not only depend upon the current action of agent but it is also influenced by the strategy developed by both the opponents for future moves.
- We have one more type of environment in this category. That is when the environment type are not fully observable or non-deterministic; such type of environment is called as **uncertain environment.**

4. Episodic vs. Sequential

- An **episodic task environment** is the one where each of the agent's action is divided into an atomic incidents or episodes. The current incident is different than the previous incident and there is no dependency between the current and the previous incident. In each incident the agent receives an input from environment and then performs a corresponding action.
- Generally, classification tasks are considered as episodic. Consider an example of pick and place robot agent, which is used to detect defective parts from the conveyor belt of an assembly line. Here, every time agent will make the decision based on current part, there will not be any dependency between the current and previous decision.
- In **sequential environments**, as per the name suggests, the previous decision can affect all future decisions. The next action of the agent depends on what action he has taken previously and what action he is supposed to take in future.
- For example, in checkers where previous move can affect all the following moves. Also sequential environment can be understood with the help of an automatic car driving example where, current decision can affect the next decisions. If agent is initiating breaks, then he has to press clutch and lower down the gear as next consequent actions.

5. Static vs. Dynamic

- You have learnt about static and dynamic terms in previous semesters with respect to web pages. Same way we have **static (vs. dynamic) environments.** If an environment remains unchanged while the agent is performing given tasks then it is called as a static environment. For example, Sudoku puzzle or vacuum cleaner environment are static in nature.

- If environment is not changing over the time but, an agent's performance is changing then, it is called as **a semi-dynamic** environment. That means, there is a timer exist in the environment who is affecting the performance of the agent.
- For example, In chess game or any puzzle like block word problem or 8-puzzle if we introduce timer, and if agent's performance is calculated by time taken to play the move or to solve the puzzle, then it is called as semi-dynamic environment.
- Lastly, if the environment changes while an agent is performing some task, then it is called **dynamic environment**.
- In this type of environment agent's sensors have to continuously keep sending signals to agent about the current state of the environment so that appropriate action can be taken with immediate effect.
- Automatic car driver example comes under dynamic environment as the environment keeps changing all the time.

➜ **6. Discrete vs. Continuous**

- You have seen discrete and continuous signals in old semesters. When you have distinct, quantized, clearly defined values of a signal it is considered as discrete signal.
- Same way, when there are distinct and clearly defined inputs and outputs or precepts and actions, then it is called a **discrete environment**. For example : chess environment has a finite number of distinct inputs and actions.
- When a continuous input signal is received by an agent, all the precepts and actions cannot be defined beforehand then it is called **continuous environment**. **For example :** An automatic car driving system.

➜ **7. Known vs. Unknown**

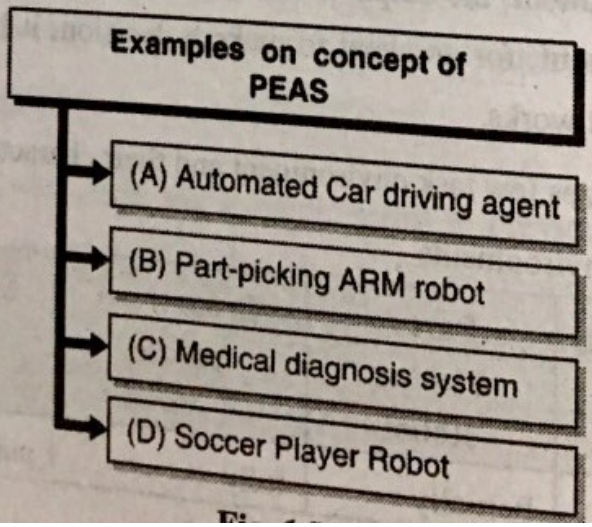
- In a **known environment**, the output for all probable actions is given. Obviously, in case of **unknown environment,** for an agent to make a decision, it has to gain knowledge about - how the environment works.
- Table 1.7.1 summarizes few task environment and their characteristics.
- Table 1.7.1 : Task Environments

Task environment	Car driving	Part – Picking Robot	Cross word puzzle	Soccer game	Checkers with clock
Observable	Partially	Partially	fully	partially	Fully

Task environment	Car driving	Part – Picking Robot	Cross word puzzle	Soccer game	Checkers with clock
Agents	Multi agent (cooperative)	Single agent	single	Multi agent (competitive)	Multi agent (competitive)
Deterministic	Stochastic	Stochastic	deterministic	Strategic	Strategic
Episodic	Sequential	Episodic	sequential	sequential	Sequential
Static	Dynamic	Dynamic	static	Dynamic	Semi
Discrete	Continuous	Discrete	Discrete	Continuous	Discrete
Known and Unknown	Unknown	Known	Known	Known	Known

1.7.2 PEAS Properties of Agent

- **PEAS :** PEAS stands for **Performance Measure, Environment, Actuators, and Sensors**. It is the short form used for performance issues grouped under Task Environment.
- You might have seen driverless/ self driving car videos of Audi/ Volvo/ Mercedes, etc. To develop such driverless cars we need to first define PEAS parameters.
- **Performance Measure :** It the objective function to judge the performance of the agent. For example, in case of pick and place robot, number of correct parts in a bin can be the performance measure.
- **Environment :** It the real environment where the agent need to deliberate actions.
- **Actuators :** These are the tools, equipment or organs using which agent performs actions in the environment. This works as the output of the agent.
- **Sensors :** These are the tools, equipment or organs using which agent captures the state of the environment. This works as the input to the agent.
- To understand the concept of PEAS, consider following examples.



➜ **(A) Automated Car driving agent**

1. **Performance measures** which should be satisfied by the automated car driver :
   - (i) **Safety :** Automated system should be able to drive the car safely without dashing anywhere.
   - (ii) **Optimum speed :** Automated system should be able to maintain the optimal speed depending upon the surroundings.
   - (iii) **Comfortable journey :** Automated system should be able to give a comfortable journey to the end user, i.e. depending upon the road it should ensure the comfort of the end user.
   - (iv) **Maximize profits :** Automated system should provide good mileage on various roads, the amount of energy consumed to automate the system should not be very high, etc. such features ensure that the user is benefited with the automated features of the system and it can be useful for maximizing the profits.
2. **Environment**
   - (i) **Roads :** Automated car driver should be able to drive on any kind of a road ranging from city roads to highway.
   - (ii) **Traffic conditions :** You will find different set of traffic conditions for different type of roads. Automated system should be able to drive efficiently in all types of traffic conditions. Sometimes traffic conditions are formed because of pedestrians, animals, etc.
   - (iii) **Clients :** Automated cars are created depending on the client's environment. For example, in some countries you will see left hand drive and in some countries there is a right hand drive. Every country/state can have different weather conditions. Depending upon such constraints automated car driver should be designed.
3. **Actuators** are responsible for performing actions/providing output to an environment.

   In case of car driving agent following are the actuators :
   - (i) **Steering wheel** which can be used to direct car in desired direction (i.e. right/left)
   - (ii) **Aaccelerator, gear**, etc. can be useful to increase or decrease the speed of the car.
   - (iii) **Brake** is used to stop the car.
   - (iv) **Light signal, horn** can be very useful as indicators for an automated car.
4. **Sensors :** To take input from environment in car driving example cameras, sonar system, speedometer, GPS, engine sensors, etc. are used as sensors.

➜ **(B) Part-picking ARM robot**

1. **Performance measures** : Number of parts in correct container.
2. **Environment :** Conveyor belt used for handling parts, containers used to keep parts, and Parts.

3. **Actuators** : Arm with tooltips, to pick and drop parts from one place to another.
4. **Sensors** : Camera to scan the position from where part should be picked and joint angle sensors which are used to sense the obstacles and move in appropriate place.

➜ **(C) Medical diagnosis system**

1. **Performance Measures**

   Healthy patient : system should make use of sterilized instruments to ensure the safety (healthiness) of the patient,

   Minimize costs : the automated system results should not be very costly otherwise overall expenses of the patient may increase, Lawsuits. Medical diagnosis system should be legal.
2. **Environment :** Patient, Doctors, Hospital Environment
3. **Sensors :** Screen, printer
4. **Actuators :** Keyboard and mouse which is useful to make entry of symptoms, findings, patient's answers to given questions. Scanner to scan the reports, camera to click pictures of patients.

➜ **(D) Soccer Player Robot**

1. **Performance Measures :** Number of goals, speed, legal game.
2. **Environment :** Team players, opponent team players, playing ground, goal net.
3. **Sensors :** Camera, proximity sensors, infrared sensors.
4. **Actuators :** Joint angles, motors.

1.8 Types of Agents

Depending upon the degree of intelligence and ability to achieve the goal, agents are categorized into five basic types. These five types of agents are depicted in the Fig. 1.8.1.

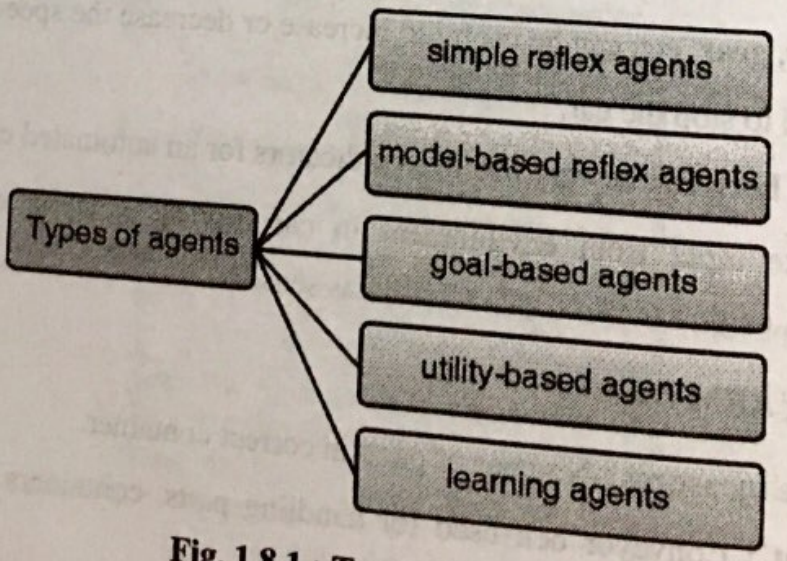


Fig. 1.8.1 : Types of agents

Let us understand these agent types one by one.

1.8.1 Simple Reflex Agents

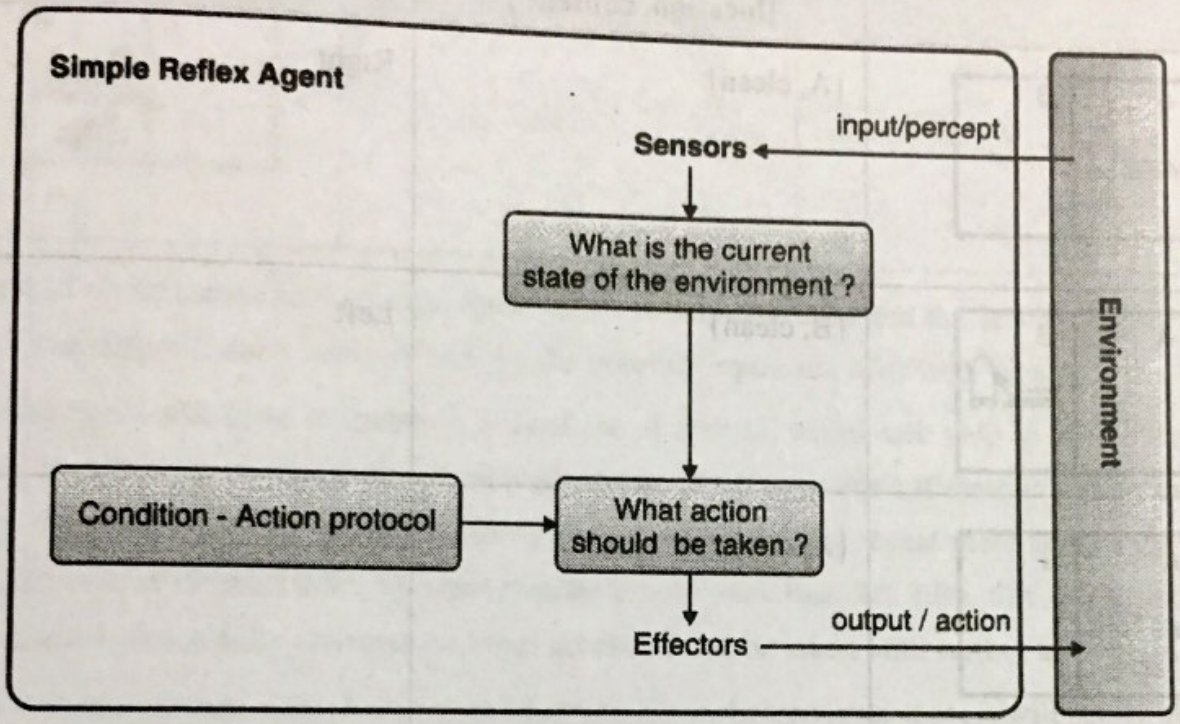


Fig. 1.8.2 : Simple reflex agents

- An agent which performs actions based on the current input only, by ignoring all the previous inputs is called as **simple reflex agent**.
- It is a totally uncomplicated type of agent. The simple reflex agent's function is based on the situation and its corresponding action (condition-action protocol). If the condition is true, then matching action is taken without considering the percept history.
- You can understand simple reflexes with the help of a real life example, say some object approaches eye then, you will blink your eye. This type of simple reflex is called natural/innate reflex.

Consider the example of the vacuum cleaner agent. It is a simple reflex agent, as its decision is based only on whether the current location contains dirt. The agent function is tabulated in Table 1.8.1.

Few possible input sequences and outputs for vacuum cleaner world with 2 locations are considered for simplicity.

Table 1.8.1

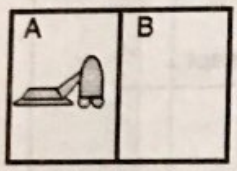
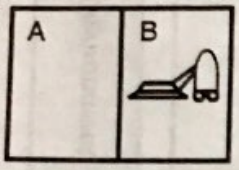
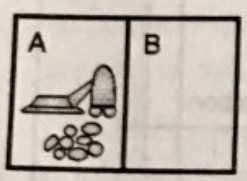
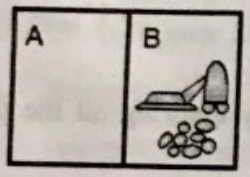
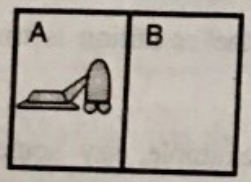
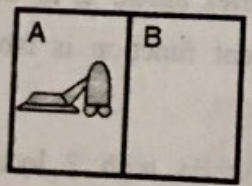
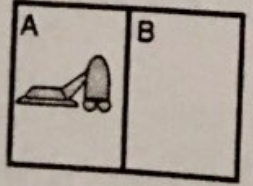
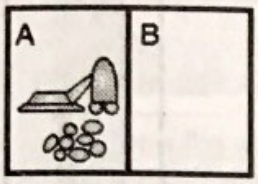
Figure	Input sequence {location, content }	Output / action Right, left, suck, no-op
	{A, clean}	Right
	{B, clean}	Left
	{A, dirt}	Suck
	{B, dirt}	Suck
	Input sequence {location, content }	Output / action Right, left, suck, no-op
	{A, clean}{A, clean}	Right
	{A, clean}{A, dirt}	Suck

Figure	Input sequence {location, content }	Output / action Right, left, suck, no-op
	: : :	: :

- In case of above mentioned vacuum agent only one sensor is used and that is a dirt sensor. This dirt sensor can detect if there is dirt or not. So the possible inputs are 'dirt' and 'clean'.
- Also the agent will have to maintain a database of actions, which will help to decide what output should be given by an agent. Database will contain conditions like : If there is dirt on the floor to left or right then find out if there is dirt in the next location and repeat these actions till the entire assigned area is cleaned then, vacuum cleaner should suck that dirt. Else, dirt should move. Once the assigned area is fully covered, no other action should be taken until further instruction.
- If the vacuum cleaner agent keeps searching for dirt and clean area, then, it will surely get trapped in an infinite loop. Infinite loops are unavoidable for simple reflex agents operating in partially observable environments. By randomizing its actions the simple reflex agent can avoid these infinite loops. For example, on receiving {clean} as input, the vacuum cleaner agent should either go to left or right direction.
- If the performance of an agent is of the right kind then randomized behaviour can be considered as rational in few multi-agent environments.

1.8.2 Model-based Reflex Agents

- Partially observable environment cannot be handled well by simple reflex agents because it does not keep track on the previous state. So, one more type of agent was created that is model based reflex agent.
- An agent which performs actions based on the current input and one previous input is called as model-based agent. Partially observable environment can be handled well by model-based agent.
- From Fig. 1.8.3 it can be seen that once the sensor takes input from the environment, agent checks for the current state of the environment. After that, it checks for the previous state which shows how the world is developing and how the environment is affected by the action which was taken by the agent at earlier stage. This is termed as model of the world.

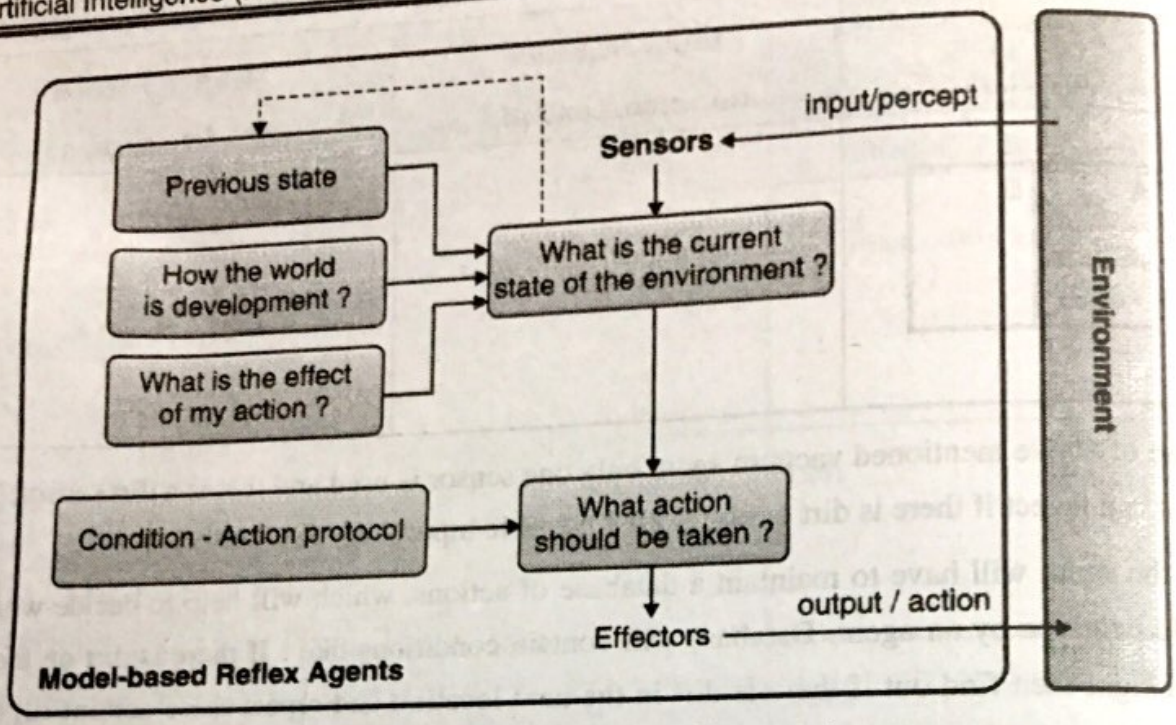


Fig. 1.8.3 : Model-based reflex agents

Once this is verified, based on the condition-action protocol an action is decided. This decision is given to effectors and the effectors give this output to the environment.

The knowledge about "how the world is changing" is called as a model of the world. Agent which uses such model while working is called as the **"model-based agent".**

Consider a simple example of automated car driver system. Here, the world keeps changing all the time. You must have taken a wrong turn while driving on some or the other day of your life. Same thing applies for an agent. Suppose if some car "X" is overtaking our automated driver agent "A", then speed and direction in which "X" and "A" are moving their steering wheels is important. Take a scenario where agent missed a sign board as it was overtaking other car. The world around that agent will be different in that case.

Internal model based on the input history should be maintained by model-based reflex agent, which can reflect at least some of the unobserved aspects of the current state. Once this is done it chooses an action in the same way as the simple reflex agent.

1.8.3 Goal-based Agents

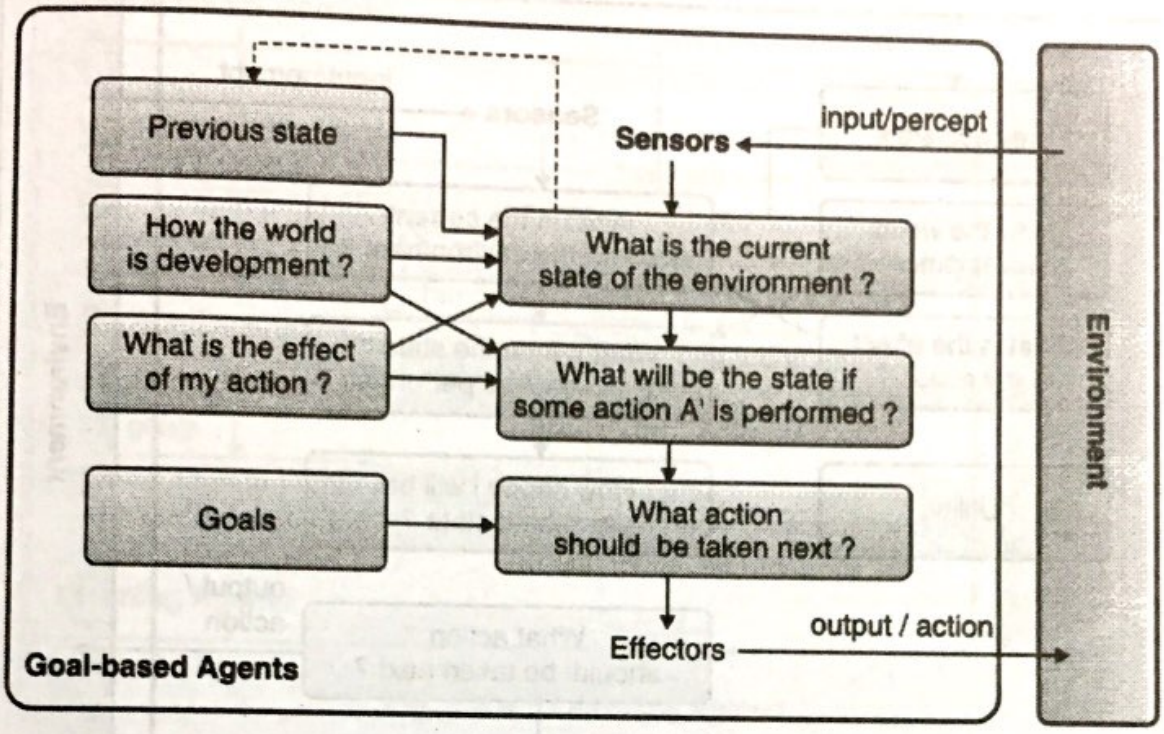


Fig. 1.8.4 : Goal-based agents

- Model-based agents are further developed based on the "goal" information. This new type of agent is called as goal-based agent. As the name suggests, Goal information will illustrate the situations that is desired. These agents are provided with goals along with the model of the world. All the actions selected by the agent are with reference of the specified goals. Goal based agents can only differentiate between goal states and non-goal states. Hence, their performance can be 100% or zero.
- The limitation of goal based agent comes with its definition itself. Once the goal is fixed, all the actions are taken to fulfil it. And the agent loses flexibility to change its actions according to the current situation.
- You can take example of a vacuum cleaning robot agent whose goal is to keep the house clean all the time. This agent will keep searching for dirt in house and will keep the house clean all the time. Remember M-O the cleaning robot from Wall-E movie which keeps cleaning all the time no matter what is the environment or the Healthcare companion robot Baymax from Big Hero 6 which does not deactivate until user says that he/she is satisfied with care.

1.8.4 Utility-Based Agents

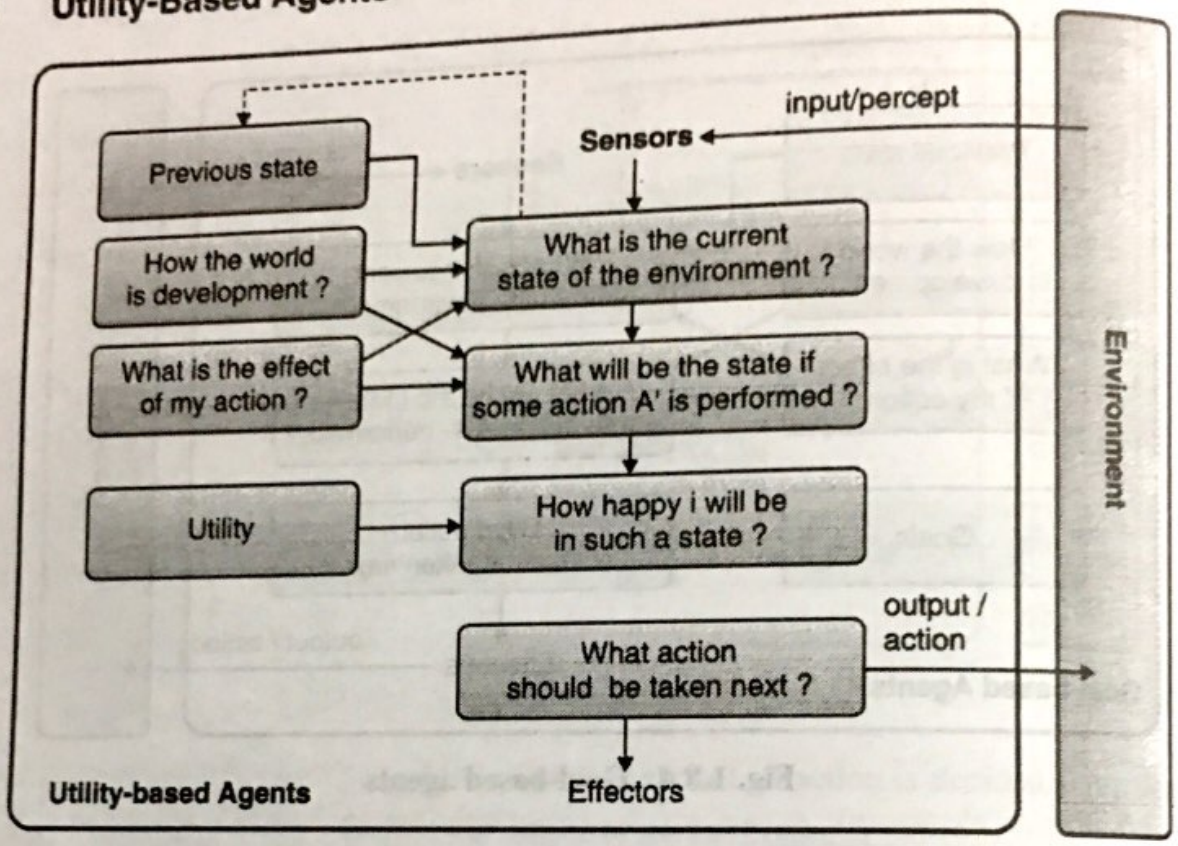


Fig. 1.8.5 : Utility-based agents

- Utility function is used to map a state to a measure of utility of that state. We can define a measu for determining how advantageous a particular state is for an agent. To obtain this measure utili function can be used.
- The term utility is used to depict how "happy" the agent is to find out a generalized performanc measure, various world states according to exactly how happy they would make an agent compared.
- Take one example; you might have used Google maps to find out a route which can take you fro source location to your destination location in least possible time. Same logic is followed by utili based automatic car driving agent.
- Goals utility based automatic car driving agent can be used to reach given location safely wit least possible time and save fuel. So this car driving agent will check the possible routes and th traffic conditions on these routes and will select the route which can take the car at destination least possible time safely and without consuming much fuel.

1.8.5 Learning Agents

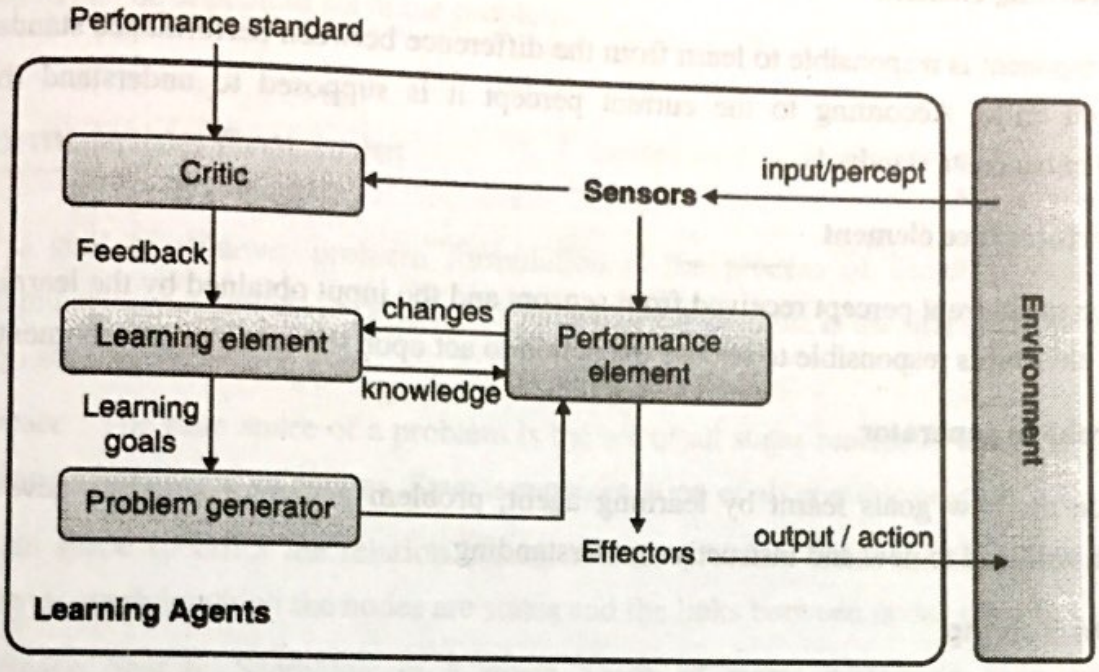


Fig. 1.8.6 : Learning agents

- Why do you give mock tests ? When you get less marks for some question, you come to know that you have made some mistake in your answer. Then you learn the correct answer and when you get that same question in further examinations, you write the correct answer and avoid the mistakes which were made in the mock test. This same concept is followed by the learning agent.
- Learning based agent is advantageous in many cases, because with its basic knowledge it can initially operate in an unknown environment and then it can gain knowledge from the environment based on few parameters and perform actions to give better results.
- Following are the components of learning agent :

1. Critic
2. Learning element
3. Performance element
4. Problem generator

➜ **1. Critic**

It is the one who compares sensor's input specifying effect of agent's action on the environment with the performance standards and generate feedback for leaning element.

➜ **2. Learning element**

This component is responsible to learn from the difference between performance standards and the feedback from critic. According to the current percept it is supposed to understand the expected behavior and enhance its standards

➜ **3. Performance element**

Based on the current percept received from sensors and the input obtained by the learning element, performance element is responsible to choose the action to act upon the external environment.

➜ **4. Problem generator**

Based on the new goals learnt by learning agent, problem generator suggests new or alternate actions which will lead to new and instructive understanding.

☞ **Problem Solving**

- Search is an indivisible part of intelligence. An intelligent agent is the one who can search and select the most appropriate action in the given situation, among the available set of actions. When we play any game like chess, cards, tic-tac-toe, etc.; we know that we have multiple options for next move, but the intelligent one who searches for the correct move will definitely win the game.
- In case of travelling salesman problem, medical diagnosis system or any expert system; all they required to do is to carry out search which will produce the optimal path, the shortest path with minimum cost and efforts. Hence, this chapter focuses on the searching techniques used in AI applications. Those are known as un-informed and informed search techniques.

Syllabus Topic : Solving Problems by Searching Solutions

1.9 Searching for Solution

- Now let us see how searching play a vital role in solving AI problems. Given a probiem, we can generate all the possible states it can have in real time, including start state and end state. To generate solution for the same is nothing but searching a path from start state to end state.
- Problem solving agent is the one who finds the goal state from start state in optimal way by following the shortest path, thereby saving the memory and time. It's supposed to maximize its performance by fulfilling all the performance measures.
- Searching techniques can be used in game playing like Tic-Tac-Toe or navigation problems like Travelling Salesman Problem.

- First, we will understand the representation of given problem so that appropriate searching techniques can be applied to solve the problem.

1.10 Formulating Problems

- Given a goal to achieve; problem formulation is the process of deciding what states to be considered and what actions to be taken to achieve the goal. This is the first step to be taken by any problem solving agent.
- State space : The state space of a problem is the set of all states reachable from the initial state by executing any sequence of actions. State is representation of all possible outcomes.
- The state space specifies the relation among various problem states thereby, forming a directed network or graph in which the nodes are states and the links between nodes represent actions.
- State Space Search: Searching in a given space of states pertaining to a problem under consideration is called a state space search.
- Path : A path is a sequence of states connected by a sequence of actions, in a given state space.

Syllabus Topic : Problem Solving Agents

1.10.1 Problems Solving Agent

Problem formulation is the basic step of problem solving agent. Problem can be defined formally using five components as follows :

1.	Initial state	2.	Actions
3.	Successor function	4.	Goal test
5.	Path cost		

➜ **1. Initial state**

The initial state is the one in which the agent starts in.

➜ **2. Actions**

It is the set of actions that can be executed or applicable in all possible states. A description of what each action does; the formal name for this is the transition model.

➜ **3. Successor function**

It is a function that returns a state on executing an action on the current state.

4. Goal test

It is a test to determine whether the current state is a goal state. In some problems the goal test can be carried out just by comparing current state with the defined goal state, called as explicit goal test. Whereas, in some of the problems, state cannot be defined explicitly but needs to be generated by carrying out some computations, it is called as implicit goal test. For example : In Tic-Tac-Toe game making diagonal or vertical or horizontal combination declares the winning state which can be compared explicitly; but in the case of chess game, the goal state cannot be predefined but it's a scenario called as "Checkmate", which has to be evaluated implicitly.

5. Path cost

It is simply the cost associated with each step to be taken to reach to the goal state. To determine the cost to reach to each state, there is a cost function, which is chosen by the problem solving agent.

Problem solution

A well-defined problem with specification of initial state, goal test, successor function, and path cost. It can be represented as a data structure and used to implement a program which can search for the goal state. A solution to a problem is a sequence of actions chosen by the problem solving agent that leads from the initial state to a goal state. Solution quality is measured by the path cost function.

Optimal solution

An optimal solution is the solution with least path cost among all solutions. A general sequence followed by a simple problem solving agent is, first it formulates the problem with the goal to be achieved, then it searches for a sequence of actions that would solve the problem, and then executes the actions one at a time.

Syllabus Topic : Example Problems

1.10.2 Example Problems

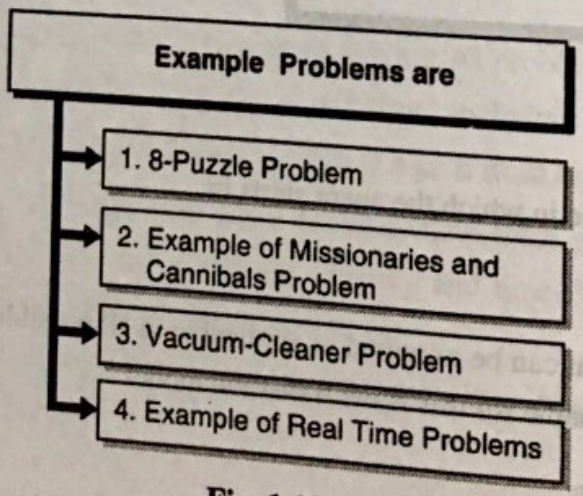


Fig. 1.10.1

1. 8-Puzzle Problem

Fig. 1.10.2 depicts a typical scenario of 8-puzzle problem. It has a 3 × 3 board with tiles having 1 through 8 numbers on it. There is a blank tile which can be moved forward, backward, to left and to right. The aim is to arrange all the tiles in the goal state form by moving the blank tile minimum number of times.

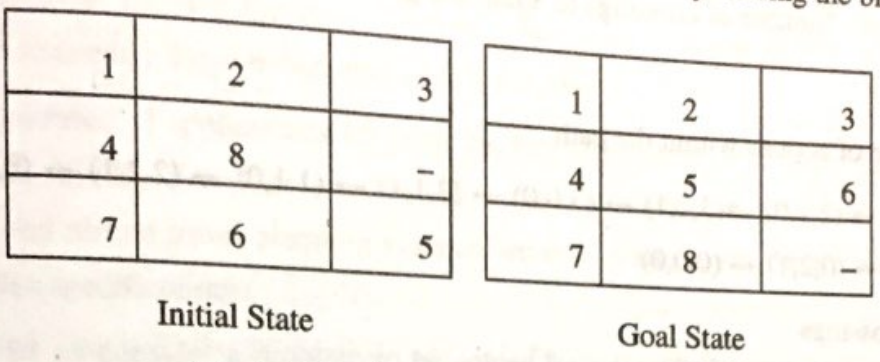


Fig. 1.10.2 : A scenario of 8-Puzzle Problem

This problem can be formulated as follows :

States : States can represented by a 3 x 3 matrix data structure with blank denoted by 0.

1. **Initial state :** {{1, 2, 3},{4, 8, 0},{7, 6, 5}}
2. **Actions :** The blank space can move in Left, Right, Up and Down directions specifying the actions.
3. **Successor function :** If we apply "Down" operator to the start state in Fig. 1.10.2, the resulting state has the 5 and the blank switching their positions.
4. **Goal test :** {{1, 2, 3},{4, 5, 6},{7, 8, 0}}
5. **Path cost :** Number of steps to reach to the final state.

Solution :

{{1, 2, 3}, {4, 8, 0}, {7, 6, 5}} → {{1, 2, 3}, {4, 8, 5}, {7, 6, 0}} → {{1, 2, 3}, {4, 8, 5}, {7, 0, 6}} → {{1, 2, 3}, {4, 0, 5}, {7, 8, 6}} → {{1, 2, 3}, {4, 5, 0}, {7, 8, 6}} → {{1, 2, 3}, {4, 5, 6}, {7, 8, 0}}

Path cost = 5 steps

2. Example of Missionaries and Cannibals Problem

The problem statement as discussed in the previous section. Let's formulate the problem first.

States : In this problem, state can be data structure having triplet (i, j, k) representing the number of missionaries, cannibals, and canoes on the left bank of the river respectively.

1. Initial state : It is (3, 3, 1), as all missionaries, cannibals and canoes are on the left bank of the river.

2. Actions : Take x number of missionaries and y number of cannibals
3. Successor function : If we take one missionary, one cannibal the other side of the river will have two missionaries and two cannibals left.
4. Goal test : Reached state (0, 0, 0)
5. Path cost : Number of crossings to attain the goal state.

Solution :

The sequence of actions within the path :

$(3,3,1) \rightarrow (2,2,0) \rightarrow (3,2,1) \rightarrow (3,0,0) \rightarrow (3,1,1) \rightarrow (1,1,0) \rightarrow (2,2,1) \rightarrow (0,2,0) \rightarrow (0,3,1) \rightarrow$
$(0,1,0) \rightarrow (0,2,1) \rightarrow (0,0,0)$

Cost = 11 crossings

3. Vacuum-Cleaner Problem

States : In vacuum cleaner problem, state can be represented as [<block>, clean] or [<block>, dirty]. The agent can be in one of the two blocks which can be either clean or dirty. Hence there are total 8 states in the vacuum cleaner world.

1. Initial State : Any state can be considered as initial state. For example, [A, dirty]
2. Actions : The possible actions for the vacuum cleaner machine are left, right, absorb, idle.
3. Successor function : Fig. 1.10.3 indicating all possible states with actions and the next state.

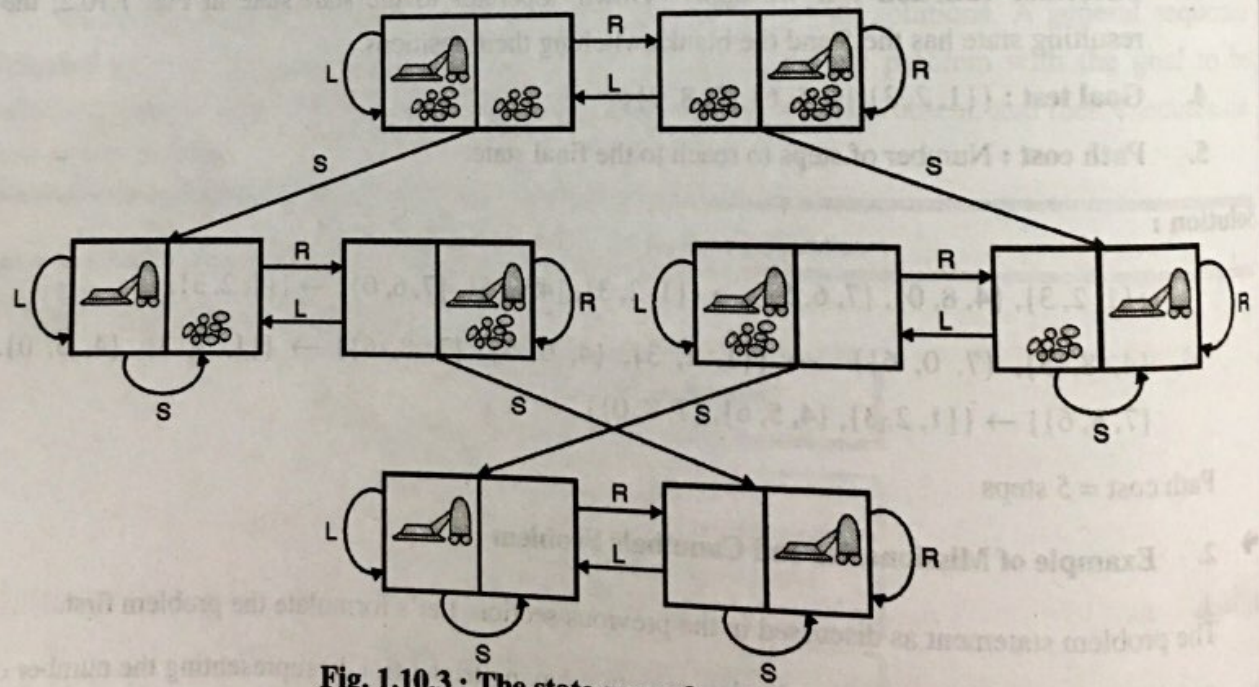


Fig. 1.10.3 : The state space for vacuum world

4. Goal Test : The aim of the vacuum cleaner is to clean both the blocks. Hence the goal test if [A, Clean] and [B, Clean].

5. Path Cost : Assuming that each action/ step costs 1 unit cost. The path cost is number of actions/ steps taken.

4. Example of Real Time Problems

- There are varieties of real time problems that can be formulated and solved by searching. Robot Navigation, Rout Finding Problem, Traveling Salesman Problem (TSP), VLSI design problem, Automatic Assembly Sequencing, etc. are few to name.
- There are number of applications for route finding algorithms. Web sites, car navigation systems that provide driving directions, routing video streams in computer networks, military operations planning, and airline travel-planning systems are few to name. All these systems involve detailed and complex specifications.
- For now, let us consider a problem to be solved by a travel planning web site; the airline travel problem.

State

State is represented by airport location and current date and time. In order to calculate the path cost state may also record more information about previous segments of flights, their fare bases and their status as domestic or international.

1. Initial state : This is specified by the user's query, stating initial location, date and time.
2. Actions : Take any flight from the current location, select seat and class, leaving after the current time, leaving enough time for within airport transfer if needed.
3. Successor function : After taking the action i.e. selecting fight, location, date, time; what is the next location date and time reached is denoted by the successor function. The location reached is considered as the current location and the flight's arrival time as the current time.
4. Goal test : Is the current location the destination location?
5. Path cost : In this case path cost is a function of monetary cost, waiting time, flight time, customs and immigration procedures, seat quality, time of day, type of airplane, frequent-flyer mileage awards and so on.

1.10.3 Measuring Performance of Problem Solving Algorithm / Agent

- There are variety of problem solving methods and algorithms available in AI. Before studying any of these algorithms in detail, let's consider the criteria to judge the efficiency of those algorithms. The performance of all these algorithms can be evaluated on the basis of following factors.

1. Completeness : If the algorithm is able to produce the solution if one exists then it satisfies completeness criteria.
2. Optimality : If the solution produced is the minimum cost solution, the algorithm is said to be optimal.
3. Time complexity : It depends on the time taken to generate the solution. It is the number of nodes generated during the search.
4. Space complexity : Memory required to store the generated nodes while performing the search.

- Complexity of algorithms is expressed in terms of three quantities as follows :
  1. b : Called as branching factor representing maximum number of successors a node can have in the search tree.
  2. d : Stands for depth of the shallowest goal node.
  3. m : It is the maximum depth of any path in the search tree.

1.10.4 Node Representation in Search Tree

- In order to carry out search, first we need to build the search tree. The nodes are the various possible states in the state space.
- The connectors are the indicators of which all states are directly reachable from current state, based on the successor function.
- Thus the parent child relation is build and the search tree can be generated. Fig. 1.10.4 shows the representation of a tree node as a data structure in 8-puzzle problem.

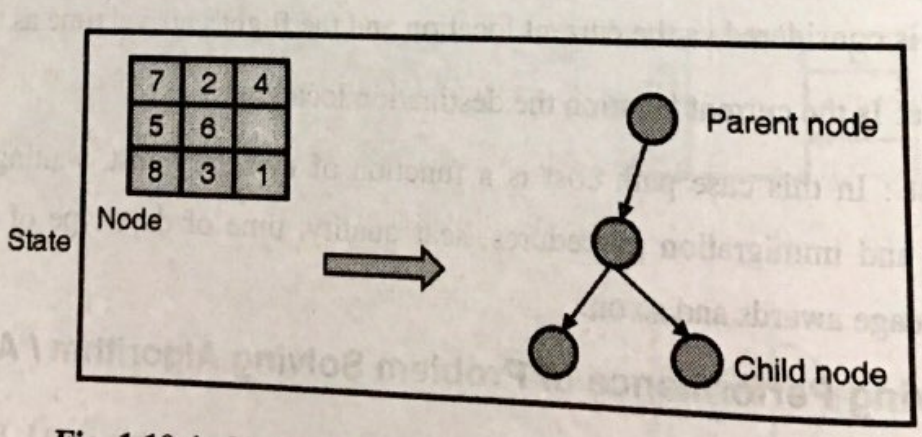


Fig. 1.10.4 : Node representation of state in searching

- Node is the data structure from which the search tree is constructed. Each node has a parent, a state, and children nodes directly reachable from that node.

- For each node of the tree, we can have following structure components :

1. State / Value : The state in the state space to which the node corresponds or value assigned to the node.
2. Parent node : The node in the search tree that generated this node.
3. Number of children : Indicating number of actions that can be taken to generate next states (children nodes).
4. Path cost : The cost of the path from the initial state to the node.

Syllabus Topic : Uninformed Search Strategies

1.11 Uninformed Search

- The term "uninformed" means they have only information about what is the start state and the end state along with the problem definition.
- These techniques can generate successor states and can distinguish a goal state from a non-goal state.
- All these search techniques are distinguished by the order in which nodes are expanded.
- The uninformed search techniques also called as "blind search".

1.11.1 Depth First Search (DFS)

1.11.1.1 Concept

- In depth-first search, the search tree is expanded depth wise; i.e. the deepest node in the current branch of the search tree as expanded. As the leaf node is reached, the search backtracks to previous node. The progress of the search is illustrated in Fig. 1.11.1.
- The explored nodes are shown in light gray. Explored nodes with no descendants in the fringe are removed from memory. Nodes at depth three have no successors and M is the only goal node.

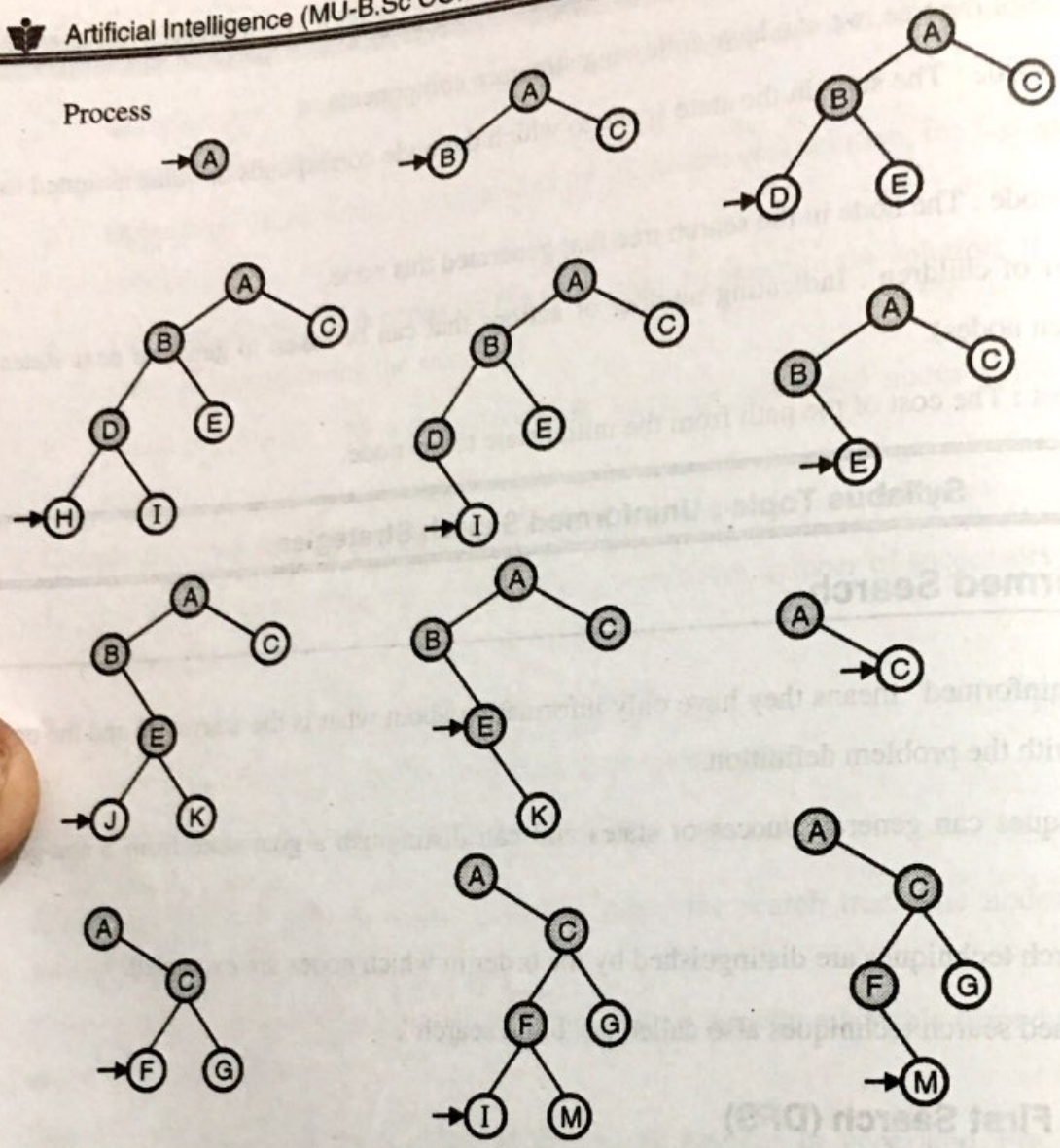


Fig. 1.11.1 : Working of Depth first search on a binary tree

1.11.1.2 Implementation

- DFS uses a LIFO fringe i.e. stack. The most recently generated node, which is on the top in the fringe, is chosen first for expansion. As the node is expanded, it is dropped from the fringe and its successors are added.
- So when there are no more successors to add to the fringe, the search "back tracks" to the next deepest node that is still unexplored. DFS can be implemented in two ways, recursive and non-recursive. Following is the algorithm for the same.

1.11.1.3 Algorithm

(a) Non recursive implementation of DFS

1. Push the root node on a stack
2. while (stack is not empty)
   a) pop a node from the stack;
   (i) if node is a goal node then return success;
   (ii) push all children of node onto the stack;
3. return failure

(b) Recursive implementation of DFS

DFS(c) :

1. If node is a goal, return success;
2. for each child c of node
   a) if DFS (c) is successful,
   (i) return success
3. return failure;

1.11.1.4 Performance Evaluation

Completeness : Complete, if m is finite.

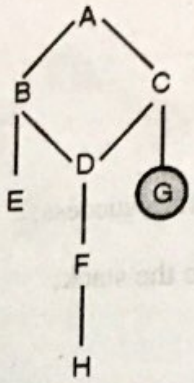
Optimality : No, as it cannot guarantee the shallowest solution.

Time Complexity : A depth first search, may generate all of the O(bm) nodes in the search tree, where m is the maximum depth of any node; this can be much greater than the size of the state space.

Space Complexity : For a search tree with branching factor b and maximum depth m, depth first search requires storage of only O(bm) nodes, as at a time only the branch, which is getting explored, will reside in memory.

Ex. 1.11.1 : Consider following graph.

Starting from state A execute DFS. The goal node is G. Show the order in which the nodes are expanded. Assume that the alphabetically smaller node is expanded first to break ties.



Soln. :

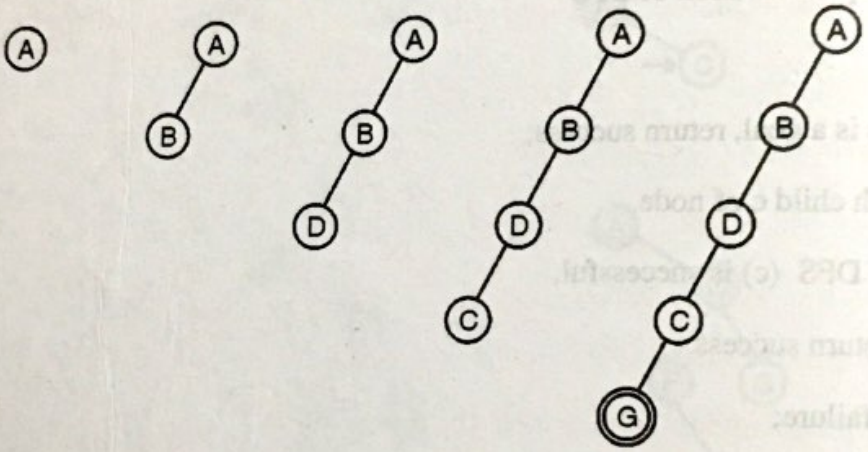


Fig. 1.12.1

1.11.2 Breadth First Search (BFS)

1.11.2.1 Concept

- As the name suggests, in breadth-first search technique, the tree is expanded breadth wise.
- The root node is expanded first, then all the successors of the root node are expanded, then their successors, and so on.
- In turn, all the nodes at a particular depth in the search tree are expanded first and then the search will proceed for the next level node expansion.
- Thus, the shallowest unexpanded node will be chosen for expansion. The search process of BFS is illustrated in Fig. 1.11.2.

1.11.2.2 Process

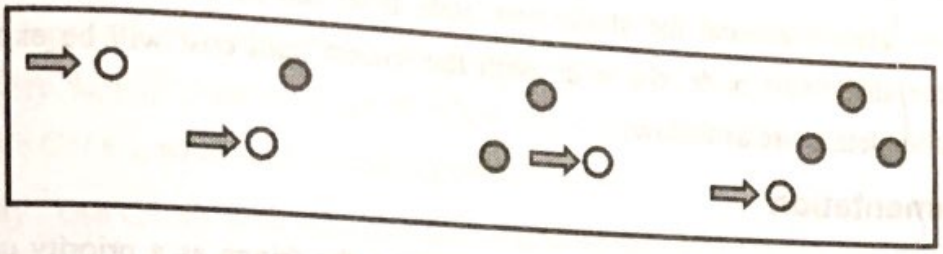


Fig. 1.11.2 : Working of BFS on binary tree

1.11.2.3 Implementation

- In BFS we use a FIFO queue for the fringe. Because of which the newly inserted nodes in the fringe will automatically be placed after their parents.
- Thus, the children nodes, which are deeper than their parents, go to the back of the queue, and old nodes, which are shallower, get expanded first. Following is the algorithm for the same.

1.11.2.4 Algorithm

1. Put the root node on a queue
2. while (queue is not empty)
   a) remove a node from the queue
      (i) if (node is a goal node) return success;
      (ii) put all children of node onto the queue;
3. return failure;

1.11.2.5 Performance Evaluation

Completeness : It is complete, provided the shallowest goal node is at some finite depth.

Optimality : It is optimal, as it always finds the shallowest solution.

Time complexity : O(bd), number of nodes in the fringe.

Space complexity : O(bd), total number of nodes explored.

1.11.3 Uniform Cost Search (UCS)

1.11.3.1 Concept

- Uniform cost search is a breadth first search with all paths having same cost. To make it work in real time conditions we can have a simple extension to the basic implementation of BFS. This results in an algorithm that is optimal with any path cost.

- In BFS as we always expand the shallowest node first; but in uniform cost search, instead of expanding the shallowest node, the node with the lowest path cost will be expanded first. The implementation details are as follow.

1.11.3.2 Implementation

- Uniform cost search can be achieved by implementing the fringe as a priority queue ordered by path cost. The algorithm shown below is almost same as BFS; except for the use of a priority queue and the addition of an extra check in case a shorter path to any node is discovered.
- The algorithm takes care of nodes which are inserted in the fringe for exploration, by using a data structure having priority queue and hash table.
- The priority queue used here contains total cost from root to the node. Uniform cost search gives the minimum path cost the maximum priority. The algorithm using this priority queue is the following.

1.11.3.3 Algorithm

- Insert the root node into the queue.
- While the queue is not empty :
  - (i) Dequeue the maximum priority node from the queue.

    (If priorities are same, alphabetically smaller node is chosen)
  - (ii) If the node is the goal node, print the path and exit.

    Else
- Insert all the children of the dequeued node, with their total costs as priority.
- The algorithm returns the best cost path which is encountered first and will never go for other possible paths. The solution path is optimal in terms of cost.
- As the priority queue is maintained on the basis of the total path cost of node, the algorithm never expands a node which has a cost greater than the cost of the shortest path in the tree.
- The nodes in the priority queue have almost the same costs at a given time, and thus the name "Uniform Cost Search".

1.11.3.4 Performance Evaluation

- Completeness : Completeness is guaranteed provided the cost of every step exceeds some small positive constant.
- Optimality : It produces optimal solution as nodes are expanded in order of their path cost.

- Time complexity : Uniform-cost search considers path costs rather than depths; so its complexity is does not merely depends on b and d. Hence we consider C* be the cost of the optimal solution, and assume that every action costs at least €. Then the algorithm's worst-case time and space complexity is O(b C*/ €), which can be much greater than bd.
- Space complexity : O(b C*/ €), indicating number of node in memory at execution time.

1.11.4 Depth Limited Search (DLS)

1.11.4.1 Concept

- In order to avoid the infinite loop condition arising in DFS, in depth limited search technique, depth-first search is carried out with a predetermined depth limit.
- The nodes with the specified depth limit are treated as if they don't have any successors. The depth limit solves the infinite-path problem.
- But as the search is carried out only till certain depth in the search tree, it introduces problem of incompleteness.
- Depth-first search can be viewed as a special case of depth-limited search with depth limit equal to the depth of the tree. The process of DLS is depicted in Fig. 1.11.3.

1.11.4.2 Process

If depth limit is fixed to 2, DLS carries out depth first search till second level in the search tree.

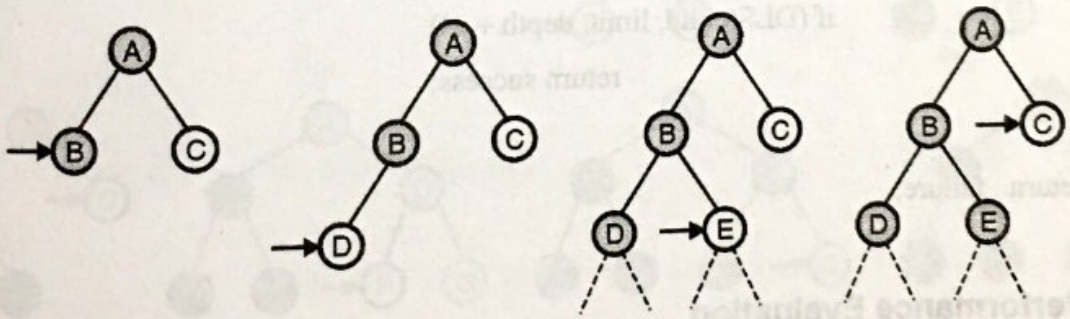


Fig. 1.11.3 : DIS working with depth limit

1.11.4.3 Implementation

- As in case of DFS in DLS we can use the same fringe implemented as queue.
- Additionally the level of each node needs to be calculated to check whether it is within the specified depth limit. Depth-limited search can terminate with two conditions :

1. If the solution is found.
2. If there is no solution within given depth limit.

1.11.4.4 Algorithm

Determine the start node and the search depth.

Check if the current node is the goal node

If not : Do nothing

If yes : return

Check if the current node is within the specified search depth

If not : Do nothing

If yes : Expand the node and save all of its successors in a stack.

Call DLS recursively for all nodes of the stack and go back to Step 2.

1.11.4.5 Pseudo Code

```
booleanDLS (Node node, int limit, int depth)
{
    if (depth > limit) return failure;
    if (node is a goal node) return success;
    for each child of node
    {
                if (DLS(child, limit, depth + 1))
                        return success;
    }
    return failure;
    }
```

1.11.4.6 Performance Evaluation

Completeness : Its incomplete if shallowest goal is beyond the depth limit.

Optimality : Non optimal, as the depth chosen can be greater than d.

Time complexity : Same as DFS, O (bl), where l is the specified depth limit.

Space complexity : Same as DFS, O(bl), where l is the specified depth limit.

1.11.5 Iterative Deepening DFS (IDDFS)

1.11.5.1 Concept

- Iterative deepening depth first search is a combination of BFS and DFS. In IDDFS search happens depth wise but, at a time the depth limit will be incremented by one. Hence iteratively it deepens down in the search tree.
- It eventually turns out to be the breadth-first search as it explores a complete layer of new nodes at each iteration before going on to the next layer.
- It does this by gradually increasing the depth limit-first 0, then 1, then 2, and so on-until a goal is found; and thus guarantees the optimal solution. Iterative deepening combines the benefits of depth-first and breadth-first search. The search process is depicted in Fig. 1.11.4.

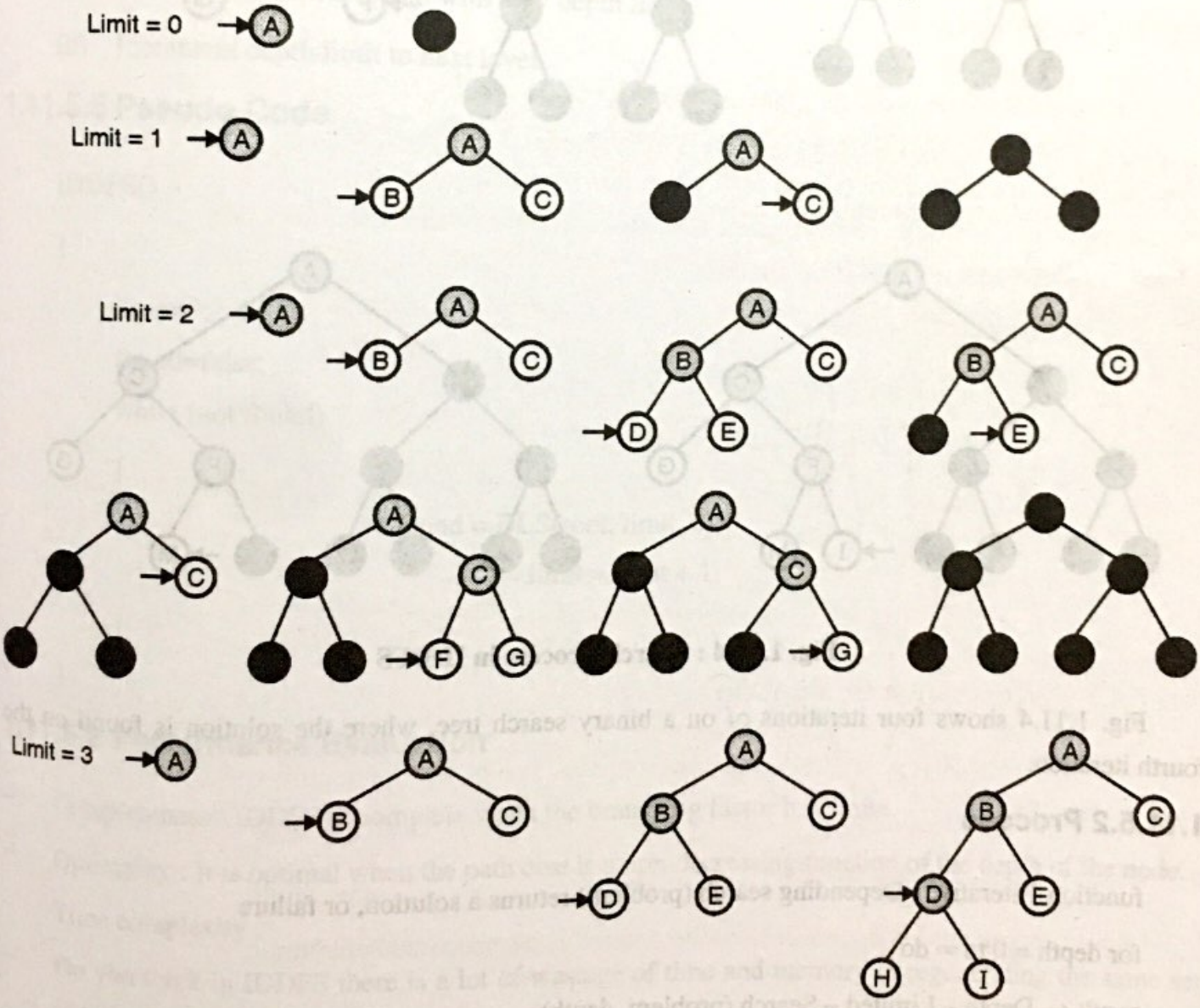


Fig. 1.11.4 : Contd...

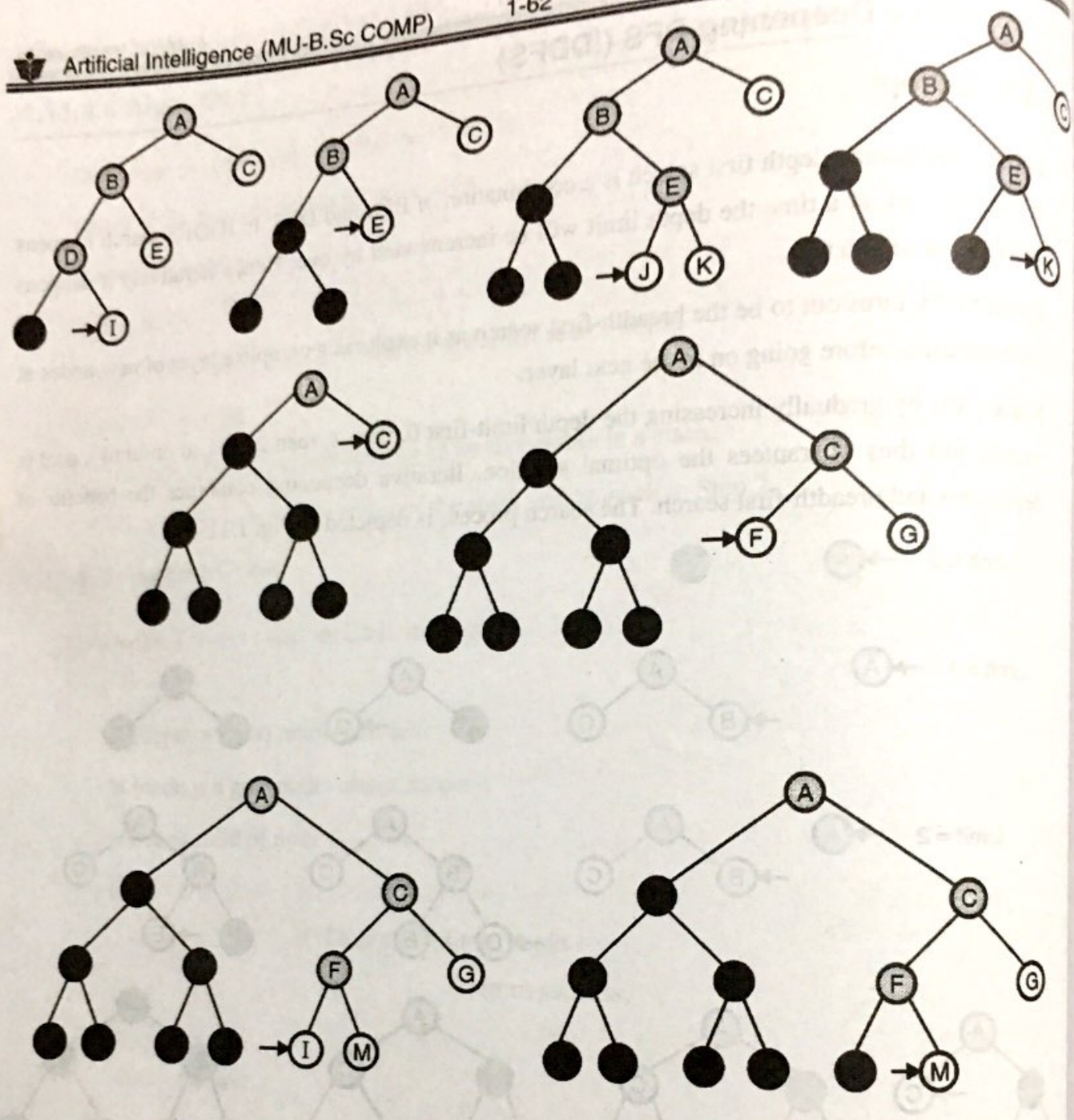


Fig. 1.11.4 : Search process in IDDFS

Fig. 1.11.4 shows four iterations of on a binary search tree, where the solution is found on the fourth iteration.

1.11.5.2 Process

function 1 iterative : Depending search (problem) returns a solution, or failure

for depth = 0 to ∞ do

result ← Depth – Limited – Search (problem, depth)

if result ≠ cutoff then return result

Fig. 1.11.4 the iterative depending search algorithm, which repeatedly applies depth limited search with increasing limits. It terminates when a solution is found or if the depth limited search returns failure, meaning that no solution exists.

1.11.5.3 Implementation

It has exactly the same implementation as that of DLS. Additionally, iterations are required to increment the depth limit by one in every recursive call of DLS.

1.11.5.4 Algorithm

Initialize depth limit to zero.

Repeat Until the goal node is found.

(a) Call Depth limited search with new depth limit.

(b) Increment depth limit to next level.

1.11.5.5 Pseudo Code

```
IDDFS()
{
    limit=0;
    found=false;
    while (not found)
    {
                found = DLS(root, limit, 0);
                limit = limit + 1;
    }
}
```

1.11.5.6 Performance Evaluation

- Completeness : IDDFS is complete when the branching factor b is finite.
- Optimality : It is optimal when the path cost is a non-decreasing function of the depth of the node.
- Time complexity :
- Do you think in IDDFS there is a lot of wastage of time and memory in regenerating the same set of nodes again and again?

- It may appear to be waste of memory and time, but it's not so. The reason is that, in a search tree with almost same branching factor at each level, most of the nodes are in the bottom level which are explores very few times as compared to those on upper level.
- The nodes on the bottom level that is level 'd' are generated only once, those on the next to bottom level are generated twice, and so on, up to the children of the root, which are generated d times. Hence the time complexity is O(bd).
- Space complexity : Memory requirements of IDDFS are modest, i.e. O(bd).
- Note : As the performance evaluation is quite satisfactory on all the four parameters, IDDFS is the preferred uninformed search method when the search space is large and the depth of the solution is not known.

1.11.6 Bidirectional Search

1.11.6.1 Concept

In bidirectional search, two simultaneous searches are run. One search starts from the initial state, called forward search and the other starts from the goal state, called backward search. The search process terminates when the searches meet at a common node of the search tree. Fig. 1.11.5 shows the general search process in bidirectional search.

1.11.6.2 Process

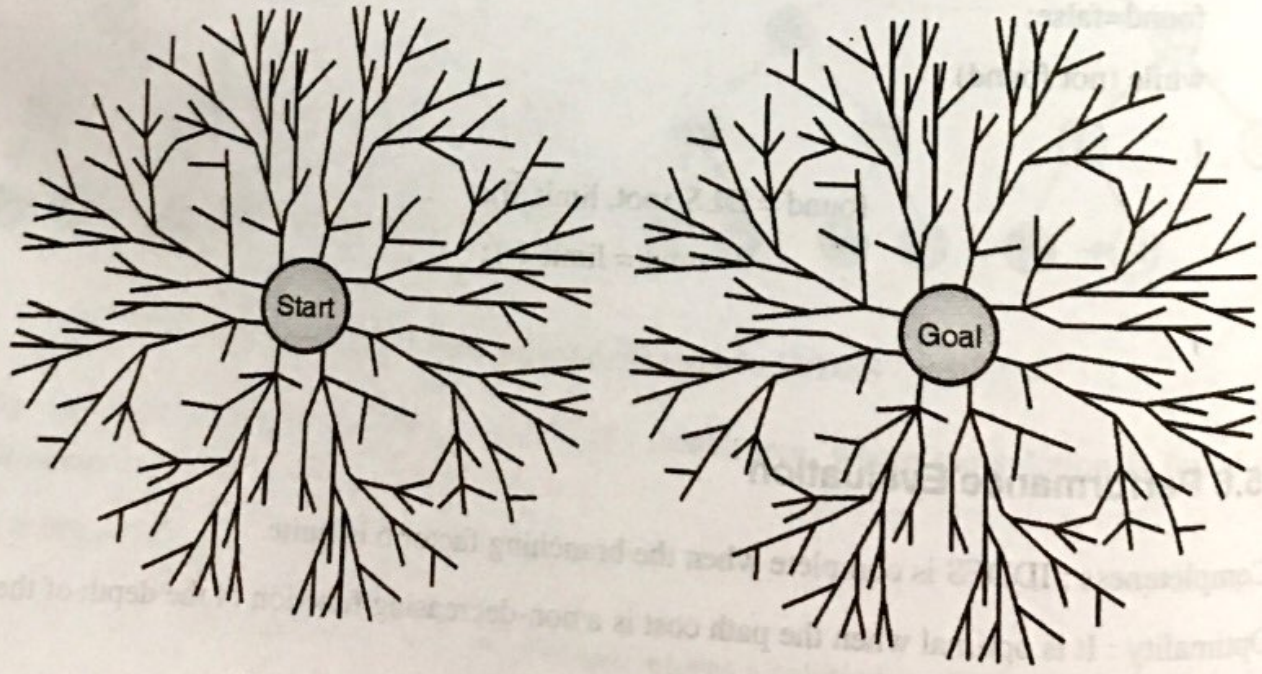


Fig. 1.11.5 : Search process in bidirectional search

1.11.6.3 Implementation

- In Bidirectional search instead of checking for goal node, one need to check whether the fringes of the two searches intersect; as they do, a solution has been found.
- When each node is generated or selected for expansion, the check can be done. It can be implemented with a hash table, to guarantee constant time.
- For example, consider a problem which has solution at depth d = 6. If we run breadth first search in each direction, then in the worst case the two searches meet when they have generated all of the nodes at depth 3. If b = 10.
- This requires a total of 2,220 node generations, as compared with 1,111,110 for a standard breadth-first search.

1.11.6.4 Performance Evaluation

- Completeness : Yes, if branching factor b is finite and both directions use breadth first search.
- Optimality : Yes, if all costs are identical and both directions use breadth first search.
- Time complexity : Time complexity of bidirectional search using breadth-first searches in both directions is O(bd/2).
- Space complexity : As at least one of the two fringes need to kept in memory to check for the common node, the space complexity is O(bd/2).

1.11.6.5 Pros of Bidirectional Search

- It is much more efficient.
- Reduces space and time requirements as, we perform two bd/2 searches, instead of one bd search.
- Example :
- Suppose b = 10, d = 6. Breadth first search will examine106 = 1, 000, 000 nodes.
- Bidirectional search will examine $2 \times 103 = 2,000$ nodes.
- One can combine different search strategies in different directions to avail better performance.

1.11.6.6 Cons of Bidirectional Search

- The search requires generating predecessors of states.
- Overhead of checking whether each new node appears in the other search is involved.
- For large d, is still impractical!
- For two bi-directional breadth-first searches, with branching factor b and depth of the solution d we have memory requirement of bd/2 for each search.

1.12 Comparing Different Techniques

- Table 1.12.1 depicts the comparison of all uninformed search techniques basis on their performance evaluation. As stated in Chapter 1, the algorithms are evaluated on four criteria viz. completeness, optimality, time complexity and space complexity.
- The notations used are as follows :

b : Branching factor

d : Depth of the shallowest solution

m : Maximum depth of the search tree

l : Depth limit

Table 1.12.1 : Comparison of tree-search strategies basis on performance Evaluation

Parameters	BFS	Uniform Cost	DFS	DLS	IDDFS	Bidirectional
Completeness	Yes	Yes	No	No	Yes	Yes
Optimality	Yes	Yes	No	No	Yes	Yes
Time Complexity	O(bd)	O(b C*/€)	O(bm)	O(bl)	O(bd)	O(bd/2)
Space Complexity	O(bd)	O(b C*/€)	O(bm)	O(bl)	O(bd)	O(bd/2)

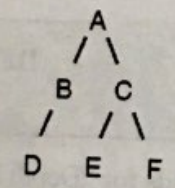
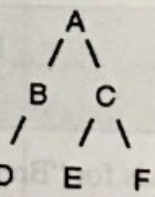
1.12.1 Difference between Unidirectional and Bidirectional Search

Sr. No.	Unidirectional search method	Bidirectional search method
1.	These methods use search tree, start node and goal node as input for starting search.	These methods have additional information about the search tree nodes, along with the start and goal node.
2.	They use only the information from the problem definition.	They incorporate additional measure of a potential of a specific state to reach the goal.
3.	Sometimes these methods use past explorations, e.g. cost of the path generated so far.	All these methods use a potential of a state (node) to reach a goal is measured through heuristic function.

Sr. No.	Unidirectional search method	Bidirectional search method
4.	All unidirectional techniques are based on the pattern of exploration of nodes in the search tree.	All bidirectional search techniques totally depend on the evaluated value of each node generated by heuristic function.
5.	In real time problems uninformed search techniques can be costly with respect to time and space.	In real time problems informed search techniques are cost effective with respect to time and space.
6.	Comparatively more number of nodes will be explored in these methods.	As compared to uninformed techniques less number of nodes are explored in this case.
7.	Example : Breadth First Search Depth First search Uniform Cost search, Depth Limited search, Iterative Deepening DFS	Example : Hill Climbing search Best First search, A* Search, IDA* search, SMA* search

1.12.2 Difference between BFS and DFS

Sr. No.	BFS	DFS
1.	BFS Stands for "Breadth First Search".	DFS stands for "Depth First Search".
2.	BFS traverses the tree level wise. i.e. each node near to root will be visited first. The nodes are explored left to right.	DFS traverses tree depth wise. i.e. nodes in particular branch are visited till the leaf node and then search continues branch by branch from left to right in the tree.
3.	Breadth First Search is implemented using queue which is FIFO list.	Depth First Search is implemented using Stack which is LIFO list.
4.	This is a single step algorithm, wherein the visited vertices are removed from the queue and then displayed at once.	This is two step algorithm. In first stage, the visited vertices are pushed onto the stack and later on when there is no vertex further to visit those are popped out.
5.	BFS requires more memory compare to DFS.	DFS require less memory compare to BFS.

Sr. No.	BFS	DFS
6.	Applications of BFS : To find Shortest path Single Source & All pairs shortest paths In Spanning tree In Connectivity	Applications of DFS : Useful in Cycle detection In Connectivity testing Finding a path between V and W in the graph. Useful in finding spanning trees & forest.
7.	BFS always provides the shallowest path solution.	DFS does not guarantee the shallowest path solution.
8.	No backtracking is required in BFS.	Backtracking is implemented in DFS.
9.	BFS is optimal and complete if branching factor is finite.	DFS is neither complete nor optimal even in case of finite branching factor.
10.	BFS can never get trapped into infinite loops.	DFS generally gets trapped into infinite loops, as search trees are dense.
11.	Example :  A, B, C, D, E, F	Example :  A, B, D, C, E, F

Syllabus Topic : Informed (Heuristic) Search Strategic

1.13 Informed Search Techniques

- Informed searching techniques is a further extension of basic un-informed search techniques. The main idea is to generate additional information about the search state space using the knowledge of problem domain, so that the search becomes more intelligent and efficient.
- The evaluation function is developed for each state, which quantities the desirability of expanding that state in order to reach the goal.

- All the strategies use this evaluation function in order to select the next state under consideration, hence the name "Informed Search".
- These techniques are very much efficient with respect to time and space requirements as compared to uninformed search techniques.

Syllabus Topic : Heuristic Function

1.14 Heuristic Function

- A heuristic function is an evaluation function, to which the search state is given as input and it generates the tangible representation of the state as output.
- It maps the problem state description to measures of desirability, usually represented as number weights. The value of a heuristic function at a given node in the search process gives a good estimate of that node being on the desired path to solution.
- It evaluates individual problem state and determines how much promising the state is. Heuristic functions are the most common way of imparting additional knowledge of the problem states to the search algorithm. Fig. 1.14.1 shows the general representation of heuristic function.

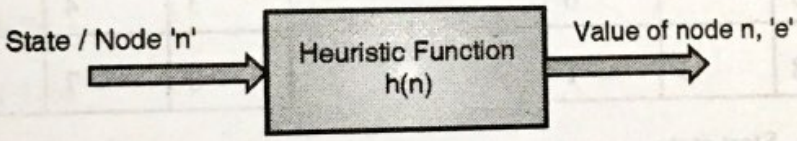


Fig. 1.14.1 : General representation of Heuristic function

- The representation may be the approximate cost of the path from the goal node or number of hopes required to reach to the goal node, etc.
- The heuristic function that we are considering in this syllabus, for a node n is, h(n) = estimated cost of the cheapest path from the state at node n to a goal state.
- Example : For the Travelling Salesman Problem, the sum of the distances traveled so far can be a simple heuristic function.
- Heuristic function can be of two types depending on the problem domain. It can be a Maximization Function or Minimization function of the path cost.
- In maximization types of heuristic, greater the cost of the node, better is the node while; in case of minimization heuristic, lower is the cost, better is the node. There are heuristics of every general applicability as well as domain specific. The search strategies are general purpose heuristics.
- It is believed that in general, a heuristic will always lead to faster and better solution, even though there is no guarantee that it will never lead in the wrong direction in the search tree.

- Design of heuristic plays a vital role in performance of search.
- As the purpose of a heuristic function is to guide the search process in the most profitable path among all that are available; a well designed heuristic functions can provides a fairly good estimate of whether a path is good or bad.
- However in many problems, the cost of computing the value of a heuristic function would be more than the effort saved in the search process. Hence generally there is a trade-off between the cost of evaluating a heuristic function and the savings in search that the function provides.
- So, are you ready to think of your own heuristic function definitions? Here is the word of caution. See how the function definition impact.
- Following are the examples demonstrate how design of heuristic function completely alters the scenario of searching process.

1.14.1 Example of 8-puzzle Problem

- Remember 8-puzzle problem? Can we estimate the number of steps required to solve an 8-puzzle from a given state?? What about designing a heuristic function for it?

7	5	4
5		6
8	3	1

Start state

	1	2
3	4	5
6	7	8

Goal State

Fig. 1.14.2 : A scenario of 8-puzzle problem

Two simple heuristic functions are :

- h1 = the number of misplaced tiles. This is also known as the Hamming Distance.
- In the Fig. 1.14.2 example, the start state has h1 = 8. Clearly, h1 is an acceptable heuristic because any tile that is out of place will have to be moved at least once, quite logical. Isn't it?
- h2 = the sum of the distances of the tiles from their goal positions. Because tiles cannot be moved diagonally, the distance counted is the sum of horizontal and vertical distances. This is also known as the Manhattan Distance. In the Fig. 3.14.2, the start state has h2 = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18. Clearly, h2 is also an admissible heuristic because any move can, at best, move one tile one step closer to the goal.
- As expected, neither heuristic overestimates the true number of moves required to solve the puzzle, which is 26 (h1 + h2). Additionally, it is easy to see from the definitions of the heuristic functions that for any given state, h2 will always be greater than or equal to h1. Thus, we can say that h2 dominates h1.

1.14.2 Example of Block World Problem

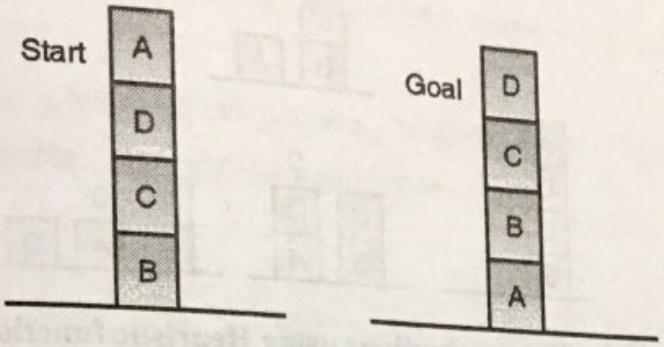


Fig. 1.14.3 : Block Problem

- Fig. 1.14.3 depicts a block problem world, where the A, B, C,D letter bricks are piled up on one another and required to be arranged as shown in goal state, by moving one brick at a time. As shown, the goal state with the particular arrangement of blocks need to be attain from the given start state. Now it's time to scratch your head and define a heuristic function that will distinguish start state from goal state. Confused??
- Let's design a function which assigns + 1 for the brick at right position and – 1 for the one which is at wrong position.

Consider Fig. 1.14.4

Local heuristic

+ 1 for each block that is resting on the thing it is supposed to be resting on.

– 1 for each block that is resting on a wrong thing.

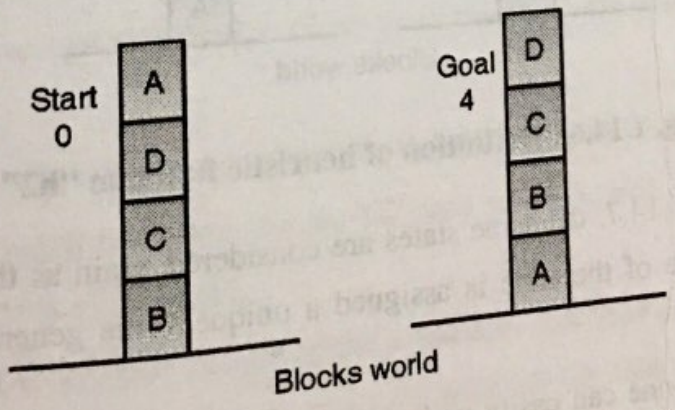


Fig. 1.14.4 : Definition of Heuristic Function "h1"

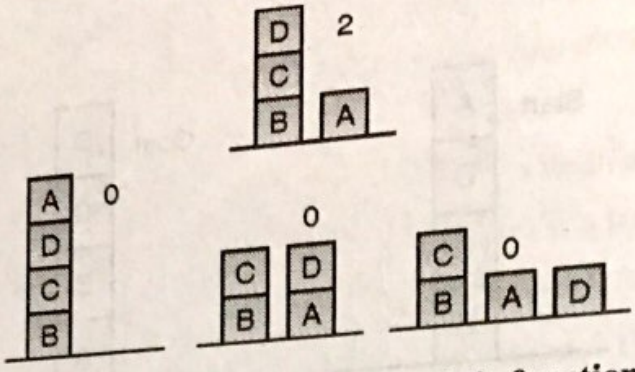


Fig. 1.14.5 : State evaluations using Heuristic function "h1"

- Fig. 1.14.5 shows the heuristic values generated by heuristic function "h1" for various different states in the state space. Please observe that, this heuristic is generating same value for different states.
- Due to this kind of heuristic the search may end up in limitless iterations as the state showing most promising heuristic value may not hold true or search may end up in finding an undesirable goal state as the state evaluation may lead to wrong direction in the search tree.
- Let's have another heuristic design for the same problem. Fig. 1.14.6 is depicting a new heuristic function "h2" definition, in which the correct support structure of each brick is given +1 for each brick in the support structure. And the one not having correct support structure, −1 for each brick in the wrong support structure.

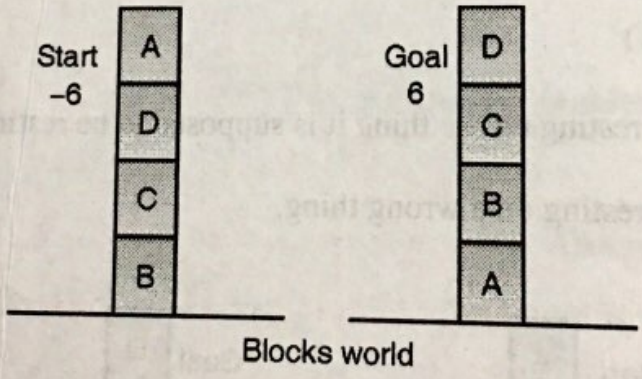


Fig. 1.14.6 : Definition of heuristic function "h2"

- As we observe in Fig. 1.14.7, the same states are considered again as that of Fig. 3.14.5, but this time using h2, each one of the state is assigned a unique value generate according to heuristic function h2.
- Observing this example one can easily understand that, in the second part of the example, search will be carried out smoothly as each unique state is getting a unique value assigned to it.

- This example makes it clear that, the design of heuristic plays a vital role in search process, as the whole search is carried out by considering the heuristic values as basis for selecting the next state to be explored.
- The state having the most promising value to reach to the goal state will be the first prior candidate for exploration, this continues till we find the goal state.

Global heuristic

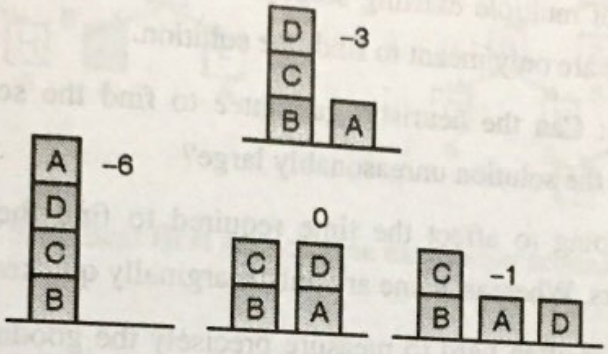


Fig. 1.14.7 : State evaluations using Heuristic function "h2"

- For each block that has the correct support structure : + 1 to every block in the support structure.
- For each block that has the wrong support structure : – 1 to every block in the support structure.
- This leads to a discussion of a better heuristic function definition.
- Is there any particular way of defining a heuristic function that will guarantee a better performance in search process??

1.14.3 Properties of Good Heuristic Function

1. It should generate a unique value for each unique state in search space.
2. The values should be a logical indicator of the profitability of the state in order to reach the goal state.
3. It may not guarantee to find the best solution, but almost always should find a very good solution.
4. It should reduce the search time; specifically for hard problems like travelling salesman problem where the time required is exponential.

- The main objective of a heuristic is to produce a solution in a reasonable time frame that is good enough for solving the problem, as it's an extra task added to the basic search process.
- The solution produced by using heuristic may not be the best of all the actual solutions to this problem, or it may simply approximate the exact solution. But it is still valuable because finding the solution does not require a prohibitively long time. So we are investing some amount of time in

generating heuristic values for each state in search space but reducing the total time involved in actual searching process.

- Do we require to design heuristic for every problem in real world? There is a trade-off criterion for deciding whether to use a heuristic for solving a given problem. It is as follows.
- Optimality : Does the problem require to find the optimal solution, if there exist multiple solutions for the same?
- Completeness : In case of multiple existing solution of a problem, is there a need to find all of them? As many heuristics are only meant to find one solution.
- Accuracy and precision : Can the heuristic guarantee to find the solution within the precision limits? Is the error bar on the solution unreasonably large?
- Execution time : Is it going to affect the time required to find the solution? Some heuristics converge faster than others. Whereas, some are only marginally quicker than classic methods.
- In many AI problems, it is often hard to measure precisely the goodness of a particular solution. But still it is important to keep performance question in mind while designing algorithm. For real world problems, it is often useful to introduce heuristics based on relatively unstructured knowledge. It is impossible to define this knowledge in such a way that mathematical analysis can be performed.

1.15 Best First Search

1.15.1 Concept

- In depth first search all competing branches are not getting expanded. And breadth first search never gets trapped on dead end paths. If we combine these properties of both DFS and BFS, it would be "follow a single path at a time, but switch paths whenever some competing path look more promising than the current one". This is what the Best First search is..!!
- Best-first search is a search algorithm which explores the search tree by expanding the most promising node chosen according to the heuristic value of nodes. Judea Pearl described best-first search as estimating the promise of node n by a "heuristic evaluation function f(n) which, in general, may depend on the description of n, the description of the goal, the information gathered by the search up to that point, and most important, on any extra knowledge about the problem domain".
- Efficient selection of the current best candidate for extension is typically implemented using a priority queue. Fig. 3.15.1 depicts the search process of Best first search on an example search tree. The values noted below the nodes are the estimated heuristic values of nodes.

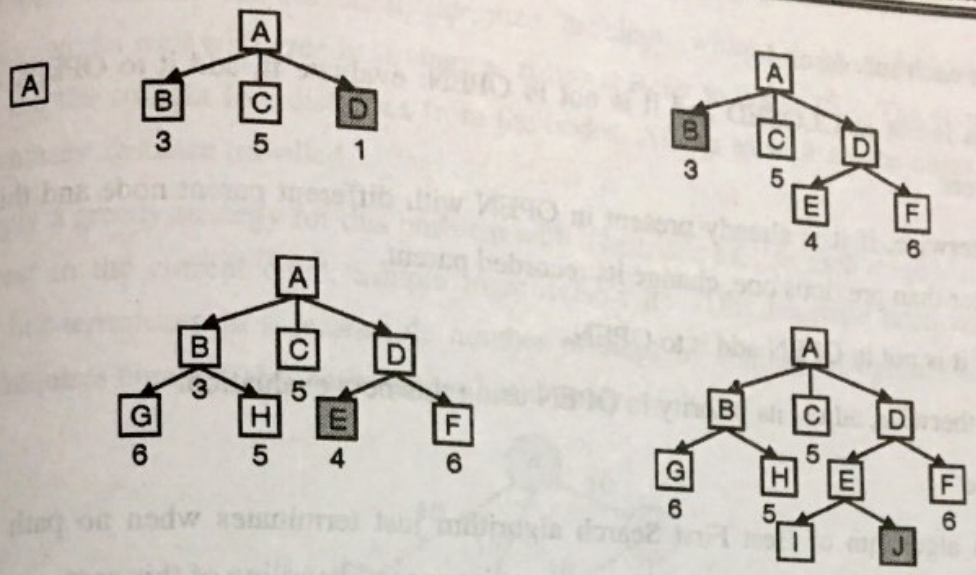


Fig. 1.15.1 : Best first search tree expansion scenario

1.15.2 Implementation

- Best first search uses two lists in order to record the path. These are namely OPEN list and CLOSED list for implementation purpose.
- OPEN list stores nodes that have been generated, but have not examined. This is organized as a priority queue, in which nodes are stored with the increasing order of their heuristic value, assuming we are implementing maximization heuristic. It provides efficient selection of the current best candidate for extension.
- CLOSED list stores nodes that have already been examined. This CLOSED list contains all nodes that have been evaluated and will not be looked at again. Whenever a new node is generated, check whether it has been generated before. If it is already visited before, check its recorded value and change the parent if this new value is better than previous one. This will avoid any node being evaluated twice, and will never get stuck into an infinite loops.

1.15.3 Algorithm : Best First Search

OPEN = [initial state]

CLOSED = []

while OPEN is not empty

do

1. Remove the best node from OPEN, call it n, add it to CLOSED.
2. If n is the goal state, backtrack path to n through recorded parents and return path.
3. Create n's successors.

4. For each successor do:

a. If it is not in CLOSED and it is not in OPEN: evaluate it, add it to OPEN, and record its parent.

b. Otherwise, if it is already present in OPEN with different parent node and this new path is better than previous one, change its recorded parent.

i. If it is not in OPEN add it to OPEN.

ii. Otherwise, adjust its priority in OPEN using this new evaluation.

done

This algorithm of Best First Search algorithm just terminates when no path is found. An actual implementation would of course require special handling of this case.

1.15.4 Performance Measures for Best first search

- Completeness : Not complete, may follow infinite path if heuristic rates each state on such a path as the best option. Most reasonable heuristics will not cause this problem however.
- Optimality : Not optimal; may not produce optimal solution always.
- Time Complexity : Worst case time complexity is still O(bm) where m is the maximum depth.
- Space Complexity : Since must maintain a queue of all unexpanded states, space-complexity is also O(bm).

1.15.5 Greedy Best First Search

- A greedy algorithm is an algorithm that follows the heuristic of making the locally optimal choice at each stage with the hope of finding a global optimum.
- When Best First Search uses a heuristic that leads to goal node, so that nodes which seems to be more promising are expanded first. This particular type of search is called greedy best-first search.
- In greedy best first search algorithm, first successor of the parent is expanded. For the successor node, check the following :

1. If the successor node's heuristic is better than its parent, the successor is set at the front of the queue, with the parent reinserted directly behind it, and the loop restarts.
2. Else, the successor is inserted into the queue, in a location determined by its heuristic value. The procedure will evaluate the remaining successors, if any of the parent.

- In many cases, greedy best first search may not always produce an optimal solution, but the solution will be locally optimal, as it will be generated in comparatively less amount of time. In mathematical optimization, greedy algorithms solve combinatorial problems.

For example, consider the traveling salesman problem, which is of a high computational complexity, works well with greedy strategy as follows. Refer to Fig. 1.15.2. The values written on the links are the straight line distances from the nodes. Aim is to visit all the cities A through F with the shortest distance travelled.

Let us apply a greedy strategy for this problem with a heuristic as, "At each stage visit an unvisited city nearest to the current city". Simple logic... isn't it? This heuristic need not find a best solution, but terminates in a reasonable number of steps by finding an optimal solution which typically requires unreasonably many steps. Let's verify.

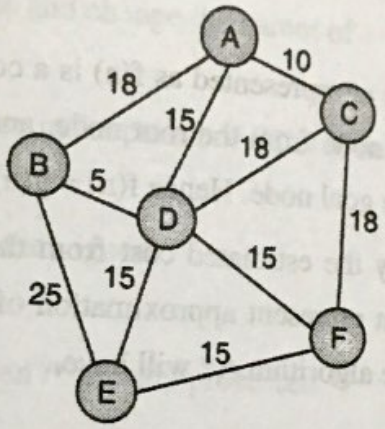


Fig. 1.15.2 : Travelling Salesmen Problem example

As greedy algorithm, it will always make a local optimal choice. Hence it will select node C first as it found to be the one with less distance from the next non-visited node from node A, and then the path generated will be A→C→D→B→E→F with the total cost = 10 + 18 + 5 + 25 + 15 = 73. While by observing the graph one can find the optimal path and optimal distance the salesman needs to travel. It turns out to be, A→B→D→E→F→C where the cost comes out to be 18 + 5 + 15 + 15 + 18 = 68.

1.15.6 Properties of Greedy Best-first Search

1. Completeness : It's not complete as, it can get stuck in loops, also is susceptible to wrong start and quality of heuristic function.
2. Optimality : It's not optimal; as it goes on selecting a single path and never checks for other possibilities.
3. TimeComplexity : O(bm), but a good heuristic can give dramatic improvement.
4. SpaceComplexity : O(bm) , It needs to keep all nodes in memory.

1.16 A* Search

1.16.1 Concept

- A* pronounced as "Aystar" (Hart, 1972) search method is a combination of branch and bound and best first search, combined with the dynamic programming principle.
- It's a variation of Best First search where the evaluation of a state or a node not only depends on the heuristic value of the node but also considers its distance from the start state. It's the most widely known form of best-first search. A* algorithm is also called as OR graph / tree search algorithm.
- In A* search, the value of a node n, represented as f(n) is a combination of g(n), which is the cost of cheapest path to reach to the node from the root node, and h(n), which is the cost of cheapest path to reach from the node to the goal node. Hence $f(n) = g(n) + h(n)$.
- As the heuristic can provide only the estimated cost from the node to the goal we can represent h(n) as h*(n); similarly g*(n) can represent approximation of g(n) which is the distance from the root node observed by A* and the algorithm A* will have,

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

- As we observe the difference between the A* and Best first search is that; in Best first search only the heuristic estimation of h(n) is considered while A* counts for both, the distance travelled till a particular node and the estimation of distance need to travel more to reach to the goal node, it always finds the cheapest solution.
- A reasonable thing to try first is the node with the lowest value of $g^*(n) + h^*(n)$. It turns out that this strategy is more than just reasonable, provided that the heuristic function h*(n) satisfies certain conditions which are discussed further in the chapter. A* search is both complete and optimal.

1.16.2 Implementation

A* does also use both OPEN and CLOSED list.

1.16.3 Algorithm (A*)

1. Initialization OPEN list with initial node; CLOSED= ϕ; g = 0, f = h, Found = false;
2. While (OPEN $\neq \phi$ and Found = false)

{ 1

i. Remove the node with the lowest value of f from OPEN to CLOSED and call it as a Best_Node.

ii If Best_Node = Goal state then Found = true

iii. else

{2

a.Generate the Succ of Best_Node

b.For each Succ do

{3

i.Compute g(Succ) = g(Best_Node) + cost of getting from Best_Node to Succ.

ii.If Succ∈ OPEN then /* already being generated but not processed */

{4

a.Call the matched node as OLD and add it in the list of Best_Node successors.

b. Ignore the Succ node and change the parent of OLD, if required.

- If g(Succ) < g(OLD) then make parent of OLD to beBest_Node and change the values of g and f for OLD

- If g(Succ) >= g(OLD) then ignore

}4

a. If Succ∈ CLOSED then /* already processed */

{5

i.Call the matched node as OLD and add it in the list of Best_Node successors.

ii. Ignore the Succ node and change the parent of OLD, if required

- If g(Succ) < g(OLD) then make parent of OLD to be Best_Node and change the values of g and f for OLD.

- Propogate the change to OLD's children using depth first search

- If g(Succ) >= g(OLD) then do nothing

}5

a. If Succ∉ OPEN or CLOSED

{6

i.Add it to the list of Best_Node's successors

ii.Compute f(Succ) = g(Succ) + h(Succ)

iii.Put Succ on OPEN list with its f value

}6

}3 /* for loop*/

}2 /* else if */

}1 /* End while */.

3. If Found = true then report the best path else report failure
4. Stop

1.16.4 Behaviour of A* Algorithm

- As stated already the success of A* totally depends upon the design of heuristic function and how well it is able to evaluate each node by estimating its distance from the goal node. Let us understand the effect of heuristic function on the execution of the algorithm and how the optimality gets affected by it.

A. Underestimation
B. Overestimation

A. Underestimation

- If we can guarantee that heuristic function 'h' never over estimates actual value from current to goal that is, the value generated by h is the always lesser than the actual cost or actual number of hopes required to reach to the goal state. In this case, A* algorithm is guaranteed to find an optimal path to a goal, if one exists.

Example :

f = g + h, Here h is underestimated.

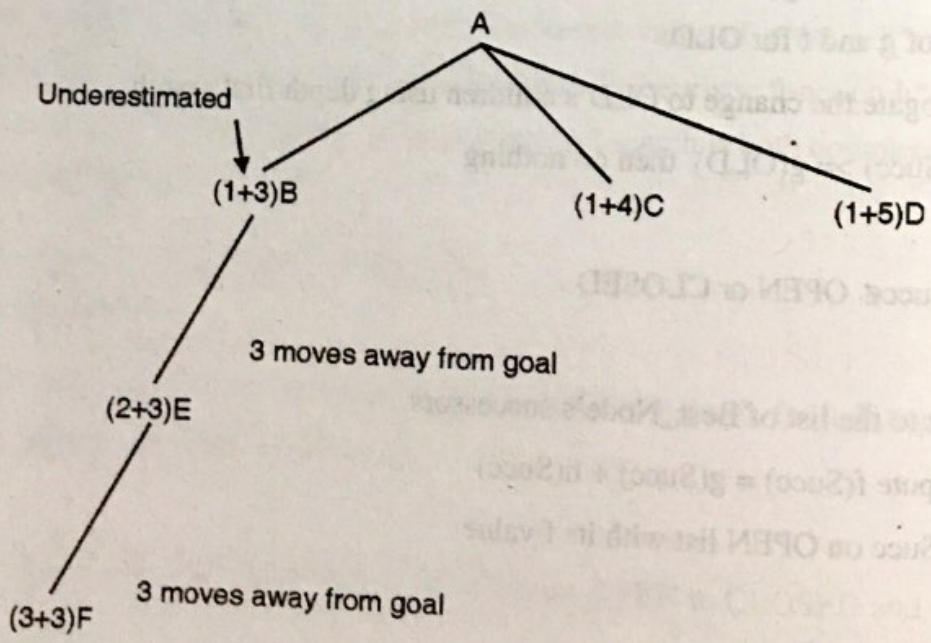


Fig. 1.16.1

}1 /* End while */.

3. If Found = true then report the best path else report failure
4. Stop

1.16.4 Behaviour of A* Algorithm

- As stated already the success of A* totally depends upon the design of heuristic function and how well it is able to evaluate each node by estimating its distance from the goal node. Let us understand the effect of heuristic function on the execution of the algorithm and how the optimality gets affected by it.

A. Underestimation
B. Overestimation

→ **A. Underestimation**

- If we can guarantee that heuristic function 'h' never over estimates actual value from current to goal that is, the value generated by h is the always lesser than the actual cost or actual number of hopes required to reach to the goal state. In this case, A* algorithm is guaranteed to find an optimal path to a goal, if one exists.

Example :

f = g + h, Here h is underestimated.

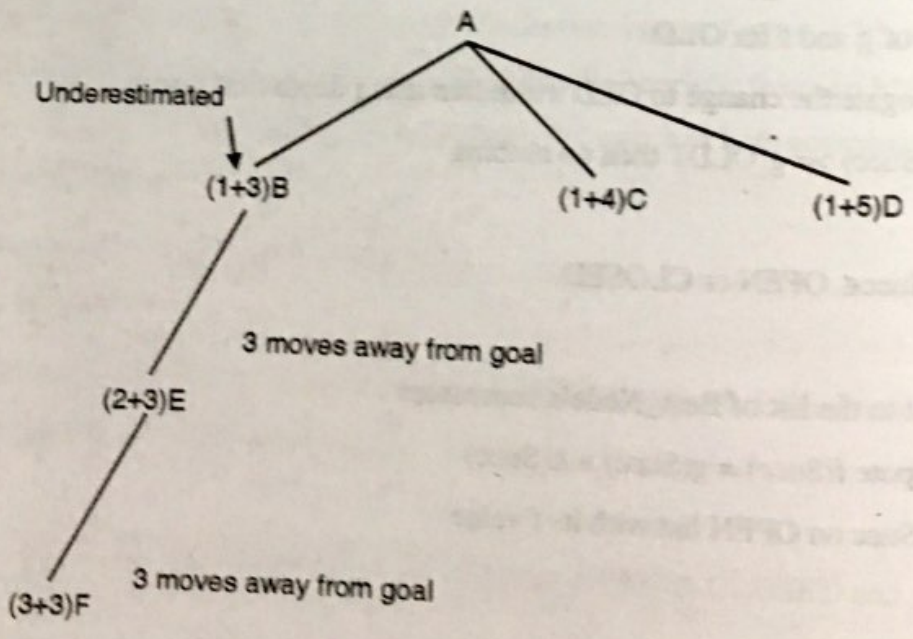


Fig. 1.16.1

If we consider cost of all arcs to be 1. A is expanded to B, C and D. 'f' values for each node is computed. B is chosen to be expanded to E. We notice that f(E) = f(C) = 5. Suppose we resolve in favor of E, the path currently we are expanding. E is expanded to F. Expansion of a node F is stopped as f(F) = 6 so we will now expand node C.

Hence by underestimating h(B), we have wasted some effort but eventually discovered that B was farther away than we thought. Then we go back and try another path, and will find optimal path.

B. Overestimation

Here h is overestimated that is, the value generated for each node is greater than the actual number of steps required to reach to the goal node.

Example

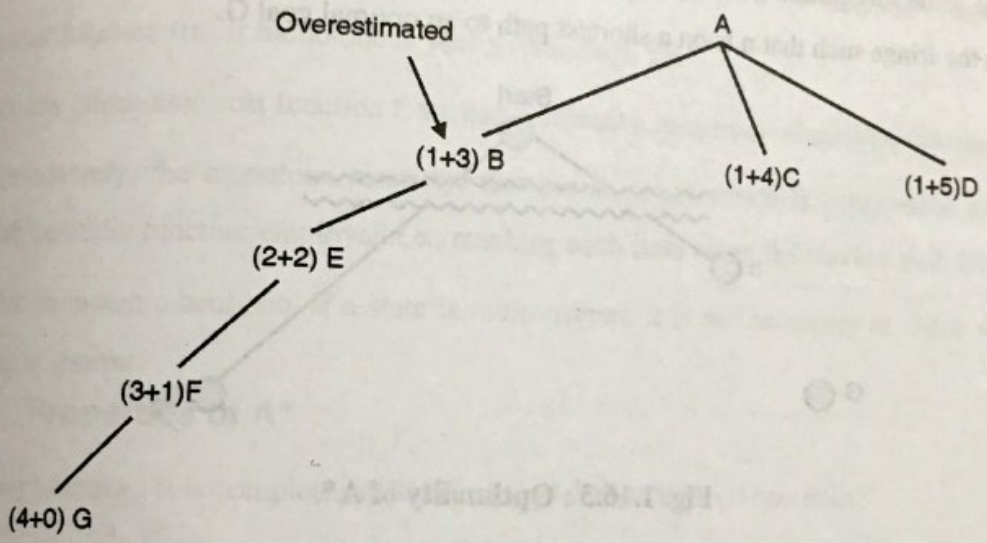


Fig. 1.16.2

- As shown in the example, A is expanded to B, C and D. Now B is expanded to E, E to F and F to G for a solution path of length 4. Consider a scenario when there a direct path from D to G with a solution giving a path of length 2.This path will never be found because of overestimating h(D).
- Thus, some other worse solution might be found without ever expanding D. So by overestimating h, one cannot guarantee to find the cheaper path solution.

1.16.5 Admissibility of A*

- A search algorithm is admissible, if for any graph, it always terminates in an optimal path from initial state to goal state, if path exists. A heuristic is admissible if it never over estimates the actual cost from current state to goal state. Alternatively, we can say that A* always terminates with the optimal path in case h(n) is an admissible heuristic function.

- A heuristic h(n) is admissible if for every node n, if $h(n) \leq h^*(n)$, where $h^*(n)$ is the true cost to reach the goal state from n. An admissible heuristic never overestimates the cost to reach the goal. Admissible heuristics are by nature optimistic because they think the cost of solving the problem is less than it actually is.
- An obvious example of an admissible heuristic is the straight line distance. Straight line distance is admissible because the shortest path between any two points is a straight line, so the straight line cannot over estimate the actual road distance.

Theorem : If h(n) is admissible, tree search using A* is optimal.

Proof : Optimality of A*with admissible heuristic.

- Suppose some suboptimal goal G2 has been generated and is in the fringe. Let n be an unexpanded node in the fringe such that n is on a shortest path to an optimal goal G.

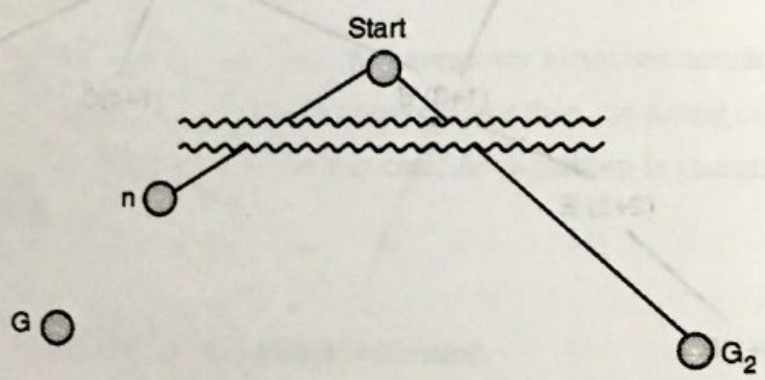


Fig. 1.16.3 : Optimality of A*

f (G2)	= g (G2)	since h (G2) = 0
g (G2)	> g (G)	since G2 is suboptimal
f (G)	= g (G)	since h (G) = 0
f (G2)	> f(G)	from above
h (n)	≤ h*(n)	since h is admissible
g (n) + h (n)	≤ g (n) + h*(n)	
f (n)	≤ f (G)	

Hence f (G2) > f (n), and A* will never select G2 for expansion

1.16.6 Monotonicity

- A heuristic function h is monotone or consistent if,
- ∀states Xi and Xj such that Xj is successor of Xi,

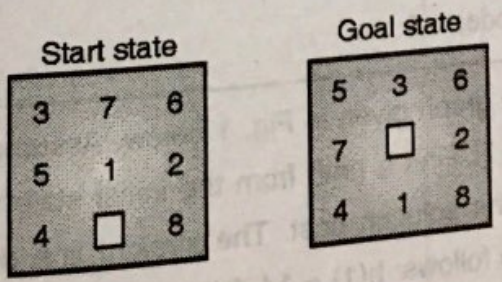
$$h(Xi) - h(Xj) \leq \text{cost}(Xi, Xj) \text{ where,}$$

- cost (Xi, Xj)actual cost of going from Xi to Xj and h (goal) = 0
- In this case, heuristic is locally admissible i.e., consistently finds the minimal path to each state they encounter in the search. The monotone property in other words is that search space which is everywhere locally consistent with heuristic function employed i.e., reaching each state along the shortest path from its ancestors. With monotonic heuristic, if a state is rediscovered, it is not necessary to check whether the new path is shorter. Each monotonic heuristic is admissible.
- A cost function f(n) is monotone if $f(n) \leq f(\text{succ}(n))$, ∀n.
- For any admissible cost function f, we can construct a monotone admissible function.
- Alternatively, the monotone property: that search space which is everywhere locally consistent with heuristic function employed i.e., reaching each state along the shortest path from its ancestors.
- With monotonic heuristic, if a state is rediscovered, it is not necessary to check whether the new path is shorter.

1.16.7 Properties of A*

1. Completeness : It is complete, as it will always find solution if one exist.
2. Optimality : Yes, it is Optimal.
3. Time Complexity : O(bm), as the number of nodes grows exponentially with solution cost.
4. Space Complexity : O(bm),as it keeps all nodes in memory.

1.16.8 Example : 8 Puzzle Problem using A* Algorithm



Evaluation function - f for EPP

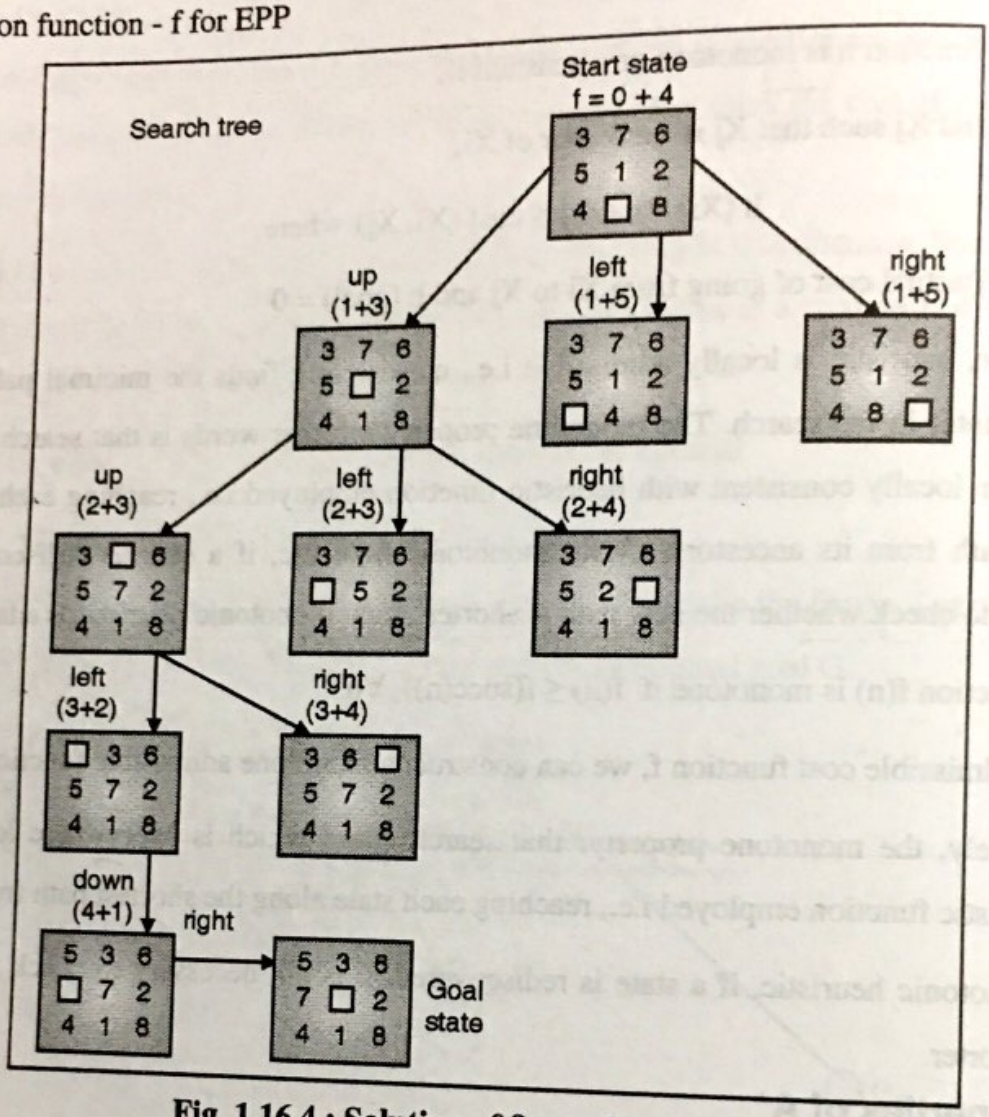


Fig. 1.16.4 : Solution of 8-puzzle using A*

The choice of evaluation function critically determines search results.

Consider Evaluation function

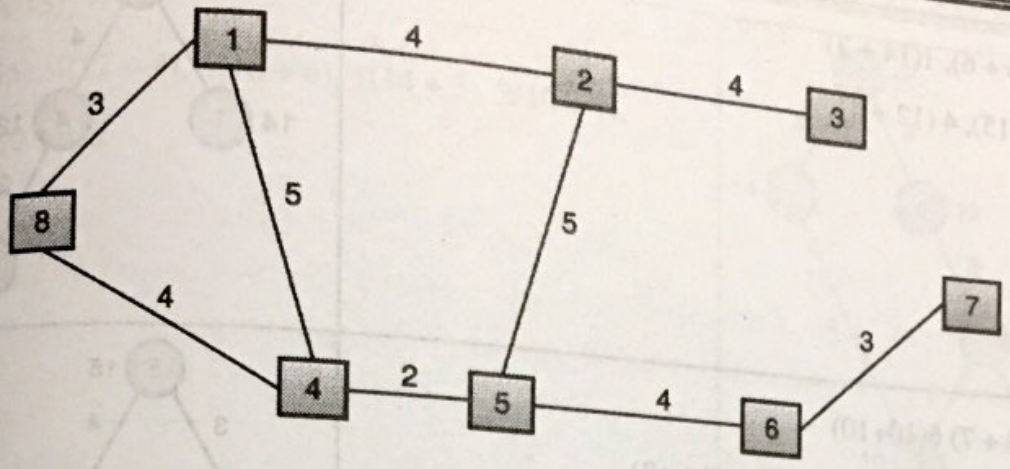
$f(X) = g(X) + h(X)$

$h(X)$ = the number of tiles not in their goal position in a given state X

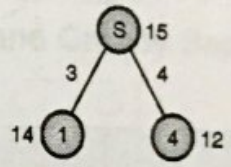
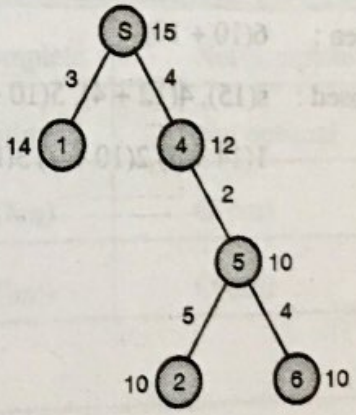
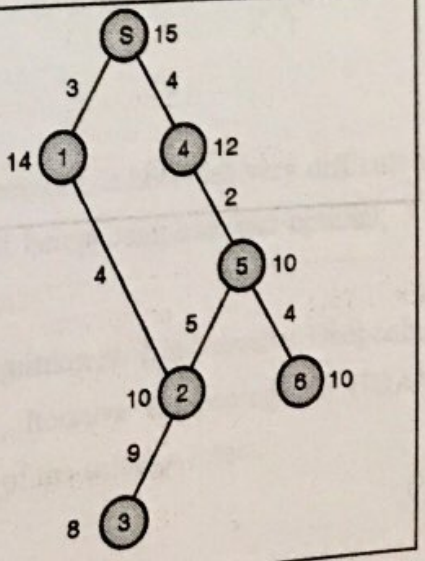
$g(X)$ = depth of node X in the search tree

For Initial node f(initial_node) = 4

Ex. 1.16.1 : Consider the graph given in Fig. 1 below. Assume that the initial state is S and the goal state is 7. Find a path from the initial state to the goal state using A* Search. Also report the solution cost. The straight line distance heuristic estimates for the nodes are as follows: h(1) = 14, h(2) = 10. h(3) = 8, h(4) = 12, h(5) = 10, h(6) = 10, h(S) = 15.

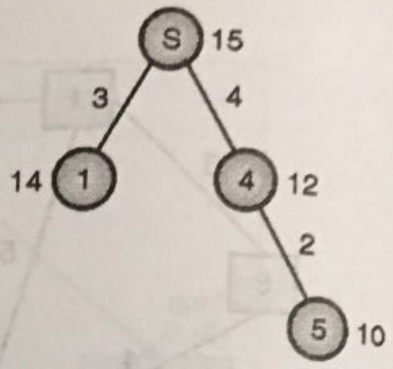


Soln. :

Open : 4(12 + 4), 1(14 + 3) Closed : S(15)	
Open : 1(14 + 3) 6(10 + 10) 2(10 + 11) Closed : S(15) 4(12 + 4) 5(10 + 6)	
Open : 3(8 +11), 6(10 + 10) Closed : S(15), 4(12+4), 5(10+6), 1(14+3), 2(10+7)	

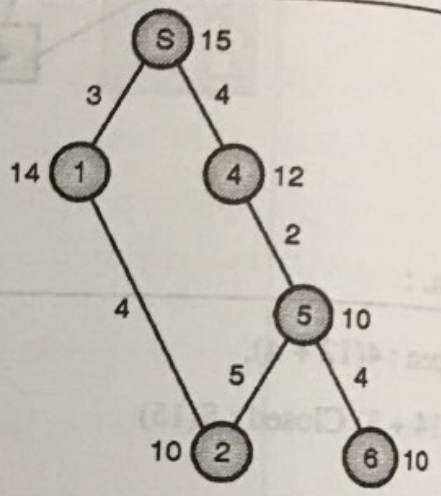
Open : 5(10 + 6), 1(14 + 3)

Closed : S (15), 4 (12 + 4)



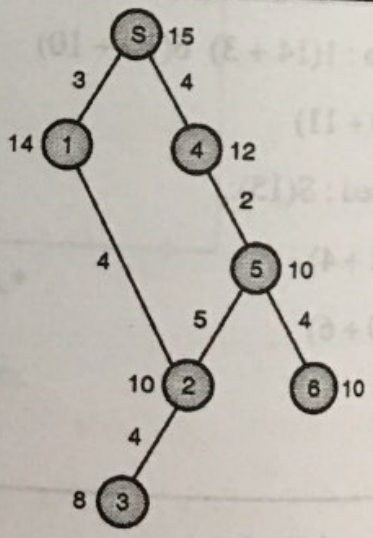
Open : 2(10 + 7) 6(10+10)

Closed : S(5) 4(12 + 4) 5(10 + 6) 1(14 +3)



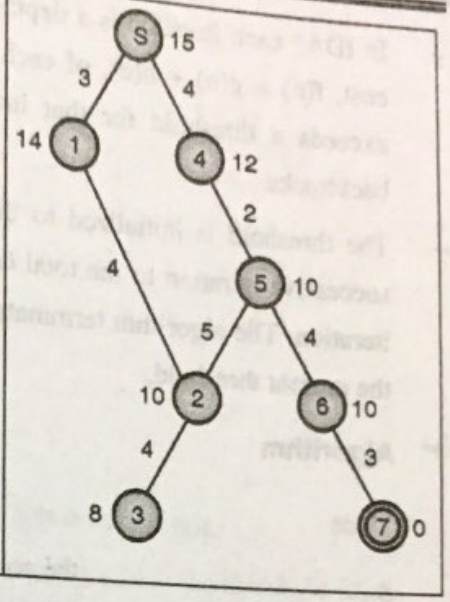
Open : 6(10 + 10)

Closed : s(15), 4(12 + 4), 5(10 + 6),
1(14 + 3) 2(10 + 7) 3(8 + 11)



Open : 7(0 + 13)

Closed : S(15), 4(12 + 4), 5(10 + 6), 1(14 + 3), 2(10 + 7), 5(8 + 4), 6(10 + 10)



1.16.9 Caparison among Best First Search, A* search and Greedy Best First Search

Algorithm	Greedy Best First Search	A* search	Best First Search
Completeness	Not complete	complete	Not complete
Optimality	Not optimal	optimal	Not optimal
Time Complexity	O(bm)	O(bm)	O(bm)
Space Complexity	O(bm)	O(bm)	O(bm)

1.17 Memory Bounded Heuristic Searches

1.17.1 Iterative Deepening A* (IDA*)

Concept

- As we have learned that A* requires to keep all the nodes in memory, it becomes very difficult to manage in case of large and complex state space. In spite of being complete and optimal, A* cannot be used for large state problems because of this limitation.
- Breadth First search faces same problem of huge memory requirement. But Iterative Deepening (IDDFS) solved it by limiting the search depth. Similarly, Iterative Deepening A* (IDA*) eliminates the memory constraints of A* by keeping optimality of the solution intact.

- In IDA* each iteration is a depth-first search with a threshold assigned to it. It keeps track of the cost, f(n) = g(n) + h(n), of each node generated. As soon as the cost of newly generated node exceeds a threshold for that iteration, the node will not be explored further and the search backtracks.
- The threshold is initialized to the heuristic estimate of the initial state. It is increased in each successive iteration to the total cost of the lowest cost node that was pruned during the previous iteration. The algorithm terminates when a goal state is reached whose total cost does not exceed the current threshold.

Algorithm

```
Node                    current node
g                       the cost to reach current node
f     estimated cost of the cheapest path (root..node..goal)
h(node)     estimated cost of the cheapest path (node..goal)
cost(node, succ)        step cost function
is_goal(node)           goal test
successors(node)  node expanding function
procedureida_star(root)
bound := h(root)
loop
t := search(root, 0, bound)
if t = FOUND then return bound
if t = ∞ then return NOT_FOUND
bound := t
end loop
end procedure
function search(node, g, bound)
f := g + h(node)
if f > bound then return f
ifis_goal(node) then return FOUND
min := ∞
```

```
for succinsuccessors(node) do
t := search(succ, g + cost(node, succ), bound)
if t = FOUND then return FOUND
if t < min then min := t
end for
return min
end function
```

- **Performance evaluation**

1. Optimality : IDA* finds optimal solution, if the heuristic function is admissible.
2. Completeness : IDA* is complete. It always finds solution if it exists within the threshold limit.
3. Time Complexity : IDA* expands the same number of nodes, as A*. So it has exponential time complexity.
4. Space Complexity : IDA* performs a series of depth-first searches. Hence, its memory requirement is linear with respect to the maximum search depth.

- From the above discussion it is clear that IDA* is optimal in time and space as compared to all other heuristic search algorithms that find optimal solutions on a tree. As in case of A*, in IDA*, we need not manage OPEN and CLOSED list hence, it often runs faster than A* and the implementation is much simpler as compared to A*.

1.17.2 Simplified Memory-bounded A* (SMA*)

- AS IDA* does not retain any path history, but the only thing kept back in iterations is the threshold f-cost value; it is bound to repeat the same expansions through various iterations. Hence there cannot be an optimize use of memory. Let's see how SMA* handles this memory management issue.

- It has following properties :

1. SMA* uses all the available memory efficiently.
2. SMA* doesn't generate states repeatedly.
3. SMA* is complete, if the available memory is sufficient to store the shallowest solution path.

4. SMA* is optimal if the available memory is sufficient to store the shallowest optimal solution path. Otherwise it generates the best solution possible with the available amount of memory.
5. SMA* is optimally efficient when enough memory is available for the entire tree search.

- The basic procedure of SMA* is just like A*. It expands the best leaf until memory is full. When there is no memory left to add newly generated node, it needs to drop one of the early expanded old node. SMA* always drops the leaf node with the highest f -cost. The dropped node is called as "forgotten node".
- SMA* then backs up the value of the forgotten node to its parent. In this way, the quality of the best path in that sub-tree is always known to the ancestor of a forgotten sub-tree. This information is used when all other paths look worse than the path it has forgotten. In that case, SMA* regenerates the forgotten sub-tree.
- What if all the leaf nodes have the same f-value? Rare case, but possible. In such situation, to avoid selecting the same node for deletion and expansion, SMA* expands the "newestbest" leaf and deletes the "oldest worst" leaf.
- If there is only one leaf, even these coincide, but in that case, the current search tree must be a single path from root to leaf that fills all of memory. If the leaf is not a goal node, then even if it is on an optimal solution path, that solution is not reachable with the available memory.
- When to choose SMA* given a choice?
- As SMA* always finds optimal solution; so in following situations SMA* is a perfect choice.
- Connected state space, i.e. when state space is graph.
- Un-uniform step costs.
- Node generation is expensive compared to the overhead of maintaining the OPEN and the CLOSED lists.
- Fig. 1.17.1 depicts a typical example where memory can store only three nodes. The original search tree is shown at the top. The second half of the figure shows iterations of SMA*. Each node is shown with the f-cost, i.e. $f = g + h$. The goal nodes D, I, F, J, K are shown in squares. The numbers in parenthesis stand for the f-cost of the best forgotten sub-tree.

The algorithm proceeds as follows :

1. At each step, one successor is added to the lowest f-cost node, which still has some successors unexplored. The root node A is expanded to generate B.
2. As f(A) = 12 which is less than f(B), which is 15, the algorithm proceeds for further expansion of A, and generates node G, and f(G) = 13. Now as all the children of A are expanded we can update its f-cost to the minimum of its children, that is 13. And notice that three nodes are already expanded hence memory is FULL.
3. Now to proceed further, we can explore the node G. but first need to make space for the new node. As the algorithm is designed we need to drop the shallowest highest f-cost leaf node. That is we need to drop node B. also add this forgotten descendant with f-cost 15, as shown in parenthesis. We can now add node H with f(H) = 18. Unfortunately, this is not the goal node, and again memory is full. Hence, there is no path to solution from H, so we set f(H) = ∞.
4. Again G is expanded for next child, and generate I with f(I) = 24. Now as G is explored fully, with children H and I with f-cost ∞ and 24, we drop H and f(G) becomes 24. As I is also a goal node, but this might not be the optimal solution as still A's f-cost is only 15.
5. As A is once again the most promising node, so B is generated again. We have found that the path through G is not so great.
6. C is first successor of B. Observe that it is a non-goal node and with maximum f-cost, so f(C) = ∞.
7. In order to expand node B further we need to first drop C. Generate node D with f(D) = 20, and this value is forwarded to B and A.

Now, the deepest and lowest cost node D is selected for further expansion, as this turns out to be the goal node, search terminates.

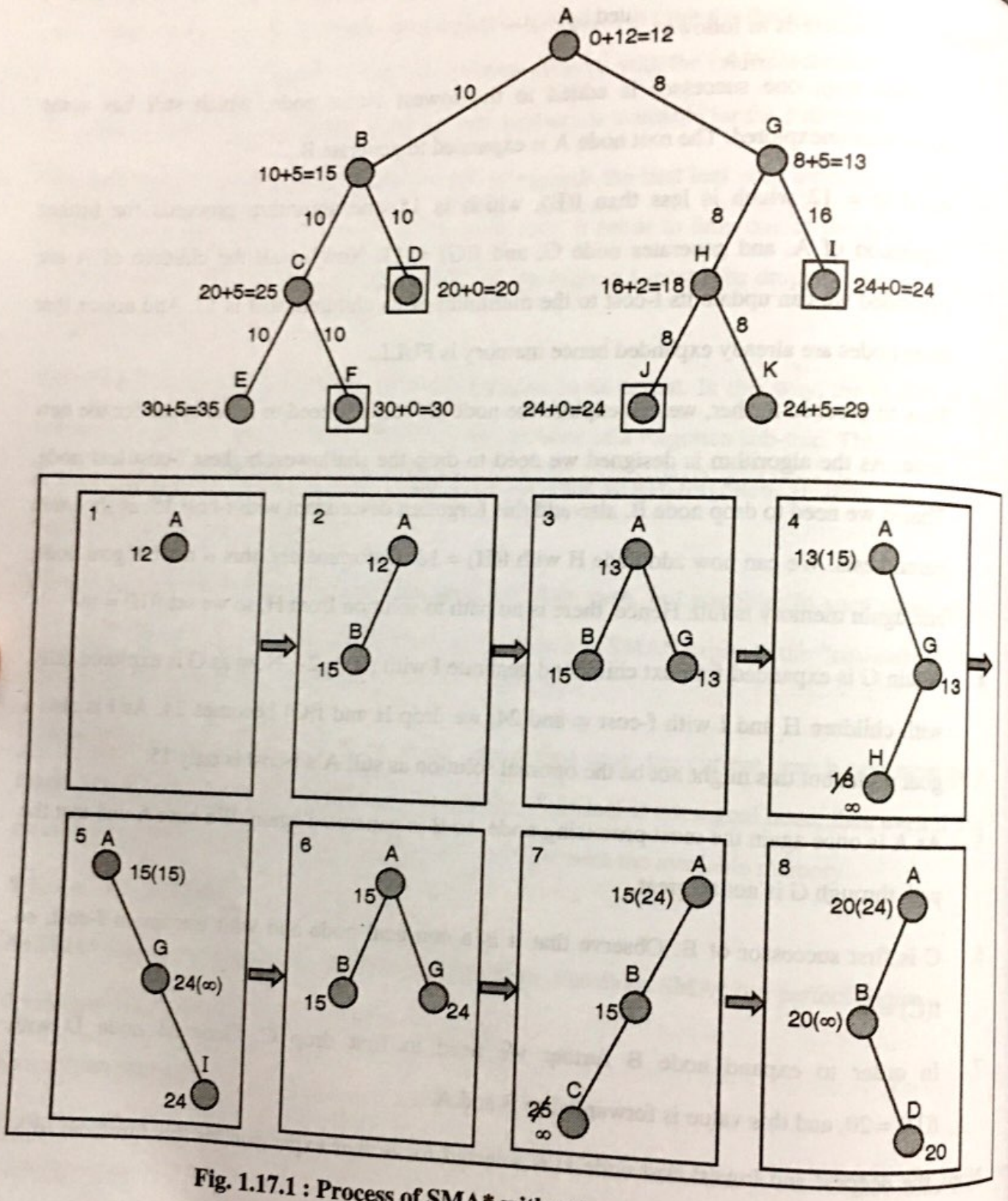


Fig. 1.17.1 : Process of SMA* with memory size of 3 nodes

1.17.3 Advantages of SMA* over A* and IDA*

- SMA* always solves more difficult problems in less amount of memory as compared to A* or IDA*.
- SMA* generates minimum number of nodes, thereby avoiding significant overhead.

- SMA* outperforms in case of real valued heuristic and in case of highly connected state space.

1.17.4 Limitation of SMA*

- As there is a limited amount of memory available in case of SMA*, in case of very hard problems where the state space is complex, SMA* needs to switch back and forth con forced to switch back and forth continually among many candidate solution paths as only a small subset of all the expanded nodes can fit in memory. Remember the problem of thrashing in disk paging systems!! Then the extra time required for repeated regeneration of the same nodes.
- This means that problems that would be practically solvable by A*, given unlimited memory, become intractable for SMA*. Hence in case of SMA*, memory limitations can make a problem intractable, with respect to computation time.

1.18 Local Search Algorithms and Optimization Problems

1.18.1 Hill Climbing

- Hill climbing is simply a combination of depth first with generate and test where a feedback is used here to decide on the direction of motion in the search space.
- Hill climbing technique is used widely in artificial intelligence, to solve solving computationally hard problems, which has multiple possible solutions.
- 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.
- In Hill climbing, each state is provided with the additional information needed to find the solution, i.e. the heuristic value. The algorithm is memory efficient since it does not maintain the complete search tree. Rather, it looks only at the current state and immediate level states.
- For example, if you want to find a mall from your current location. There are n possible paths with different directions to reach to the mall. The heuristic function will just give you the distance of each path which is reaching to the mall, so that it becomes very simple and time efficient for you to reach to the mall.
- Hill climbing attempts to iteratively improve the current state by means of an evaluation function. "Consider all the possible states laid out on the surface of a landscape. The height of any point on the landscape corresponds to the evaluation function of the state at that point" (Russell & Norvig, 2003). Fig. 1.18.1 depicts the typical hill climbing scenario, where multiple paths are available to reach to the hill top from ground level.

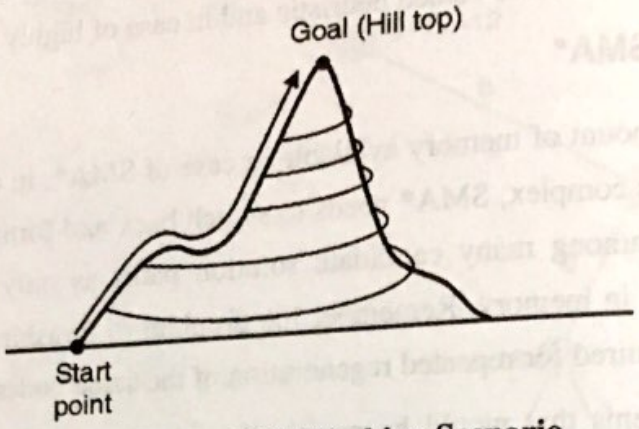


Fig. 1.18.1 : Hill Climbing Scenario

- Hill climbing always attempts to make changes that improve the current state. In other words, hill climbing can only advance if there is a higher point in the adjacent landscape.
- Hill climbing is a type of local search technique. It is relatively simple to implement. In many cases where state space is of moderate size, hill climbing works even better than many advanced techniques.
- For example, hill climbing when applied to travelling salesman problem; initially it produces random combinations of solutions having all the cities visited. Then it selects the better rout by switching the order, which visits all the cities in minimum cost.
- There are two variations of hill climbing as discussed follow.

1.18.1.1 Simple Hill Climbing

- It is the simplest way to implement hill climbing. Following is the algorithm for simple hill climbing technique. Overall the procedure looks similar to that of generate and test but, the main difference between the two is use of heuristic function for state evaluation which is used in hill climbing. The goodness of any state is decided by the heuristic value of that state. It can be either incremental heuristic or detrimental one.

Algorithm

1. Evaluate the initial state. If it is a goal state, then return and quit; otherwise make it a current state and go to Step 2.
2. Loop until a solution is found or there are no new operators left to be applied (i.e. no new children nodes left to be explored).
   a. Select and apply a new operator (i.e. generate new child node)
   b. Evaluate the new state :
   (i) If it is a goal state, then return and quit.

(ii) If it is better than current state then make it a new current state.

(iii) If it is not better than the current state then continue the loop, go to Step 2.

- As we study the algorithm, we observe that in every pass the first node / state that is better than the current state is considered for further exploration. This strategy may not guarantee that most optimal solution to the problem, but may save upon the execution time.

1.18.1.2 Steepest Ascent Hill Climbing

- As the name suggests, steepest hill climbing always finds the steepest path to hill top. It does so by selecting the best node among all children of the current node / state. All the states are evaluated using heuristic function. Obviously, the time requirement of this strategy is more as compared to the previous one. The algorithm for steepest ascent hill climbing is as follows.

Algorithm

1. Evaluate the initial state, if it is a goal state, return and quit; otherwise make it as a current state.
2. Loop until a solution is found or a complete iteration produces no change to current state :
   a. SUCC = a state such that any possible successor of the current state will be better than SUCC.
   b. For each operator that applies to the current state, evaluate the new state:
      (i) If it is goal; then return and quit
      (ii) If it is better than SUCC then set SUCC to this state.
   c. SUCC is better than the current state → set the current state to SUCC.

- As we compare simple hill climbing with steepest ascent, we find that there is a tradeoff for the time requirement and the accuracy or optimality of the solution.
- In case of simple hill climbing technique as we go for first better successor, the time is saved as all the successors are not evaluated but it may lead to more number of nodes and branches getting explored, in turn the solution found may not be the optimal one.
- While in case of steepest ascent hill climbing technique, as every time the best among all the successors is selected for further expansion, it involves more time in evaluating all the successors at earlier stages, but the solution found will be always the optimal solution, as only the states leading to hill top are explored. This also makes it clear that the evaluation function i.e. the heuristic function definition plays a vital role in deciding the performance of the algorithm.

1.18.1.3 Limitations of Hill Climbing

- Now let's see what can be the impact of incorrect design of heuristic function on the hill climbing

- Following are the problems that may arise in hill climbing strategy. Sometimes the algorithms may lead to a position, which is not a solution, but from which there is no move possible which will lead to a better place on hill i.e. no further state that is going closer to the solution. This will happen if we have reached one of the following three states.
- Local Maximum : A "local maximum" is a location in hill which is at height from other parts of the hill but is not the actual hill top. In the search tree, it is a state better than all its neighbors, but there is not next better state which can be chosen for further expansion. Local maximum sometimes occur within sight of a solution. In such cases they are called "Foothills".

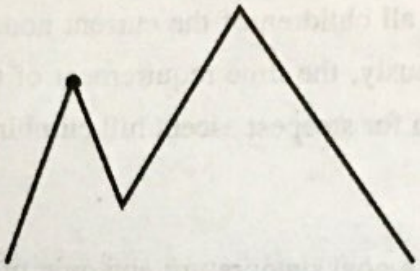


Fig. 1.18.2 : Local Maximum

- In the search tree local maximum can be seen as follows :

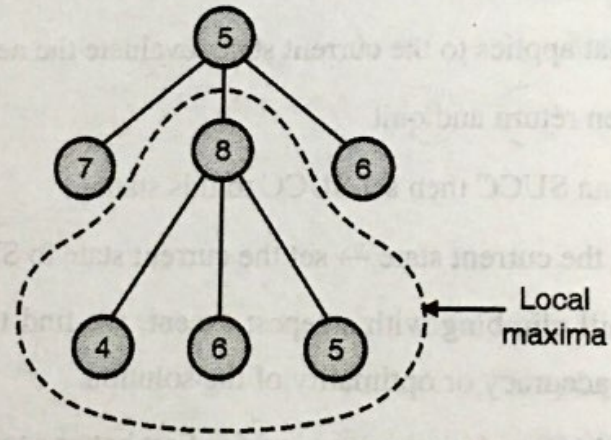


Fig. 1.18.3 : Local maxima in Search Tree

- Plateau : A "plateau" is a flat area at some height in hilly region. There is a large area of same height in plateau. In the search space, plateau situation occurs when all the 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.

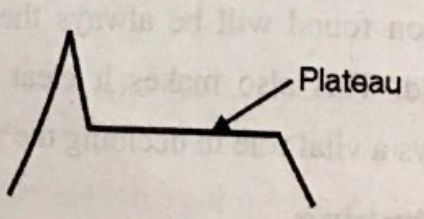


Fig. 1.18.4 : Plateau in hill climbing

In the search tree plateau can be identified as follows :

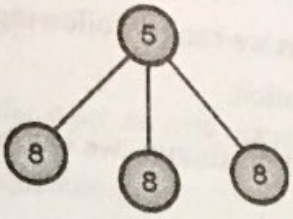


Fig. 1.18.5

- Ridge : A "ridge" is an area in the hill such that, it is higher than the surrounding areas, but there is no further uphill path from ridge. In the search tree it is the situation, where all successors are either of same value or lesser, it's a ridge condition. The suitable successor cannot be searched in a simple move.

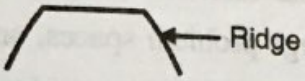


Fig. 1.18.6 : Ridge in Hill Climbing

- In the search tree ridge can be identified as follows :

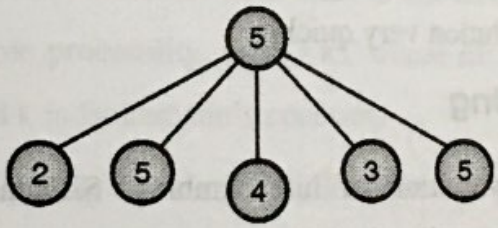


Fig. 1.18.7

- Fig. 1.18.8 depicts all the different situations together in hill climbing.

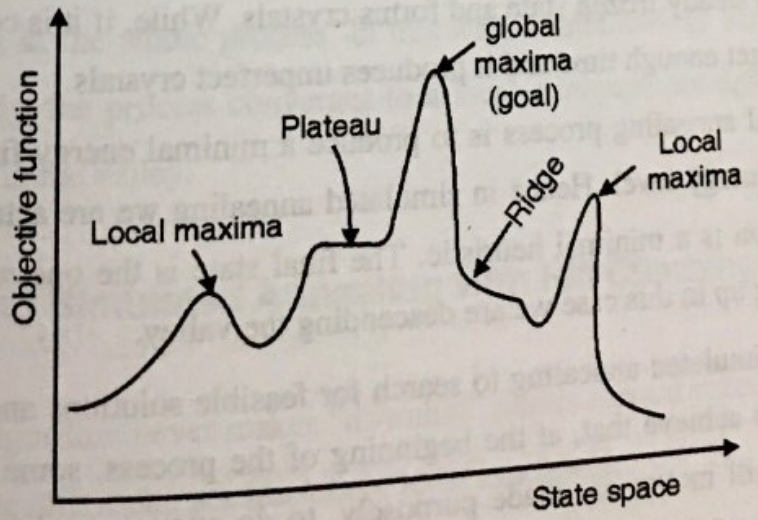


Fig. 1.18.8 : Hill climbing problems scenario

1.18.1.4 Solutions on Problems in Hill Climbing

- In order to overcome these problems we can try following techniques. At times combination of techniques will provide a better solution.
  1. A good way to deal with local maximum, we can back track to some earlier nodes and try different direction.
  2. In case of plateau and ridges, make a big jump in 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 plateau and ridges.
- Hill climbing is a local method. It decides what to do next by looking only at the "immediate" consequences of its choices. Global information might be encoded in heuristic functions. Hill climbing becomes inefficient in large problem spaces, and when combinatorial explosion occurs. But it uses very little memory, usually a constant amount, as it doesn't retain the path.
- It is a useful when combined with other methods. The success of hill climbing depends very much on the shape of the state space. If there are few local maxima and plateau, random-restart hill climbing will find a good solution very quickly.

1.18.2 Simulated Annealing

- Simulated annealing is a variation of hill climbing. Simulated annealing technique can be explained by an analogy to annealing in solids. In the annealing process in case of solids, a solid is heated past melting point and then cooled.
- With the changing rate of cooling, the solid changes its properties. If the liquid is cooled slowly, it gets transformed in steady frozen state and forms crystals. While, if it is cooled quickly, the crystal formation will not get enough time and it produces imperfect crystals.
- The aim of physical annealing process is to produce a minimal energy final state after raising the substance to high energy level. Hence in simulated annealing we are actually going downhill and the heuristic function is a minimal heuristic. The final state is the one with minimum value, and rather than climbing up in this case we are descending the valley.
- The idea is to use simulated annealing to search for feasible solutions and converge to an optimal solution. In order to achieve that, at the beginning of the process, some downhill moves may be made. These downhill moves are made purposely, to do enough exploration of the whole space early on, so that the final solution is relatively insensitive to the starting state. It reduces the chances of getting caught at a local maximum, or plateau, or a ridge.

Algorithm

1. Evaluate the initial state.
2. Loop until a solution is found or there are no new operators left to be applied:

   Set T according to an annealing schedule

   Select and apply a new operator

   Evaluate the new state :

   goal→ quit

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

   $\Delta E < 0 \rightarrow$ new current state

   else→ new current state with probability $e^{-\Delta E / kT}$.

- We observe in the algorithm that, if the next state is better than the current, it readily accepts it as a new current state. But in case when the next state is not having the desirable value even then it accepts that state with some probability, $e^{-\Delta E / kT}$ where ΔE is the positive change in the energy level, T is temperature and k is Boltzmann's constant.
- Thus, in the simulated annealing there are very less chances of large uphill moves than the small one. Also, the probability of uphill moves decreases with the temperature decrease. Hence uphill moves are more likely in the beginning of the annealing process, when the temperature is high. As the cooling process starts, temperature comes down, in turn the uphill moves. Downhill moves are allowed any time in the whole process. In this way, comparatively very small upward moves are allowed till finally, the process converges to a local minimum configuration, i.e. the desired low point destination in the valley.

1.18.2.1 Comparing Simulated Annealing with Hill Climbing

- A hill climbing algorithm never makes "downhill" moves toward states with lower value and it can be incomplete, because it can get stuck on a local maximum.
- In contrast, a purely random walk, i.e. moving to a successor chosen at random from the set of success or sin dependent of whether it is better than the current state, is complete but extremely inefficient. Therefore, it is reasonable to try a combination of hill climbing with a random walk in some way that yields both efficiency and completeness. Simulated annealing is the answer...!!

- As we know that, hill climbing can get stuck at local minima or maxima, thereby halting the algorithm abruptly, it may not guarantee optimal solution. Few attempts were made to solve this problem by trying hill climbing considering multiple start points or by increasing the size of neighborhood, but none worked out to produce satisfactory results. Simulated annealing has solved the problem by performing some downhill moves at the beginning of search so that, local maximum can be avoided at later stage.
- Hill climbing procedure chooses the best state from those available or at least better than the current state for further expansion. Unlike hill climbing, simulated annealing chooses a random move from the neighborhood. If the successor state turned out to be better than its current state then simulated annealing will accept it for further expansion. If the successor state is worse, then it will be accepted based on some probability.

1.18.3 Local Beam Search

- In all the variations of hill climbing till now, we have considered only one node getting selected at a time for further search process. These algorithms are memory efficient in that sense. But when an unfruitful branch gets explored even for some amount of time it is a complete waste of time and memory. Also the solution produce may not be the optimal one.
- The local beam search algorithm keeps track of k best states by performing parallel k searches. At each step it generates successor nodes and selects k best nodes for next level of search. Thus rather than focusing on only one branch it concentrates on k paths which seems to be promising. If any of the successors found to be the goal, search process stops.
- In parallel local beam search, the parallel threads communicate to each other, hence useful information is passed among the parallel search threads.
- In turn, the states that generate the best successors say to the others, "Come over here, the grass is greener!" The algorithm quickly terminates unfruitful branches exploration and moves its resources to where the path seems most promising. In stochastic beam search the maintained successor states are chosen with a probability based on their goodness.

Algorithm : Local Beam search

Step 1 : Found = false;

Step 2 : NODE = Root_node;

Step 3 : If NODE is the goal node, then *Found = true* else find SUCCs of NODE, if any with its estimated cost and store in OPEN list;

Step 4 : While (Found = false and not able to proceed further)

```
{
Sort OPEN list;
Select top W elements from OPEN list and put it in W_OPEN list and empty OPEN list;
    While (W_OPEN ≠ ϕ and Found = false)
    {
    Get NODE from W_OPEN;
        if NODE = Goal state then Found = true else
        {
        Find SUCCs of NODE, if any with its estimated cost
        store in OPEN list;
        }
} // end inner while
} // end outer while
```

Step 5 : If *Found = true* then return *Yes* otherwise return *No* and Stop

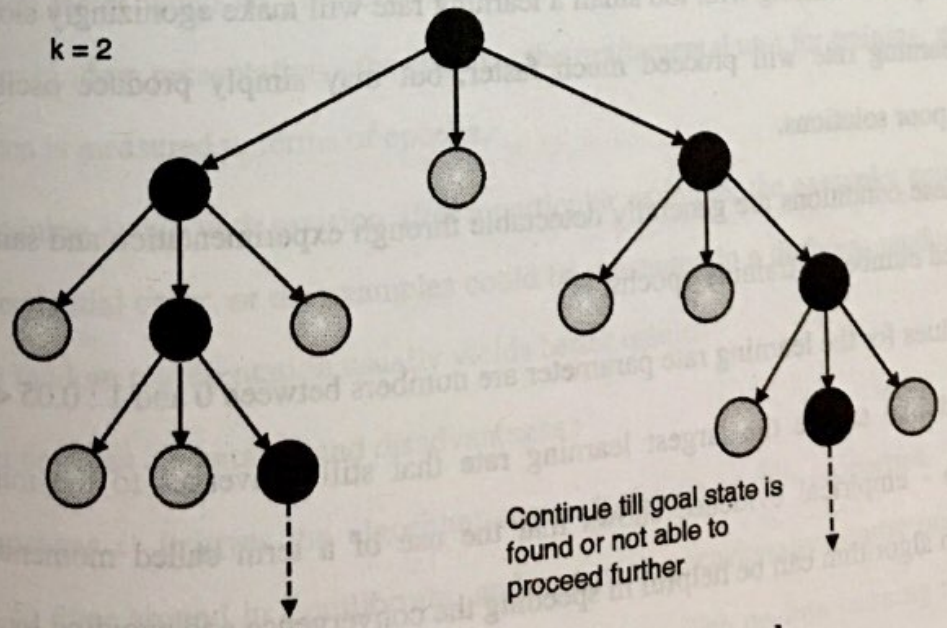


Fig. 1.18.9 : Process of local Beam Search

- As shown in Fig. 3.18.9, here k = 2, hence two better successors are selected for expansion at first level of search and at each next level, two better successors will be selected by both searches. They do exchange their information with each other throughout the search process. The search will continue till goal state is found or no further search is possible.
- It may seem to be that local beam search is same as running k searches in parallel. But it is not so. In case of parallel searches, all search run independent of each other. While in case of local beam search, the parallel running threads continuously coordinate with one another to decide the fruitful region of the search tree.
- Local beam search can suffer from a lack of diversity among the k states by quickly concentrating to small region of the state space.
- While it is possible to get excellent fits to training data, the application of back propagation is fraught with difficulties and pitfalls for the prediction of the performance on independent test data. Unlike most other learning systems that have been previously discussed, there are far more choices to be made in applying the gradient descent method.
- The key variations of these choices are : The learning rate and local minima - the selection of a learning rate is of critical importance in finding the true global minimum of the error distance.
- Back propagation training with too small a learning rate will make agonizingly slow progress. Too large a learning rate will proceed much faster, but may simply produce oscillations between relatively poor solutions.
- Both of these conditions are generally detectable through experimentation and sampling of results after a fixed number of training epochs.
- Typical values for the learning rate parameter are numbers between 0 and 1 : $0.05 < h < 0.75$.
- One would like to use the largest learning rate that still converges to the minimum solution momentum - empirical evidence shows that the use of a term called momentum in the back propagation algorithm can be helpful in speeding the convergence and avoiding local minima.

The idea about using a momentum is to stabilize the weight change by making non radical revisions using a combination of the gradient decreasing term with a fraction of the previous weight change :

$$\Delta w(t) = -\partial Ee/\partial w(t) + \alpha \Delta w(t-1)$$

where a is taken 0£ a £ 0.9, and t is the index of the current weight change.

This gives the system a certain amount of inertia since the weight vector will tend to continue moving in the same direction unless opposed by the gradient term.

The momentum has the following effects :

- It smooths the weight changes and suppresses cross-stitching, that is cancels side-to-side oscillations accross the error valey;
- When all weight changes are all in the same direction the momentum amplifies the learning rate causing a faster convergence;
- Enables to escape from small local minima on the error surface.

The hope is that the momentum will allow a larger learning rate and that this will speed convergence and avoid local minima. On the other hand, a learning rate of 1 with no momentum will be much faster when no problem with local minima or non-convergence is encountered ; · sequential or random presentation - the epoch is the fundamental unit for training, and the length of training often is measured in terms of epochs.

During a training epoch with revision after a particular example, the examples can be presented in the same sequential order, or the examples could be presented in a different random order for each epoch. The random representation usually yields better results.

The randomness has advantages and disadvantages :

- **Advantages :** It gives the algorithm some stochastic search properties. The weight state tends to jitter around its equilibrium, and may visit occasionally nearby points. Thus it may escape trapping in suboptimal weight configurations. The on-line learning may have a better chance of finding a global minimum than the true gradient descent technique.

- **Disadvantages :** The weight vector never settles to a stable configuration. Having found a good minimum it may then continue to wander around it.

- Random initial state - unlike many other learning systems, the neural network begin in a random state. The network weights are initialized to some choice of random numbers with a range typically between -0.5 and 0.5 (The inputs are usually normalized to numbers between 0 and 1). Even with identical learning conditions, the random initial weights can lead to results that differ from one training session to another.

- The training sessions may be repeated till getting the best results.

Review Questions

Q. Define agent and give classification of agents.

Q. What is intelligent agent?

Q. Write a short note on : Structure of Intelligent agents.

Q. Give types of agents.

Q. Define artificial intelligence.

Q. Write a short note on : Applications of artificial intelligence.

Q. Explain the various artificial intelligence problems and artificial intelligence techniques.

Q. Explain various techniques for solving problems by searching.

Q. What is artificial intelligence?

Q. What are the components of AI?

Q. What are the various AI techniques?

Q. Explain various applications of Artificial Intelligence.

Q. Differentiate between BFS and DFS.

Q. Write short note on bidirectional search.

Q. Write a note on BFA and Uniform cost search.

Q. Compare and contrast DFS, FLS and IDDFS.

Q. What are various informed search techniques? Explain A* with example.

Q. Compare Best First Search and A* with an example.

Q. Write algorithm for Best first search and specify its properties.

Q. What is heuristic function? What are the qualities of a good heuristic?

Chapter Ends...

❑❑❑

CHAPTER 2

Unit II

Learning From Examples

Syllabus

Learning from Examples : Forms of Learning, Supervised Learning, Learning Decision Trees, Evaluating and Choosing the Best Hypothesis, Theory of Learning, Regression and Classification with Linear Models, Artificial Neural Networks, Nonparametric Models, Support Vector Machines, Ensemble Learning, Practical Machine Learning.

Syllabus Topic : Learning from Examples

2.1 Learning from Examples

- We all learn a lot from examples. You must have solved word problems in school, to teach these word problems teachers give one simple example in class and then solve that example with exaplaination on board.
- As a student we observe how teacher is solving the given problem, which steps are followed while solving, what explaination is given by teacher while solving problem, etc and then we try to use same method to solve the next word problem.
- In Explanation or example Based Learning (EBL), agent learns by examining particular situations and relating them to gained knowledge base (knowledge base stores knowledge in the form of known general facts).
- Also agent makes use of this gained knowledge for solving similar type of problems.

2.1.1 Example Based Learning (EBL) Architecture

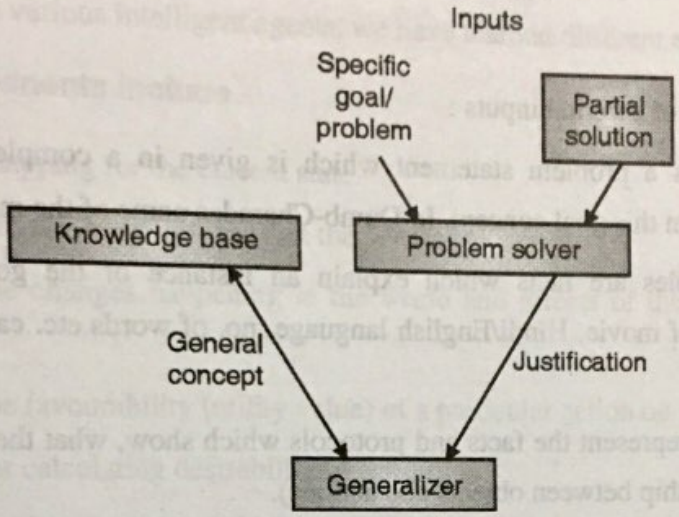


Fig. 2.1.1 : EBL Architecture

- EBL architecture shows that Problem solver takes two inputs from the environment : Specific goal/problem and a partial solution. Problem solver processes these inputs and gives the justification to generalizer.
- Generalizer takes general concepts as input from the knowledge base and compares the explanation of problem solver with it to come up with solution to the given problem.

2.1.2 EBL System Representation

- EBL system representation shows how explanation based learning is carried out in a system. Have you played "Dumb-Charades"?
- Explanation based learning is similar to this game. Let us understand how so. Player-1 gets a movie and other player has to guess that movie.

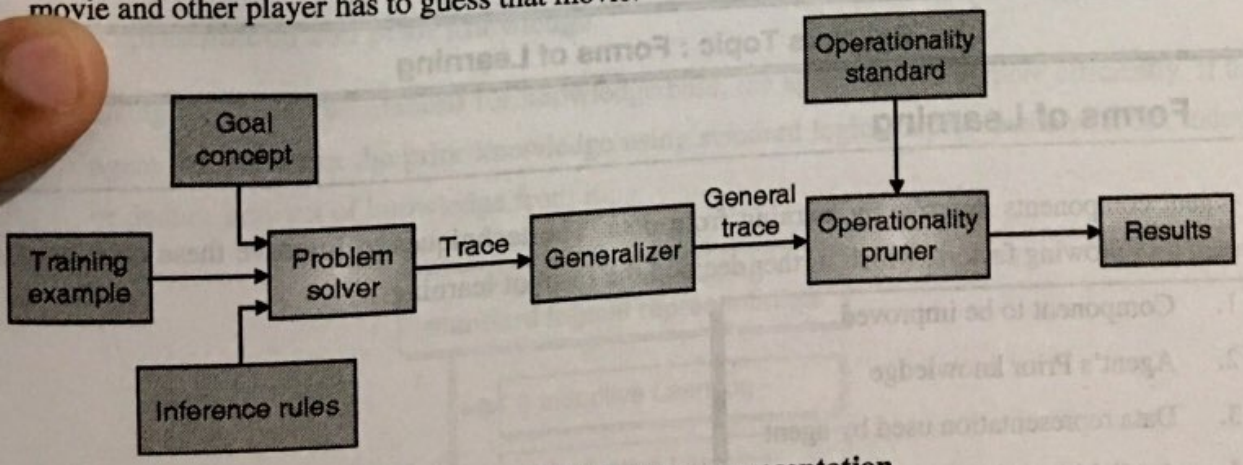


Fig. 2.1.2 : EBL System representation

There are three main components of the EBL system :

➜ **1. Problem solver**

It accepts three types of external inputs :

- Goal concept is a problem statement which is given in a complex format and agent is supposed to learn this goal concept. In Dumb-Charades name of the movie is a goal concept.
- Training examples are facts which explain an instance of the goal concept. In Dumb-Charades type of movie, Hindi/English language, no. of words etc. can be treated as training examples
- Inference rules represent the facts and protocols which show, what the learner already knows (i.e. the relationship between objects and actions).
- In Dumb-Charades actors, songs, director, etc. can be treated as inference rules.
- Based on these three inputs problem solver tries to generate a solution.

➜ **2. Generalizer**

- Output of the problem solver is given as input to the generalizer which compares the explanation of problem solver with knowledge base and gives the output to the operationality pruner.

➜ **3. Operationality pruner**

- It takes two inputs one from generalized and one from Operationality standard.
- Operationality standard gives description of the final concept; also it specifies the form in which learned concept should be expressed.

Syllabus Topic : Forms of Learning

2.2 Forms of Learning

Agent components improve by learning from data. The techniques to improve these components are based on following factors, which further decided the form of learning.

1. Component to be improved
2. Agent's Prior knowledge
3. Data representation used by agent
4. Availability of type of feedback

1. Component to be improved

While learning about various intelligent agents, we have learned different components of them.

The agents' components include

1. Condition – action mapping for the current state.
2. Mechanism to form model of world basis on the percept sequence.
3. Knowledge about the changes happening in the world and effects of the possible actions on the world.
4. Approximation of the favourability (utility value) of a particular action on the world.
5. Action-value data for calculating desirability of action.
6. Information about goals to be achieved in order to improve the agents' performance.

Any of these components can be improved.

Consider following example :

- A taxi driver agent can learn and improve many of its components in the following way.
- Component 1: Every time the instructor shouts 'Brake!" the agent might learn a condition-action model for when to apply brakes.
- Component 2: By seeing many camera images that it is told contain buses, it can learn to recognize them.
- Component 3: By braking hard on a wet road, it can learn the effects of its actions.
- Component 4: When it receives no tip from passengers who have been thoroughly shaken up during the trip, it can learn a useful component of its overall utility function.

2. Representation and prior knowledge

- Using formal representations for knowledge base, the agent cam learn more efficiently. If the agent can represent the prior knowledge using standard logical representations, it can induce or deduce new set of knowledge from it.

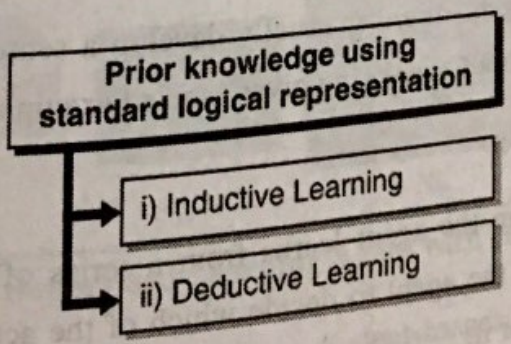


Fig. 2.2.1

➜ **(i) Inductive Learning :** This learning pattern is based on examples is called as Inductive learning. For example, learning a function based on input - output pairs.

➜ **(ii) Deductive Learning :** This learning is useful because it allows more efficient processing. In deductive learning, from a known general rule, the agent generates a new rule ,that is logically entailed or derived.

➜ **3. Feedback to learn from**

There are three types of feedback that determine the three main types of learning :

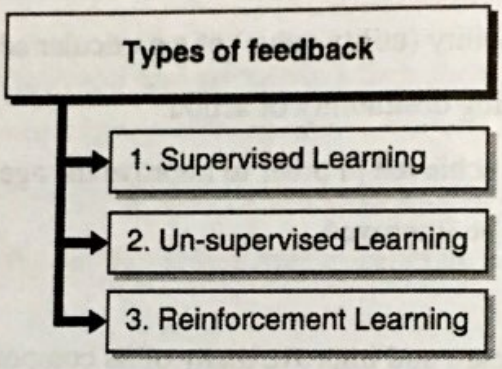


Fig. 2.2.2

➜ **1. Supervised Learning**

- In supervised learning the agent has given the set of input and desired output pairs. It observes some example input-output pairs and learns a function that maps from input to output.
- For example, in Taxi driving agent's environment, the inputs are percepts and the output is provided by a teacher who says "Brake!" or "Turn left".

➜ **2. Un-supervised Learning**

- In unsupervised learning the agent learns patterns in the input even though no explicit feedback is supplied. The most common unsupervised learning task is clustering or detecting in which, the learning pattern is based on finding similar group.
- For example, the taxi agent might gradually develop a sense of "good traffic timings" and "bad traffic timings" without ever being given labeled examples by the teacher.

➜ **3. Reinforcement Learning**

- In reinforcement learning the agent learns from a series of reinforcements i.e. rewards or punishments. It is up to the agent to decide which of the actions prior to the reinforcement were most responsible for it.

- For example, the lack of a tip at the end of the journey gives the taxi agent an indication that it did something wrong.

Syllabus Topic : Supervised Learning

2.2.1 Supervised Learning

- Learning agent and teacher both take input from surrounding environment. Teacher has the desired answer key for the given problems.
- Whereas, learning agent processes the given input and gets an output. This actual output is subtracted from the desired output by the adder, which shows the error in the actual output.

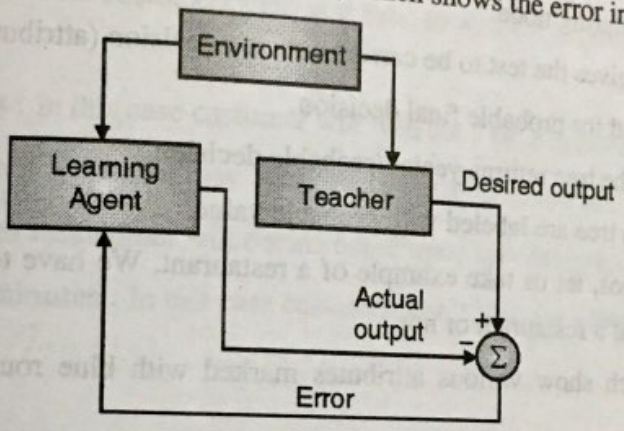


Fig. 2.2.3 : Supervised Learning

- This error is given as input to the learning agent, so that it can learn from this error and while generating output for the next time it can try to remove that error.
- As per the name Supervised Learning is performed under supervision of a Teacher. Teacher can be an agent/system which has a correct answer for each example. This answer can be a variable in numeric form, a categorical variable, etc.
- Let's take one example where Learning Agent has to identify living and nonliving things. Supervised Learning will have correct answers see labels shown in Fig. 2.2.4.

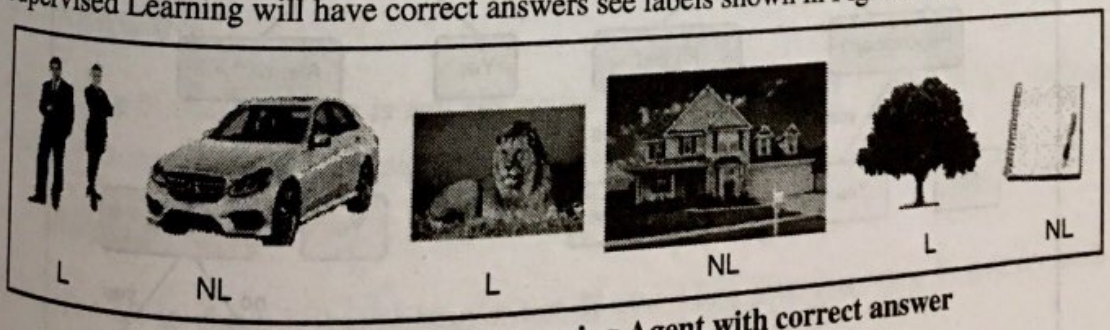


Fig. 2.2.4 : Supervised Learning Agent with correct answer

- In short, the way in real life teacher guides a student to get better results; supervised learning method guides learning agent with the help of teacher to get better results.

Syllabus Topic : Learning Decision Trees

2.3 Learning Decision Trees

- Decision tree based learning is classification based learning for discrete values and regression for continuous values.
- Set of values are given as input to the decision tree and it gives output based on the classification and prediction.
- Decision Tree represents protocols which can be easily understood by humans. It is a classifier which can be represented in the tree structure, where each node is leaf node or a decision node.
- There are few characteristics of a decision tree, which are given as follows :
  - Root node is a staring node.
  - Decision node gives the test to be carried out on a decision (attribute value).
  - Leaf nodes stand for probable final decision.
  - Every node in the tree returns yes/no/probable decision.
  - Branches of the tree are labeled with probable value.
- To learn decision trees, let us take example of a restaurant. We have to predict whether customer will wait for a table at a restaurant or not.
- See Fig. 2.3.1, which show various attributes marked with blue rounded corner rectangles for solving the problem.

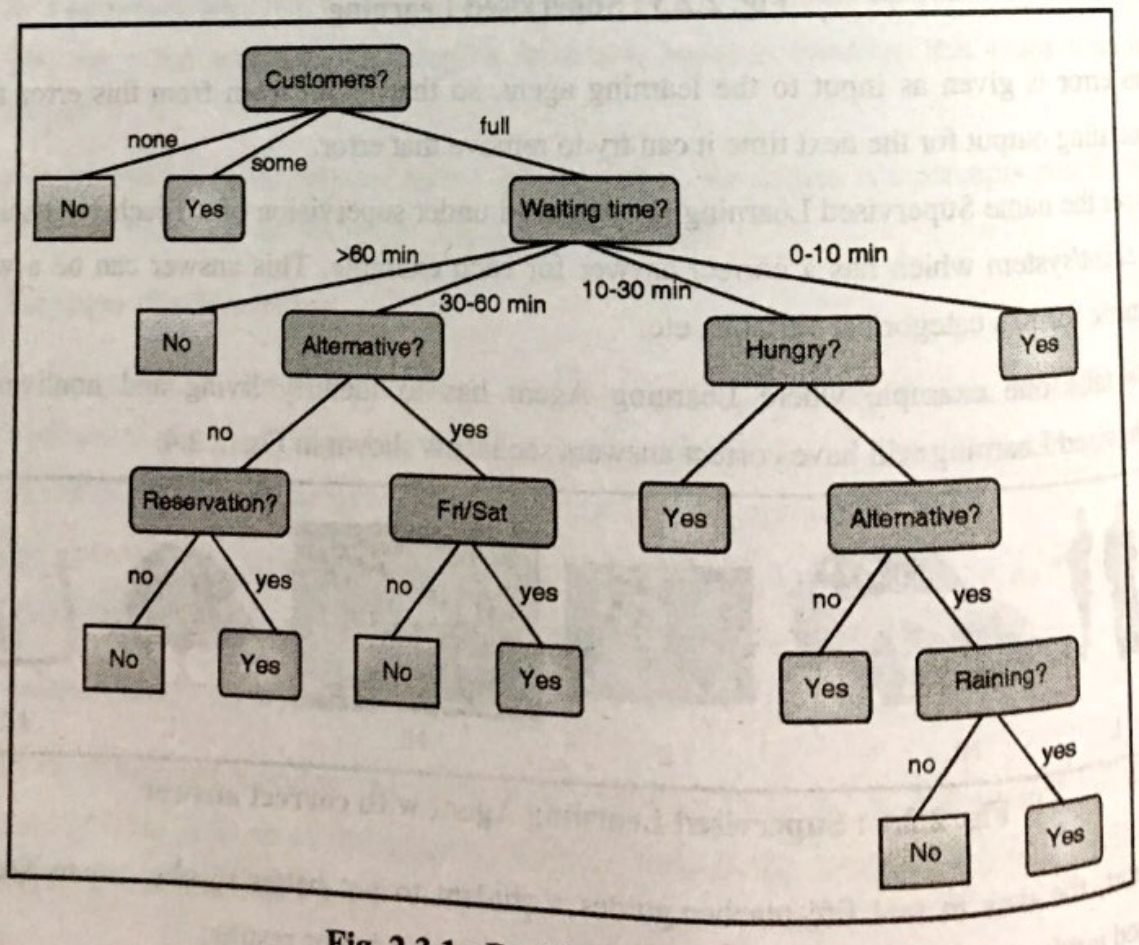


Fig. 2.3.1 : Decision tree example

As it can be observed from Fig. 2.3.1 there are six attributes which are considered for solving the problem.

1. **Customers**

Number of customers present in the restaurant. This attribute can have three probable results :

- **None :** There are no customers, so you can get a any table in the restaurant;
- **Some :** There are few customers but you can get a table in the restaurant;
- **Full :** All tables are occupied with customers, so you have to wait for some time.

2. **Waiting time**

If the restaurant is full with customers then you have to wait for some time. The estimated waiting time can be :

- **0 to 10 minutes :** In this case customer will wait for a table at restaurant.
- **10 to 30 minutes :** Customer may check for alternative restaurant.
- **30 to 60 minutes :** Customer will decide based upon how hungry he/she is.
- **More than 60 minutes :** In this case customer won't wait for a table at restaurant.

3. **Alternative**

Customer will check if there is any alternative restaurant in nearby locality. While finding out an alternative customer has to consider two possibilities which are attributes:

(a) **Fri/ Sat :** Customer has to check if it is a Friday or Saturday.

(b) **Reservation :** Customer has to check if they have any reservation.

4. **Hungry**

If the waiting time is between 10-30 minutes then customer will check if they are hungry. If not hungry then they can wait, else they can check if there is any alternative.

5. **Alternative**

Customer will check if there is any alternative restaurant in nearby locality. While finding out an alternative customer has to consider if it is raining or not.

6. **Raining**

Customer has to check if it is raining or not. If it's raining then it is better to wait for table at the same restaurant else customer can search for other restaurants.

Green rounded corner rectangles in Fig. 2.3.1 displays the conditions when customers can wait for the table at the same restaurant.

Table 2.3.1 : Decision tree learning example

Example	Attributes							Target
	Alt	Fri	Hun	Cust	Rain	Res	Est	Will Wait
X_1	T	F	T	Some	F	T	0-10	T
X_2	T	F	T	Full	F	F	30-60	F
X_3	F	F	F	Some	F	F	0-10	T
X_4	T	T	T	Full	F	F	10-30	T
X_5	T	T	F	Full	F	T	> 60	F
X_6	F	F	T	Some	T	T	0-10	T
X_7	F	F	F	None	T	F	0-10	F
X_8	F	F	T	Some	T	T	0-10	T
X_9	F	T	F	Full	T	F	> 60	F
X_{10}	T	T	T	Full	F	T	10-30	F
X_{11}	F	F	F	None	F	F	0-10	F
X_{12}	T	T	T	Full	F	F	30-60	T

Here, T = True, F = False

- Pure node means a binary node where, answer can be either yes or no (i.e. True or False).
- Many times it is not possible to create pure nodes (only yes/no type), so we have to make sure to divide the tree such that it leads to pure nodes, so that we can make some decision at the end.
- Entropy is a purity measure (i.e. a measure of "order" in an agent).
- To calculate entropy we have a general formula which is given as follows :

Where, $\text{Entropy} = -\sum_i P(v_i) \log_{10} [P(v_i)]$

i = number of values

P = purity of each decision

v = decision is yes /no

- For Table 2.3.1, there are total 12 example. As it can be seen there are six true values and six false values, so entropy value can be given as follows :

$$\text{Entropy} = -\left(\frac{6}{12}\right)\log_{10}\left(\frac{6}{12}\right) - \left(\frac{6}{12}\right)\log_{10}\left(\frac{6}{12}\right) = 0.30 \quad \ldots(2.3.1)$$

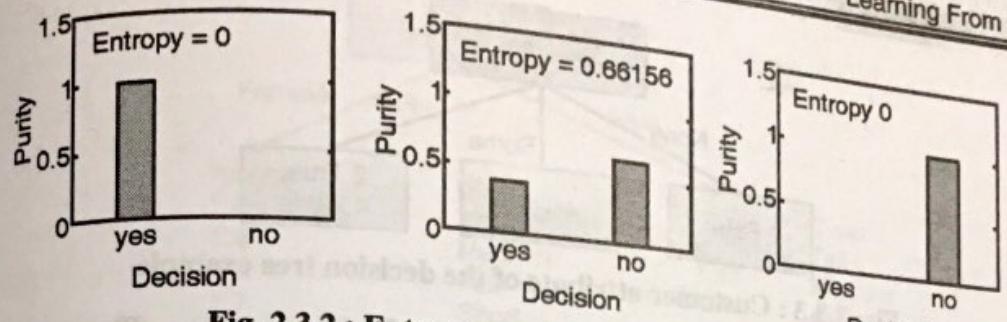


Fig. 2.3.2 : Entropy graphs : decision v/s purity

- After observing Fig. 2.3.2. We can say that,
  - Entropy value is zero for a pure node (i.e. pure "yes"/"no" node).
  - Entropy value is maximal when all probable decisions are equally likely. In such cases it is difficult to make any decision. So we have to split the tree further.
- Main goal of a decision tree is to reduce the entropy value in every node, so that agent can take some decision at the end.
- See Customer column from Table 2.3.2.

Table 2.3.2

Example	Attributes							Target
	Alt	Fri	Hun	Cust	Rain	Res	Est	Will Wait
X_1	T	F	T	Some	F	T	0-10	T
X_2	T	F	T	Full	F	F	30-60	F
X_3	F	F	F	Some	F	F	0-10	T
X_4	T	T	T	Full	F	F	10-30	T
X_5	T	T	F	Full	F	T	> 60	F
X_6	F	F	T	Some	T	T	0-10	T
X_7	F	F	F	None	T	F	0-10	F
X_8	F	F	T	Some	T	T	0-10	T
X_9	F	T	F	Full	T	F	> 60	F
X_{10}	T	T	T	Full	F	T	10-30	F
X_{11}	F	F	F	None	F	F	0-10	F
X_{12}	T	T	T	Full	F	F	30-60	T

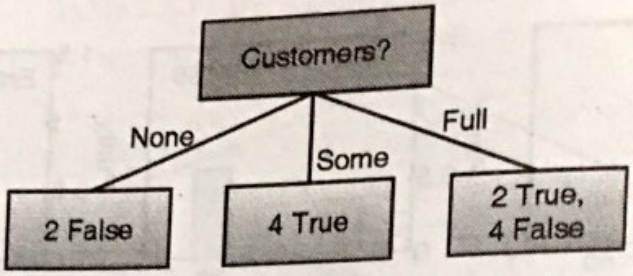


Fig. 2.3.3 : Customer attribute of the decision tree example

$$\text{Entropy} = \frac{2}{12}\left[\left(\frac{0}{2}\right)\log_{10} - \left(\frac{2}{2}\right)\log_{10}\left(\frac{2}{2}\right)\right] + \frac{4}{12}\left[-\left(\frac{4}{4}\right)\log_{10}\left(\frac{4}{4}\right) - \left(\frac{0}{4}\right)\log_{10}\left(\frac{0}{4}\right)\right]$$

$$+\frac{6}{12}\left[-\left(\frac{2}{6}\right)\log_{10}\left(\frac{2}{6}\right) - \left(\frac{4}{6}\right)\log_{10}\left(\frac{4}{6}\right)\right] = 0.14$$

$$\text{Entropy decrease} = 0.30 - 0.14 = 0.16 \quad \ldots(2.3.2)$$

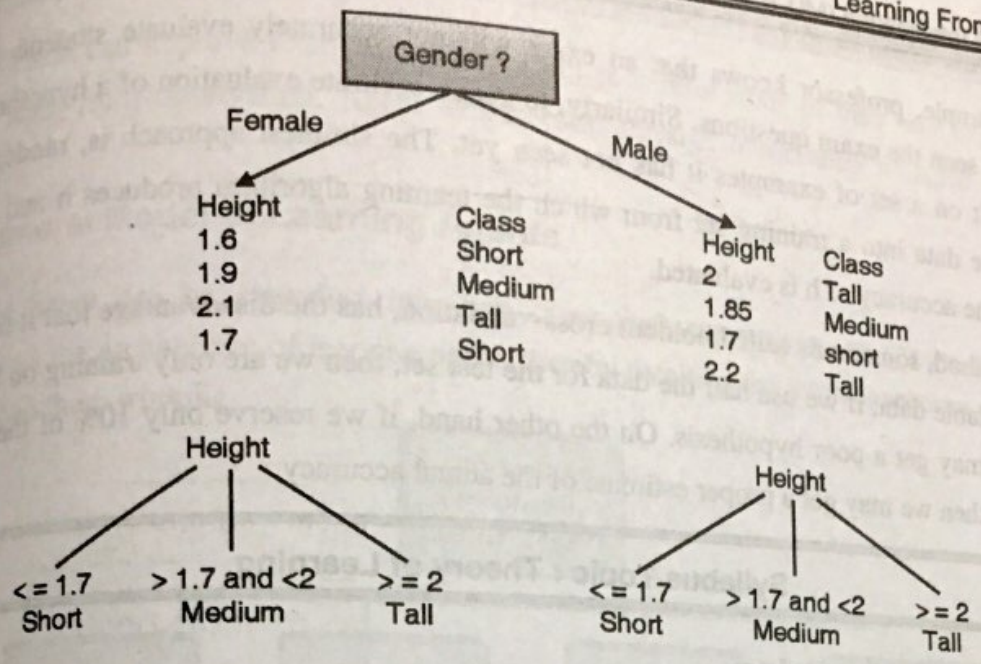
From Equations (2.3.1) and (2.3.2)

Same way we can go on reducing the entropy value till we get it as zero by dividing decision tree further.

Ex. 2.3.1 : Construct a decision tree for the following set of samples. Write any two decision rules obtained from the tree. Classify a new sample with (gender = " Female", height = "1.92m")(6 + 2 + 2)

Person ID	Gender	Height	Class
1	Female	1.6 m	Short
2	Male	2 m	Tall
3	Female	1.9 m	Medium
4	Female	2.1 m	Tall
5	Female	1.7 m	Short
6	Male	1.85 m	Medium
7	Female	1.6 m	Short
8	Male	1.7 m	Short
9	Male	2.2 m	Tall

Soln. :



Generated Rules

1. If Gender is Female or Male

   AND height >= 2

   THEN class is Tall

2. If Gender is Female or Male

   AND height <= 1.7

   THEN class is Short

Given gender = female Height = 1.92 Applying rule 2 class = Medium

Syllabus Topic : Evaluating and Choosing the Best Hypothesis

2.3.1 Evaluating and Choosing the Best Hypothesis

- A best hypothesis is the one that fits the future data best. To make that precise we need to define "future data" and "best." We make the stationary assumption : that there is a probability distribution over examples that remains stationary over time.
- Each example data point is a random variable that has an observed value and each example has an identical prior probability distribution. Examples that satisfy these assumptions are called independent and identically distributed. Such assumption connects the past to the future.
- The next step is to define "best fit." We define the error rate of a hypothesis as the proportion of mistakes it makes. Now, just because a hypothesis has a low error rate on the training set does not mean that it will generalize well.

- For example, professor knows that an exam will not accurately evaluate students if they have already seen the exam questions. Similarly, to get an accurate evaluation of a hypothesis, we need to test it on a set of examples it has not seen yet. The simplest approach is, randomly split the available data into a training set from which the teaming algorithm produces h and a test set on which the accuracy of h is evaluated.
- This method, sometimes called holdout cross-validation, has the disadvantage that it fails to use all the available data; if we use half the data for the test set, then we are only training on half the data, and we may get a poor hypothesis. On the other hand, if we reserve only 10% of the data for the test set, then we may get a proper estimate of the actual accuracy.

Syllabus Topic : Theory of Learning

2.4 Theory of Learning

- Generally, learning is based on experience. With learning we can enable the agent to perform same job more efficiently, while executing the same job next time.
- **Learning** is acquiring new knowledge or transforming the old knowledge or combining different types of information for the betterment of decision making process of an agent.
- **Formal definition for learning :** "An agent learns, based on the gained knowledge base K with respect to some set of actions A and performance measure P after performing an action A if performance measure P is improving then knowledge base K is updated, because of the improved experience in performing that action".

Given

- **A :** Action
- **P :** Performance Measure
- **K :** knowledge base
- **Goal :** Keep a generalized knowledge base which allows to improve performance on an action.
- An agent perceives its environment through its sensors and acts upon that environment through effectors/actuators.
- In computer science, agent is a software agent. Software agent allows user to take input from key strokes, mouse clicks, file content, etc and display output on screen. Software agent helps users in decision making and in performing computer related activities.

Agents which can improve their performance through careful study of their own experiences are said to learn from observation. Agent which learns through observation is called as **Learning Agent.**

2.4.1 General Model of Learning Agents

Learning agent can be classified into four theoretical components as shown in Fig. 2.4.1. Fig. 2.4.2 shows the architecture of learning agent, thereby detailing the interconnections among all the components and there working.

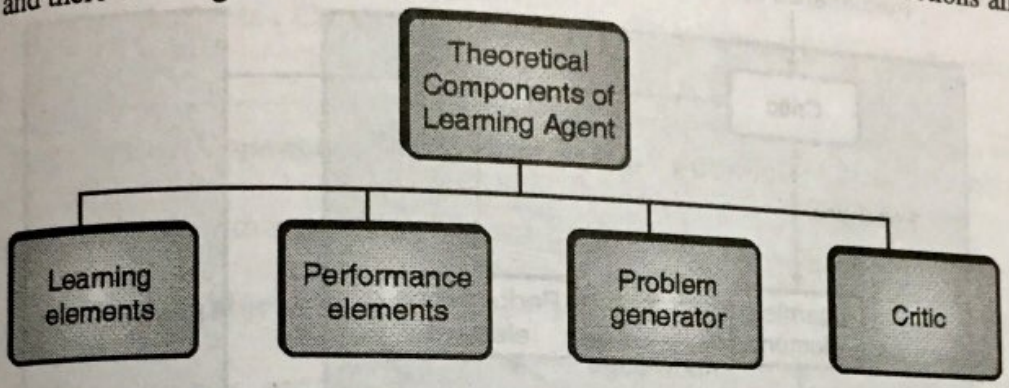


Fig. 2.4.1 : Theoretical components of Learning Agent

1. **Learning Elements**
   - Component which plays an important role in improvising an agent is called as **Learning Element.**
   - Learning element helps in improvising an agent by taking knowledge about performance element and feedback (we will learn about types of feedback in upcoming sections) from Critic.
   - After that it evaluates how to revise performance element for optimizing the results.
2. **Performance Elements**
   - Performance element is the main component of an agent like a Neuron in human body.
   - It takes input from the environment with the help of sensors and decides the appropriate action based on the interaction with learning element, and then it performs the action with the help of effectors.
3. **Problem Generator**
   - Problem generator experiments on performance element by suggesting actions which can generate new instances or experiences.
   - This is useful for training an agent further for better results.

➜ **4. Critic**

- Critic follows some basic performance standard after receiving input from the sensors (input can be a success or a failure).
- It compares this performance standard with input and based on the comparison. Critic gives feedback to the learning agent.

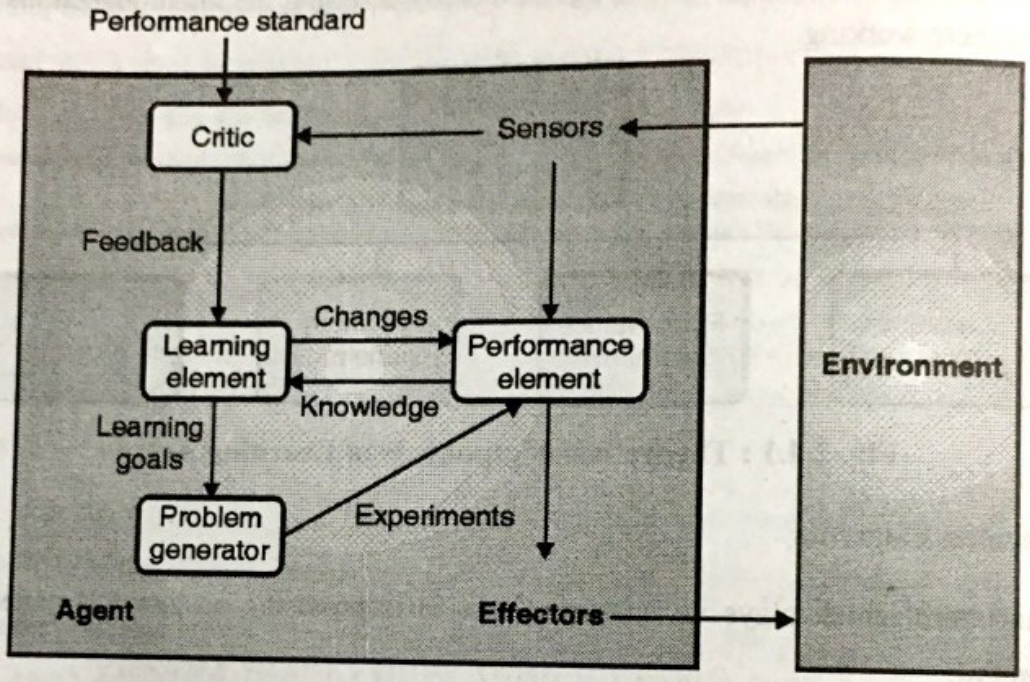


Fig. 2.4.2 : Learning Agent Architecure

Let's take on example of automated car agent, theoretical components of automated car learning agent can be given as follows :

➜ **1. Learning elements :** Create objective (e.g. Learn routing various places, Learn when to use break, accelerator, gear, etc.)

➜ **2. Performance element has information about the procedure for driving a car (e.g. actions like :** Turning with steering, accelerating to increase speed, changing gears, stop car with break, etc.)

➜ **3. Problem generator :** Trying different routes to travel is the work of problem generator.

➜ **4. Critic :** Observes surrounding environmental conditions and gives this information to the learning element. (e.g. check reaction of other drivers if there is slope on the road or if there is a speed breaker, or else if there is a quick left turn across two lanes of traffic).

Take another example of automated air conditioner. Theoretical components of automated air conditioner learning agent can be given as follows :

1. Learning elements
2. Performance element has information about the procedure for using a air conditioner
3. Problem generator
4. Critic

→ 1. **Learning elements :** Create objective (e.g. Learn when to when to switch temperature levels, etc.)

→ 2. **Performance element has information about the procedure for using a air conditioner** (e.g. actions like: how to switch temperature levels, etc.)

→ 3. **Problem generator :** Trying different temperature levels is the work of problem generator.

→ 4. **Critic :** Observes surrounding environmental conditions and gives this information to the learning element (e.g. check reaction of people if temperature levels are varied).

Syllabus Topic : Regression and Classification with Linear Models

2.5 Regression and Classification with Linear Models

- The set of linear functions with continuous input values is used for hundreds of years. Let's understand the regression with a uni-variate linear function, otherwise known as "fitting a straight line."
- Let us consider a univariate function with input x and output y. This has a form y=w1x-w0
- The coefficients are labelled as w because we consider they are the weights to be learned. The value of y is changed by changing the relative weight of one term or another. We'll define w to be the vector $[w_u, /w_i]$, and define $h_w(x)= w_i x$

Example

(1) Linear classifiers with a hard threshold
(2) Linear classification with logistic regression

→ **(1) Linear classifiers with a hard threshold**

- Linear functions can he used to do classification as well as regression. For example, Fig shows data points of two classes : earthquakes and underground explosions.

- Each point is defined by two input values, xi and x2, that refer to body and surface wave magnitudes computed from the seismic signal.
- Given these training data, the task of classification is to learn a hypothesis T2 that will take new (x1, x2) points and return either 0 for earthquakes or 1 for explosions.

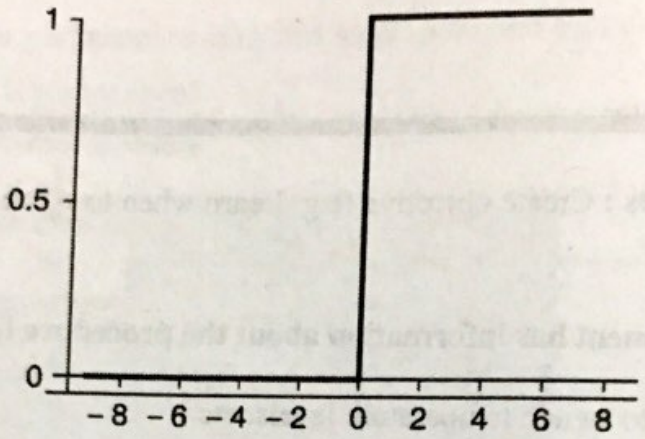


Fig. 2.5.1

(2) Linear classification with logistic regression

Problems with hard threshold :

1. The hypothesis h,(x) is not differentiable and is in fact a discontinuous function of its inputs and its weights; this makes learning with the perceptron rule a very unpredictable adventure.
2. The linear classifier always announces a completely confident prediction of 1 or 0, even for examples that are very close to the boundary; in many situations, we really need more gradated predictions.

Solution to these problems can be softening the threshold function i.e. approximating the hard threshold with a continuous, differentiable function. Although the two functions are very similar in shape, the logistic function

Logislic (z) = 1 +1e_z

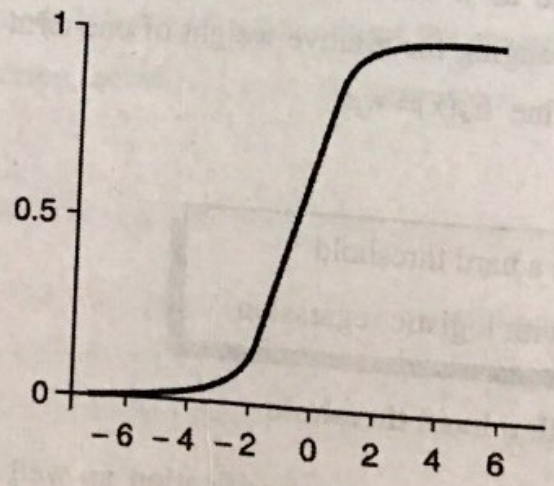


Fig. 2.5.2

2.6 Neural Networks

The basic computational unit in the nervous system is the nerve cell, or neuron. Human brain contains around 10^{11} neurons. Each neuron is connected to approximately 10^4 others. A neuron has:

Dendrites (inputs)

Cell body

Axon (output)

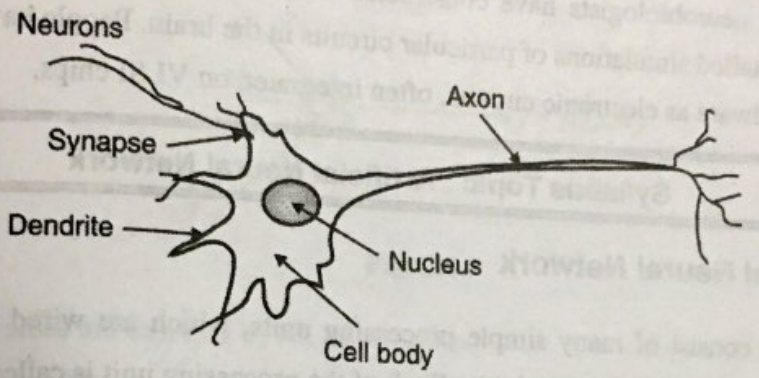


Fig. 2.6.1

...iments and evidences suggest that neurons receive, analyse and transmit information.

- A neuron receives input from other thousands of neurons. The information in transmitted in a form of electro-chemical signals (pulses).
- Once input exceeds a critical level; the neuron discharges a spike - an electrical pulse that travels from the body, down the axon, to the next neuron(s) or other receptors.
- When a neuron sends the information we say that a neuron 'fires'. This spiking event is also called depolarization.
- The axon endings (Output Zone) almost touch the dendrites or cell body of the next neuron. Transmission of an electrical signal from one neuron to the next is effected by neurotransmitters, chemicals which are released from the first neuron and which bind to receptors in the second.
- The receptors of a neuron are called synapses, and they are located on many branches, called dendrites. There are many types of synapses, but roughly they can be divided into two classes :

(i) Excitatory
(ii) Inhibitory

➜ (i) **Excitatory :** A signal received at this synapse 'encourages' the neuron to fire

➜ (ii) **Inhibitory :** A signal received at this synapse inhibits the neuron (as if asking to 'shut up')

- The neuron analyses all the signals received at its synapses. If most of them are 'encouraging', then the neuron gets 'excited' and fires its own message along a single wire, called axon. The axon may have branches to reach many other neurons.
- Computational neurobiologists have constructed very elaborate computer models of neurons in order to run detailed simulations of particular circuits in the brain. People have implemented model neurons in hardware as electronic circuits, often integrated on VLSI chips.

Syllabus Topic : Artificial Neural Network

2.6.1 Artificial Neural Network

- They typically consist of many simple processing units, which are wired together in a complex communication network as human brain. Each of the processing unit is called a s neuron.
- It consists of input signals, weights assigned to each of the input, processing function f which computes the summation of weighted inputs, and output signal. Basic structure of neuron is as shown in figure.

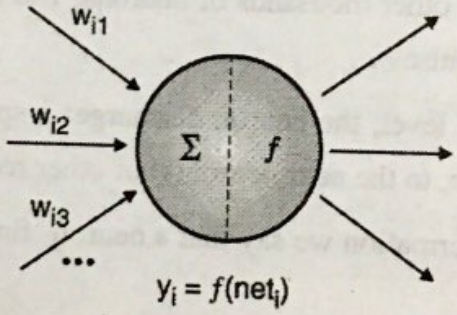


Fig. 2.6.2

2.6.2 Significance of NN in AI

- NN have the ability to learn from experience in order to improve their performance and to adapt themselves to changes in the environment, which resembles with human learning and inference patterns.
- In addition to that they are able to deal with incomplete information or noisy data and can be very effective especially in situations where it is not possible to define the rules or steps that lead to the solution of a problem as it is expected from an intelligent human to do.
- So as NN are mimicking activities of human brain, they have a significant application in artificial intelligence field. There are many AI applications like intelligent washing machine, Air

conditioner, driverless car which uses NN techniques to a complex level to develop and implement such real time systems.

2.6.3 A Model of a Single Neuron (Unit)

McCulloch and Pitts (1943) proposed the 'integrate and fire' model, which is named after him as McCulloch and Pitts model.

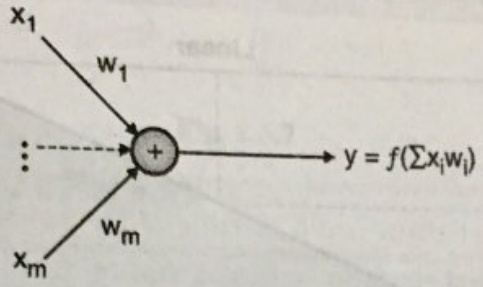


Fig. 2.6.3

The m input values are denoted by $x_1, x_2, \ldots, x_m$.

Each of the m inputs (synapses) has a weight $w_1, w_2, \ldots, w_m$.

The input values are multiplied by their weights and summed

$$v = w_1x_1 + w_2x_2 + \ldots + w_mx_m = \sum_{i=1}^{m} w_i x_i$$

The output is some function y = f(v) of the weighted sum.

Example 1: Let x = (0, 1, 1) and w = (1,–2, 4).

Then

$v = 1 \cdot 0 - 2 \cdot 1 + 4 \cdot 1 = 2$

2.6.4 Activation Function

The output of a neuron (y) is a function of the weighted sum y = f(v). This function is often called the activation function. There are different types of activation functions.

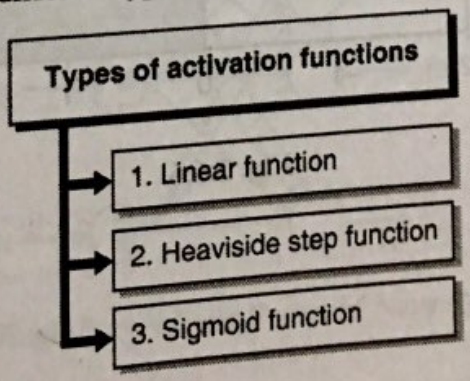


Fig. 2.6.4

1. Linear function

$$f(v) = a + v = a + \sum w_i x_i$$

where parameter a is called bias.

Noice that in this case, a neuron becomes a linear model with bias a being the intercept and the weights, $w_1, \ldots, w_m$, being the slopes.

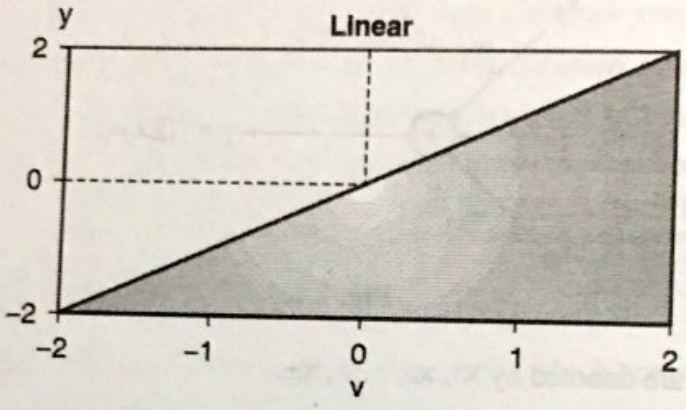


Fig. 2.6.5 : Linear Activation Function

2. Heviside step function

$$f(v) = \begin{cases} 1 & \text{if } v \geq a \\ 0 & \text{otherwise} \end{cases}$$

Here a is called the threshold

Example 2. If a = 0 and 0 = 2 > 0, then f(2) = 1, the neuron fires

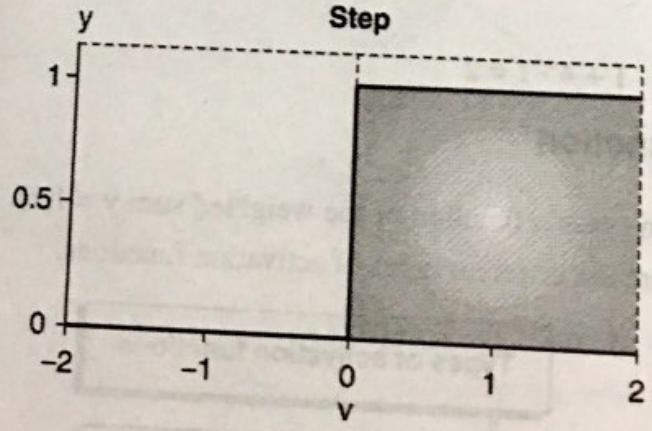


Fig. 2.6.6

3. Sigmoid function

$$f(v) = \frac{1}{1 + e^{-v}}$$

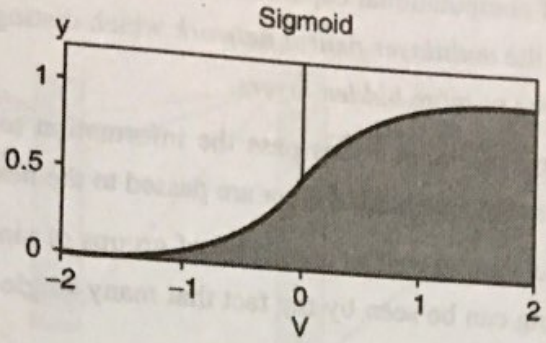


Fig. 2.6.7

2.6.5 Feed Forward Neural Network

- Following figure depicts a neural network consisting of single layer feed forward neural network. The most common structure of connecting neurons into a network is by layers. The simplest form of layered network is shown in figure.
- These are the networks without cycles or feedback loops, hence are called as feed-forward networks or perceptron.

The shaded nodes on the left are in the so-called *input layer*. The input layer neurons are to only pass and distribute the inputs and perform no computation. Thus, the only true layer of neurons is the one on the right. Each of the inputs ***x1,x2,...xN*** is connected to every artificial neuron in the output layer through the connection weight.

- Since every value of outputs ***y1,y2,...yN*** is calculated from the same set of input values, each output is varied based on the connection weights. Even if, the presented network is *fully connected*, the true biological neural network may not have all possible connections - the weight value of zero can be represented as ``no connection".

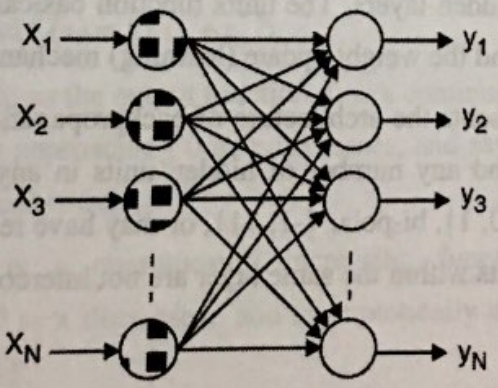


Fig. 2.6.8 : Single Layer Feed Forward Neural Network

- To achieve higher level of computational capabilities, a more complex structure of neural network is required. Figure shows the *multilayer neural network* which distinguishes itself from the single-layer network by having one or more *hidden layers*.
- In this multilayer structure, the input nodes pass the information to the units in the first hidden layer, then the outputs from the first hidden layer are passed to the next layer, and so on.
- Multilayer network can be also viewed as cascading of groups of single-layer networks. The level of complexity in computing can be seen by the fact that many single-layer networks are combined into this multilayer network.
- The designer of an artificial neural network should consider how many hidden layers are required, depending on complexity in desired computation.

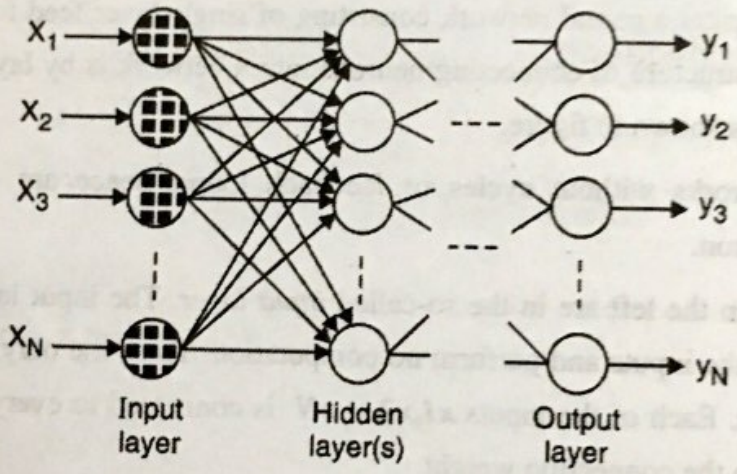


Fig. 2.6.9 : Multilayer Feed forward Network

2.6.6 Feedback ANNs

- Feedback networks, and multi layered perceptrons, in general, are feedforward networks with distinct input, output, and hidden layers. The units function basically like perceptrons, except that the transition (output) rule and the weight update (learning) mechanism are more complex.
- The figure on next page presents the architecture of backpropagation networks. There may be any number of hidden layers, and any number of hidden units in any given hidden layer. Input and output units can be binary {0, 1}, bi-polar {-1, +1}, or may have real values within a specific range such as [-1, 1]. Note that units within the same layer are not interconnected.

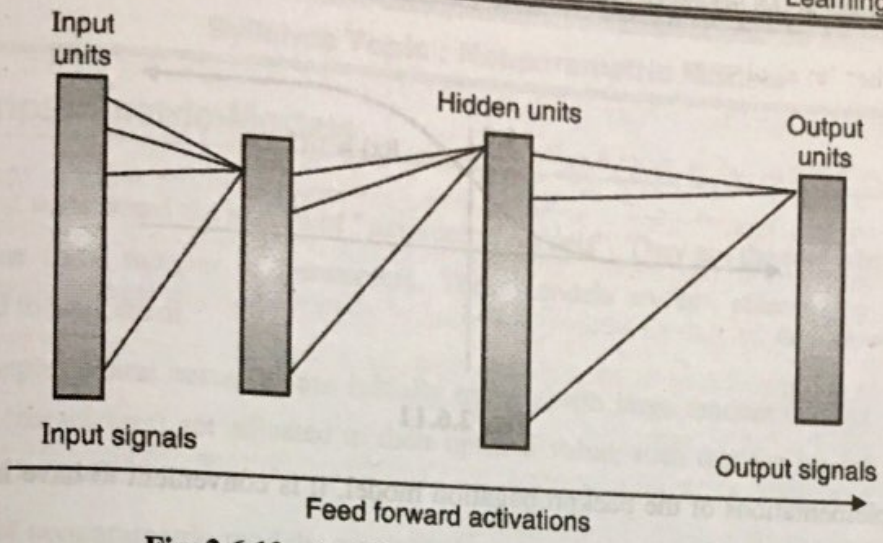


Fig. 2.6.10 : Architecture of feedback network

2.6.7 Error Back Propagation

- In feedforward activation, units of hidden layer 1 compute their activation and output values and pass these on to the next layer, and so on until the output units will have produced the network's actual response to the current input.

  The activation value ak of unit k is computed as follows.

  This is basically the same activation function of linear threshold units (McCulloch and Pitts model).

$$a_k = \sum_{i=1, n} W_{ik} \cdot X_i$$

- As illustrated above, x_i is the input signal coming from unit i at the other end of the incoming connection. wk_i is the weight of the connection between unit k and unit i. Unlike in the linear threshold unit, the output of a unit in a backpropagation network is no longer based on a threshold. The output y_k of unit k is computed as follows :

$$y_k = f(a_k) \text{ and } f(x) = 1 / (1 + e^{-x})$$

- The function f(x) is referred to as the output function. It is a continuously increasing function of the sigmoid type, asymptotically approaching 0 as x decreases, and asymptotically approaches 1 as x increases. At x = 0, f(x) is equal to 0.5.
- The output function f(x) is a continuosuly increasing function of the "sigmoid" type, asymptotically approaching 0 as x decreases, and asymptotically approaches 1 as x increases. At x equal to 0, f(x) is equal to 0.5.

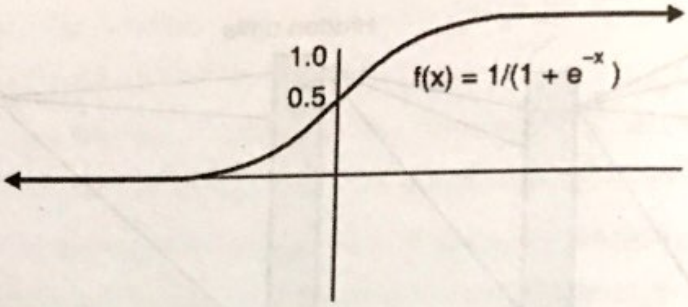


Fig. 2.6.11

- In some implementations of the backpropagation model, it is convenient to have input and output values that are bi-polar.
- In this case, the output function uses the hypertangent function, which has basically the same shape, but would be asymptotic to –1 as x decreases. This function has value 0 when x is 0.

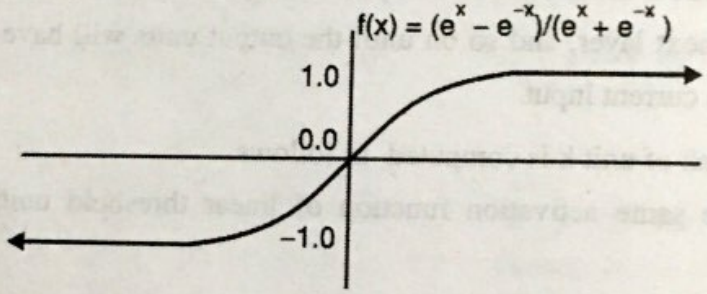


Fig. 2.6.12

- Once activation is fed forward all the way to the output units, the network's response is compared to the desired output ydi which accompanies the training pattern.
- There are two types of error. The first error is the error at the output layer. This can be directly computed as follows :

$$e_i = y_i^d - f(a_i)$$

- The second type of error is the error at the hidden layers. This cannot be computed directly since there is no available information on the desired outputs of the hidden layers. This is where the retropropagation of error is called for.

Syllabus Topic : Nonparametric Models

2.7 Nonparametric Models

- Let's first understand the notion of "parametric models". They are the one who summarize the data into some fixed number of parameters. These models are not affected by the amount of data provided to train them.
- For example, neural networks are initially trained with large amount of data. During training the weights (parameters) get adjusted to their optimal value, such that the model can be used for any new data thereafter.
- In case of nonparametric models, we don't have any fixed number of parameters to bind them by. The numbers of parameters grow, as number of examples used for training. This type of learning is called as instance based or memory based learning.
- The best example of instance based learning can be the table lookup; as the number of hypothesis increase, the lookup table consisting of event-action model increases. As the new input comes it has to search and match the event from lookup table; and take the corresponding action. If the current event is not listed in the table, it will probably return a default action.

There are various techniques to improvise the performance of table lookup. Following are few of them.

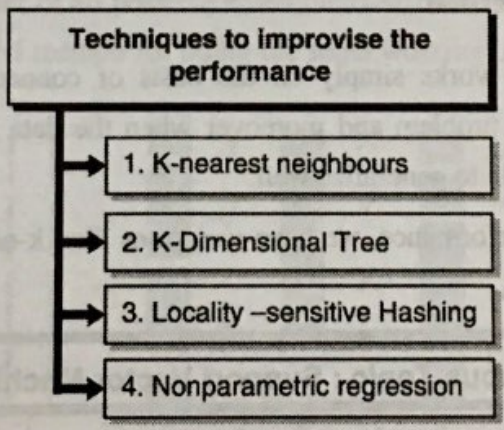


Fig. 2.7.1

➜ 1. K-nearest neighbours

- In this technique, instead of looking for exact match, we can look for the most matching neighbour. This is called K-nearest lookup.
- It does also suffer from problems like overfitting and underfitting. If k=1, it overreacts and tries to find out exact match, so it is called as overfitting. If the k value is large, it will never be able to find the logically similar neighbour, which leads to underfitting.

➜ 2. K-Dimensional Tree

- K-Dimensional tree is a balanced binary tree over data with an arbitrary number of dimensions is called a k-d tree. K-d trees are constructed same as we construct a balanced binary tree. The root value is the median of all the values given in the data set. The values less than the median will be arranged to the left of the root and the greater ones will go to the right side of the root.
- Finding nearest neighbour with K-D tree: In order to find the k nearest neighbours in k-d tree, we have to compare the query point with the median and decide the side for searching the neighbours.
- It may so happen that the chosen side does not have neighbours near to the query point, then we have to again search for the other side. K-d tree works well when we have few dimensions and more examples.

➜ 3. Locality – sensitive Hashing

- Hash tables are much faster than even binary tree. As definition, hash tables will look for exact match, but in order to find the near match we should put all the near points in the same bucket, which is termed as locality sensitive hash.
- To make this happen we can keep a threshold on the distance between any two points. If the distance is less than the threshold, there are high chances that those two points will be put in the same bin.

➜ 4. Nonparametric regression

- Nonparametric regression works simply on the basis of connect-the-dots approach. This will always lead to overfitting problem and moreover when the data is noisy, it will have spikes in between and will not be able to generalize well.
- In order to improve the performance, we have variations like, k-nearest neighbour regression and locally weighted regression.

Syllabus Topic : Support Vector Machines

2.8 Support Vector Machines

Support vector machines are the famous technique to be used in supervised learning environment. Following are the three main properties of SVM using which one can produce very good results without having domain knowledge.

1. SVM generalizes well by considering the farthest margins of decision boundaries for examples. Hence generates maximum margin separator.

2. SVM operates with high dimension linear separable environment to generate hyper planes. The data points which are non linear in two dimensional space are converted to multidimensional space, in order to make them linearly separable.
3. SVMs are non parametric method as they store all the training examples. Many times SVM just take number of examples same as the number of dimensions. They combine advantages of non parametric and parametric models. They can represent the complex problems but can avoid overfitting.

Syllabus Topic : Ensemble Learning

2.9 Ensemble Learning

- Generally in all learning techniques we consider only one hypothesis from the hypothesis space. In case of ensemble learning, multiple hypotheses are combined to vote for the prediction. The main motivation behind ensemble learning is to avoid misclassifications.
- For example, if we are considering 5 hypothesis voting for a classifying an item to class, even if two of them are wrongly classifying the item, still it will be put into the correct class if we are using simple majority voting.
- Another idea for using ensemble learning is to generate the hypothesis space. It can generate hypothesis space by as a set of all possible ensemble constructible hypothesis form original space. Boosting is the widely used method for doing the same.working of Boosting algorithm is depicted in the following figure.

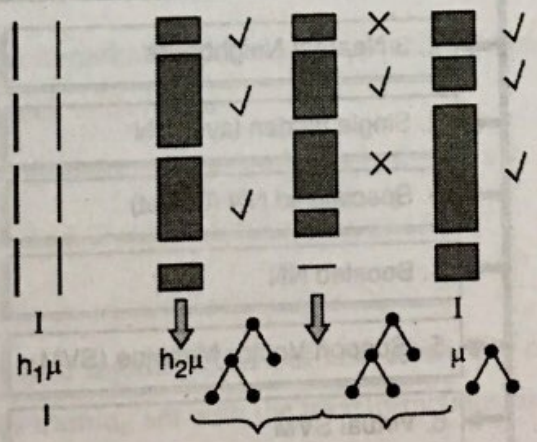


Fig. 2.9.1 : Working of Boosting Algorithm

- Each shaded rectangle corresponds to an example; the height of the rectangle corresponds to the weight.

- The checks and crosses indicate whether the example was classified correctly by the current hypothesis. The sire of the decision tree indicates the weight of that hypothesis in the final ensemble.

Syllabus Topic : Practical Machine Learning

2.10 Practical Machine Learning

As we have learned the basics of machine learning in this chapter, let us understand the practical aspects of ML through following case studies.

☞ Handwritten Digit Recognition

- In this case study lets understand how to select the algorithm best suited for the given example.
- Handwritten digit recognition is an important aspect of many applications. For example, sorting the posts by recognising the postal code numbers written on the post cards, reading cheques automatically, etc.
- A standardized dataset of 60,000 labelled digits data is developed by United States Institute of Science and Technology (NIST), so that multiple algorithms can be easily compared.
- Many techniques were applied i order to make the recognition more efficient and correct. Following are few of them :

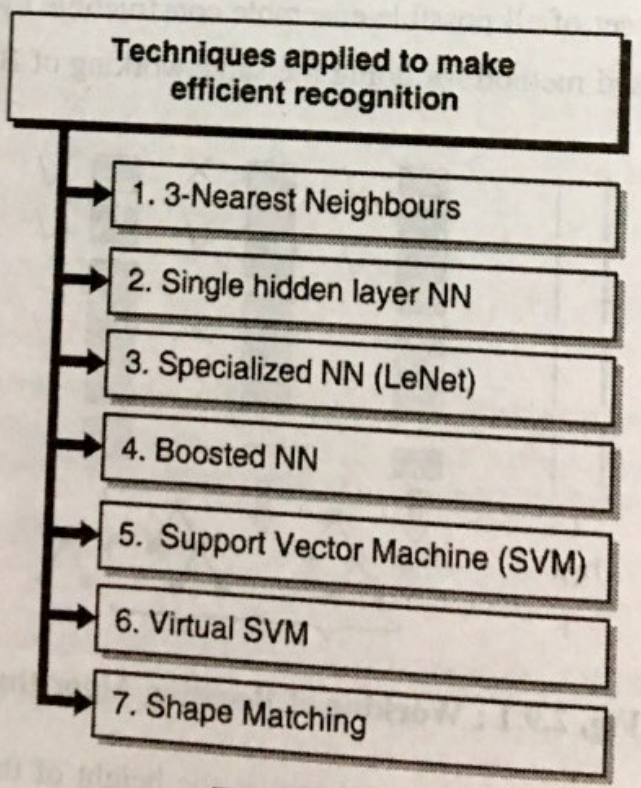


Fig. 2.10.1

1. 3-Nearest Neighbours

- It is the simplest techniques which come with the advantage of no training time. Due to which we need to store the complete dataset of 60,000 images in the main memory.
- In turn, it leads to low performance.

2. Single hidden layer NN

- A single hidden layer neural network was designed with 400 input units and 10 output units.
- A hidden layer of 300 units was observed to give best performance using cross validation.

3. Specialized NN (LeNet)

- LeNEt was the special NN designed to fit the input image size.
- The input layer was of 32 x 32 units. It has 3 hidden layers and 10 output units in the output layer.

4. Boosted NN

- Boosted Neural network was made by coping LeNet thrice. The second network was trained on mixed pattern having 50% wrongly detected inputs of first pattern.
- The third network was trained on patterns having disagreement in first two networks. Majority among the teree was given as output.

5. Support Vector Machine (SVM)

- SVM is said to be the remarkable technique as, it achieves nearly same performance as that of LeNet, which took years to develop.
- SVM does not use structure of the problem and performs well even when the pixels are permuted.

6. Virtual SVM

- Virtual SVM starts with regular SVM but also takes into consideration the structure of the problem. It combines training set with the transformations like LeNet.

7. Shape Matching

- Shape Matching is a technique from computer vision, which is used to align two parts of image.
- Training on only 20,000 images out of 60,000 using 100samples per digits could produce acceptable results.

Following is the table depicting performance and error rate for all the above mentioned techniques.

Sr. No.		3 NN	300 Hidden	LeNet	Boosted LeNet	SVM	Virtual SVM	Shape Match
1.	Error rate (pct.)	2.4	1.6	0.9	0.7	1.1	0.56 200	0.63
2.	Run time (millisec/digit)	1000	10	30	50	2000	200	-
3.	Memory requirements (Mbyte)	12	0.49	0.012	0.21	11	-	-
4.	Training time (days)	0	7	14	30	10	-	-
5.	% rejected to reach 0. 5% CHO	8.1	3.2	1.8	0.5	1.8	-	-

Review Questions

Q. Write a short note on : Supervised learning.

Q. Explain the working of a learning agent with example.

Q. Explain Decision Tree in brief.

Q. Explain the decision tree algorithm by using the example of an agent which needs to make a decision about "Whether to wait for a table" in a restaurant.

Q. What is decision tree? Hoe decision tree can be used for inference? Give suitable example.

Q. Explain decision tree learning with an example. What are decision rules? How to use it for classifying new sample?

Q. Explain general model of learning agents.

Chapter Ends...

CHAPTER 3

Unit III

Learning Probabilistic Models

Syllabus

Statistical Learning, Learning with Complete Data, Learning with Hidden Variables: The EM Algorithm. Reinforcement learning: Passive Reinforcement Learning, Active Reinforcement Learning, Generalization in Reinforcement Learning, Policy Search, Applications of Reinforcement Learning.

Syllabus Topic : Statistical Learning

3.1 Statistical Learning

- As in unit 2, we have data and hypothesis in this unit as well. Data indicates the instance or the complete data of a particular domain and hypothesis is probabilistic theory about how the model works in that domain.
- Lets us consider an example, assuming candy comes in two flavours; cherry and lime. The candies are wrapped in the same opaque wrapper, irrespective of flavour. The candy is sold in very large bags, of which there are known to be five kinds, again; indistinguishable from the outside.
- Given a new bag of candy, the random variable H for hypothesis denotes the type of the bag with following hypothesis:
- H1: 100% cherry, H2: 75% cherry + 25% lime, H3: 50% cherry + 50% lime, H4: 25% cherry + 75% lime, H5: 100% lime
- All the hypothesis are not known previously. As the candies are opened and inspected, data is revealed. Let us call it D_1, D_2, ..., D_N, where each D_i is a random variable with possible values of cherry and lime. The basic task faced by the agent is to predict the flavour of the next piece of candy.

- In Bayesian learning the probability of each hypothesis is calculated and predictions are made on the basis of given data. That is, the predictions are made by using all the hypotheses, weighted by their probabilities, rather than by using just a single "best" hypothesis.
- In this way, learning is reduced to probabilistic inference. Let D represent all the data, with observed value d; then the probability of each hypothesis is obtained by Bayes' rule:

$$P(h_i) = \alpha P(d\ h_i)P(h_i)$$

Now, suppose we want to make a prediction about an unknown quantity X. Then we have

$$P(X) = \sum P(X\ d, h_i)P(h_i, d) = \sum P(X\ h_i)P(hi\ d),$$

Where we have assumed that each hypothesis determines a probability distribution over X.

- *Bayesian prediction eventually agrees with the true hypothesis.* This is characteristic of Bayesian learning. For any fixed prior that does not rule out the true hypothesis, the posterior probability of any false hypothesis will, under certain technical conditions, eventually vanish. This happens simply because the probability of generating "uncharacteristic- data indefinitely is vanishingly small.
- The optimality of Bayesian learning comes at a price, of course. For real learning problems, the hypothesis space is usually very large and infinite.

Syllabus Topic : Learning with Complete Data

3.1.1 Learning with Complete Data

In this section we are discussing case, where we have complete data. Data is said to be complete when each data point contains values for every variable in the probability model being learned. The focus is on parameter learning by finding the numerical parameters for a probability model whose structure is fixed.

3.1.1.1 Maximum-likelihood Parameter Learning : Discrete models

In order to solve the problem with maximum likelihood parameter learning with discrete model, following are the three main steps :

1. Write down an expression for the likelihood of the data as a function of the parameter(s).
2. Write down the derivative of the log likelihood with respect to each parameter.
3. Find the parameter values such that the derivatives are zero.

The trickiest step is usually the last. In many cases one need to resort to iterative solution algorithms or other numerical optimization techniques to find the parameter required in step 3.

3.1.1.2 Naive Bayes Models

- Naive Bayes model is the most common Bayesian network model used in machine learning.
- In this model a tree structure is used to represent the variable and the class.The class variable is the root and the attribute variables are the braches. The model assumes that the attributes are conditionally independent of each other for given the class; hence called as naive model.Naive Bayes learning scales well to very large problems.
- A deterministic prediction can be obtained by choosing the most likely class. Naive Bayes learningsystems have no difficulty with noisy or missing data and can give probabilistic predictionswhen appropriate. Hence these models turn out to do surprisingly well in a wide range of applications.

3.1.1.3 Maximum-likelihood Parameter Learning : Continuous Models

- Continuous models basically deal with the continuous variables having ubiquitous nature of real world applications. The ubiquitous variables make it important to know how to learn the parameters of continuous models from data.
- The maximum-likelihood value of the mean is the sample average and the maximum likelihood value of the standard deviation is the square loot of the sample variance The principles for maximum likelihood learning are identical in the continuous and discrete cases.
- These produce comforting results that confirm 'commonsense" practice. Maximum likelihood learning is a simple procedure to be applied for continuous models, but it has some serious drawbacks with small size of data sets.

3.1.1.4 Bayesian Parameter Learning

- The Bayesian approach to parameter learning starts by defining a prior probability distribution over the possible hypotheses. It is also called as the hypothesis prior. Afterwards, as the real data arrives, the posterior probability distribution is updated.
- The entire Bayesian learning process can be formulated as an inference problem. New evidence nodes can be added to query the unknown nodes.
- This formulation of learning and prediction makes it clear that Bayesian learning requires no extra "principles of learning." Furthermore, there is, in essence, just one learning algorithm; the inference algorithm for Bayesian networks.

3.1.1.5 Learning Bayes Net Structures

- Till now we are assuming that the basic structure of Bayes net is given and we have to learn the parameters. Generally, the basic casual structure of the Bayes network is very easy to make, as it represents the basic knowledge about the domain.
- However, in few of the cases due to some disputable circumstances, it becomes difficult to form the basic model. In such situations, forming a good model can be challenging even for an expert.

Following can be the steps to build the basic model :

1. Start with model having no links.
2. Add parents for each node, fitting the parameters with the methods learned through 2.1to 2.4 and measuring the accuracy of the resulting model.
3. Or else one can start with an initial guess at the structure and use hill climbing or simulated annealing search in order to make modifications.
4. While modifying one can add, delete or reverse links.
5. Cycles should be avoided.
6. Proper ordering among the parent nodes and children node should be maintained.

- We have two methods to check whether the structure created is good or no. The first is to test whether the conditional independence assertions implicit in the structure are actually satisfied in the data.

3.1.1.6 Density Estimation with Nonparametric Models

- The task of nonparametric density estimation is typically done in continuous domains. In order to recover the model, we consider the k-nearest model first. Given a sample of data points, to estimate the unknown probability density at a query point x we can simply measure the density of the data points in the neighbourhood of x.
- The other method can be to use kernel function. To apply kernel model to estimate the density function, we have to assume that each data point generated its own little density function using Gaussian kernel. The average density becomes the estimated density at any given query point.

Syllabus Topic : Learning with Hidden Variables

3.1.2 Learning with Hidden Variables

- In the last section we learned how to deal with learning from completely visible data. But, many a times in reality it is not possible to have complete data which is visible, there might be many hidden parameters instead; called as latent variables.

- These variables are generally not visible in data but they are available for learning.
- For example, medical records of any patient will have all the details about the treatment, reports on various tests conducted and not only the disease predicted. The treatment itself leads to the prediction of disease.
- In case when the disease is not observed, and if try to proceed without the name of the disease. In such case we can make use of latent variables, which are indicative attributes for prediction.
- Theses latent variable in general, drastically reduce the number of parameters required for prediction, and in turn reduce the size of dataset required to train the network.
- In order to learn conditional distribution in case of hidden variables, we have an algorithm called, Expectation-maximization (EM) algorithm. Let see how EM solves the problem of conditional distribution.

Syllabus Topic : The Expectation Maximization (EM) Algorithm

3.1.2.1 The Expectation Maximization (EM) Algorithm

- The basic idea of EM algorithm is to pretend that the parameters are known to us and inferring probability of all the data points related to each other. Then the model is refit to calculate the actual probabilities over multiple iterations, till convergence.

In short, we are completing the data by inferring probability distribution over hidden variables.

The two steps of EM algorithms are as follows:

1. E-step	2. M-step

➜ **1. E-Step**

In this step the probabilities $p_i = P(C = i \text{ xi})$ s the probability that the datum X_i was generated by component i. Using this one can calculate the number of data points currently assigned to component i.

➜ **2. M- Step**

In this step the new mean, co-variance, and component weights are calculated. Maximisation step can be seen as computing the expected values for hidden indicator variables. This step finds out the new value of parameters that maximise the log likelihood of the data basis on the expected values of hidden indicator variables.

- EM algorithm has many applications in learning problems. Especially in case of unsupervised clustering, when the classes are to be identified from the given data; EM works upto the mark.

There are two Limitations however, to EM Algorithm

1. EM works similar to hill climbing methods by simply choosing the log likelihood of the data at every iteration. This can lead to the same problem of local maxima or minima.
2. The log likelihood for the fully learned model is slightly greater than that of the original model, from which the data is generated. This will because the data is generated randomly and might not match exactly with the original model.

Syllabus Topic : Reinforcement Learning

3.2 Reinforcement Learning

- In case of reinforcement learning, agent doesn't get to know the correct action at every stage as in case of supervised learning.
- Instead, the agent has to learn itself by recognising the rewards or reinforcement and punishments.
- Reward is nothing but a sign of positive move and punishment is the indication of bad action being taken by the agent.
- For example, while learning to play the chess game, if the checkmate happen for the opponent, then the agent should know that it's a good thing for him and if the checkmate is happening for the agent then it's a bad thing.
- In such games the feedback can only be provided at the end of the game. In some other cases there can be a prompt feedback for each of the action taken by agent. For example, taxi driving agent, ping-pong game, etc.
- In a nutshell, reinforcement can be thought as if you are playing a game which you don't know how to play and after hundreds of steps the opponent announces "you lost the game".
- The main task of reinforcement agent is to use the rewards for optimally designing the strategy of taking actions by learning the policies of task environment. In many cases, it's almost impossible to give desired output for each event that can occur while working in the environment.
- For example, in game playing or taxi driving, having set all the possible moves and counter moves is next to impossible as one cannot just think for it.
- In such cases reinforcement learning becomes the only effective solution using which the agent can learn the environment policy and act effectively in order to achieve the goal.
- Reinforcement can be thought of encompassing all of AI wherein agent has to learn the environment by being in the environment.

Syllabus Topic : Passive Reinforcement Learning

3.2.1 Passive Reinforcement Learning

- To make it simple for understanding, let's take an example of a fully observable environment, where the agent uses state based representation. In passive learning, the agent's policy is fixed
- i.e. in every state, after executing the action, it tries to learn how good the policy is thereby learns the utility function.
- Passive reinforcement does not specifies any transition model for the agent, i.e. agent doesn't know where he will be after executing a particular action.
- Nor does he know the reward function, specifying the reward for a particular state. All that he has to do is to learn the policy evaluation.
- In reinforcement learning, the agent has to execute number of trials using its policy. In each trial, he starts from initial state and reaches to the goal state basis on the rewards received at every state after taking actions.
- Each percept is provided with reward. The aim is to learn the expected utility for each state. The utility is defined by the sum of all the rewards obtained.

.1.1 Direct Utility Estimation

In 1950, Widow and Hoff had invented a simple method called direct utility estimation. The basic idea behind direct utility estimation is the utility of a state is the expected rewards from that state onwards, called as "rewards-to-go" and each trial provides the estimation for each of the state.

- At the end, the algorithm calculates the observed rewards –to – go for each state simple by measuring the running average and update the estimated utility of that state accordingly.
- In the run of infinitely many trials, the estimated average will turn out to be the real one. This strategy of direct utility estimations brings it down to supervised learning problem where state is the input and the observed rewards to go is the output.
- While reducing the reinforcement learning problem to an inductive learning problem it misses out the important fact that the states are not independent of each other but there is lot of dependency in terms of the utility value of the successor state. Hence the algorithm converges slowly.

3.2.1.2 Adaptive Dynamic Programming

- An Adaptive Dynamic Programming (ADP) agent studies the dependencies among the utilities thereby learning the transition model that connects them.

- They use dynamic programming to solve Markov decision problems. For this, they study the learned transition model and observe the rewards in order to calculate the utility for the states.
- The mode does only change slightly after each observation, the value iteration process can use the previous utility estimates as initial values and converges quickly.
- As the environment is fully observable, the learning becomes very easy. In ADP we get a transition table with probabilities, having input state and resulting state.

Algorithms for Adaptive Dynamic Learning Agent

```
function PASSIVE-ADP-AGENT(percept) returns an action
inputs: percept, a percept indicating the current state a' and reward signal r'
    persistent: 7r, a fixed policy
    mdp, an MDP with model P, rewards R, discount -y
    U, a table of utilities, initially empty
    N_s a table of frequencies for state—action pairs, initially zero
    Ns,i a table of outcome frequencies given state—action pairs, initially zero
    s, a, the previous state and action, initially null
if s'is new then U[s'] ← r'; R[s'l r'
if s is not null then
    increment N_sa.[s, a] and N_s'|sa[s' , s,
    for each t such that N_sa[t,s, a] is nonzero do
    P(t a, a) al / N_sa[s, a]
U ← POLICY-EVALUATION(n, U,mdp)
if s'.TERMINAL? then s, null else a, a ← a',
return a
```

A passive reinforcement learning agent based on adaptive dynamic program-ming. The POLICY-EVALUATION function solves the fixed-policy Hellman equations

3.2.1.3 Temporal Different Learning

- Temporal difference model works on the basis of adjusting utilities to the equilibrium that holds locally when the utility estimates are correct.
- It does not require the transition model to update the utilities a sin case of ADP. Instead observed transitions are used to supply the connections between neighbouring states.

Algorithm for passive Temporal Difference Learning Agent

```
function PASSIVE-TD-AGENT(percept) returns an action
inputs: percept, a percept indicating the current state s' and reward signal r'
        persistent 7r, a fixed policy
U, a table of utilities, initially empty
N, a table of frequencies for states, initially zero
s, a, r, the previous state, action, and reward, initially null
if s' is new then U[s'] r'
if a is not null then
increment N,[8]
U[s] – U[s] + a(N,[a])(r + y U[s'] – U[a])
if st.TERMINAL? then s, a, r null else a, a, r s', r'
return a
```

A passive reinforcement learning agent that learns utility estimates using tem-poral differences.

- Both ADP and TD try to make local adjustments to estimate utility values of states.
- One difference is that TD adjusts a state to agree with its observed successors, whereas ADP adjusts a state to agree to all its successor states weighted by their probabilities.
- This difference disappears when, the effects of TD adjustments are averaged out over all the transitions.
- The other prominent difference is that while TD makes one adjustment per observation of transition, ADP makes many as it requires to restore consistency between the utility estimates and the environment model.

Syllabus Topic : Active Reinforcement Learning

3.3 Active Reinforcement Learning

- Unlike the policy derived by passive reinforcement learning for determining the behaviour of the, active reinforcement learning decides what action to be taken next.
- The active learning agent learns the complete model with outcome probabilities for all actions. In this case agent has choice of action.
- They use Bellman equations to calculate utilities to define optimal policy.

- After obtaining the optimal utility function, the agent can just extract an optimal action by looking ahead one step and the expected utility can be maximized.

3.3.1 Exploration

- As the greedy agent ignores the actions and only focuses on the rewards given to the current learned model, the learned model varies from the actual model a lot.Agent needs to improve the model in order to receive greater rewards in the future.
- In order to achieve the same the trade-off between exploration, to obtain long term well being and exploitation, to obtain current rewards as per the current utility, must be made by the agent. BY only exploiting, agents risks into getting stuck into the rut, while only exploration to improve one's knowledge is of no use if one never puts that knowledge into practice, by learning the current utility values of the actions and states.
- In the real time problems, tere has to a continuous decision to be made regarding continuing in a comfortable existence and striking out into the unknown in the hopes of discovering a new and better life. Having greater understanding requires less exploration.
- The problem of developing an optimal strategy for the required amount of exploration is known as the **bandit problems** in of statistical decision theory.
- They are extremely difficult to solve in order to obtain an optimal exploration method. However, it is possible to come up with a reasonable scheme that will eventually lead to optimal behaviour of the agent.
- Ideally, such scheme needs to be **greedy in the limit of infinite exploration (GLIE)**. As the name suggests, GLIE scheme must try each action in each state an unbounded number of times to avoid having a finite probability that an optimal action is missed because of an unusually bad series of outcomes.
- An ADP agent using such a scheme will eventually learn the true environment model.
- Using GLIE technique,the agent's actions become optimal with respect to the learned and hence the true model.
- There are multiple GLIE schemes.
- One of the simplest is to have the agent choose a random action a fraction 1/t of the time and to follow the greedy policy otherwise.
- But this poly would be extremely slow as it converges to an optimal model.
- A more sensible approach would be to assign weight to the actions that the agent has not tried very often, while avoiding actions that are believed to be of low utility.

- This can be implemented by assigning a higher utility estimate to relativelyunexplored state-action pairs.
- Essentially, this amounts to an optimistic prior over the possible environments and causes the agent to behave initially as if there were wonderful rewardsscattered all over the place.

3.3.2 Learning an Action-Utility Function

- There is an alternative method called Q-learning. It learns action – utility model instead of learning state utilities. Q values are directly related to utility values of a state.
- Q learning is a model free method as it does not need any model for learning or for action selection.

Algorithm for an exploratory Q-learning agent

function Q-LEARNING-AGENT(pereept) returns an action

inputs: *percept,* a percept indicating the current state s' and reward signal r'

persistent: Q, a table of action values indexed by state and action, initially zero

N," a table of frequencies for state—action pairs, initially zero

a, e, r, the previous state, action, and reward, initially null

if TERmiNAL?(s) then $Q[s. None]$

if s is not null then

increment $N_{sa}[s, a]$

$Q/s, Q Is, al + c,(N_a [s,a])$(r max.' Q15', a'] Qs, al)

s, a, r s', $\text{argmax}_a f(Q Is', al N[s, a), r'$

return a

- Q-learning agent is an active learner. It learns the value $Q(s, n)$ of each action for every situation.
- The use of exploration function is same as that of ADP, but it does not learn the transition model as in case of other agents, rather it uses the Q value which can be directly related to its neighbours. Also it need not pay any attention to the policy used because of Q value usage.
- For the same reason, it is also called as "off-policy learning algorithm". It can behave better even if the policies are random or adversarial.
- As a effect it learns about what will actually happen rather than learning what the agent wants to happen.

- Due to these properties, Q-learning methods are one of the most successful methods even in games such as checkers, chess, backgammon, etc.; where the evaluation function is learned by a model based on Q-learning.

Syllabus Topic : Generalization in Reinforcement Learning

3.4 Generalization in Reinforcement Learning

- The algorithms using utility function or Q-function provide one output value for input sequence. Such approaches work well for a small state space problems. As the state space increases in size the time for convergence and time per iteration increases rapidly.
- ADP and TD methods can handle up to 10,000 states or more, which can be the case of 2-dimensional maze problems; representing tiny world.
- The other real time problems in general can be handled better by using function approximations. In this technique we any type of representation but lookup tables.
- In this case the generalized or just an approximation and might not be the case that the true utility function or Q-function can be represented in the chosen form.
- Function approximation makes it practically possible to represent utility function s for very large state space. Not only that but it also allows the learning agent to generalize from states it has visited to states it has not visited.
- This is the beauty of function approximation using which; we can have inductive generalization over input states with less memory space. But on the other side, for find a good approximation, a larger hypothesis space is required; in turn the convergence will be delayed.
- With the use of function approximator, the agent is supposed to learn faster with an assumption that the given hypothesis is moderate in size.
- Also the hypothesis must have some function as that fit reasonably well to the true utility function.
- Function approximation can also be very helpful for learning a model of the environment. In case of fully observable environment, the learning model is supervised learning problem as the next state is nothing but the outcome of the previous action. In case of partially observable environment however, the learning problem is tougher.
- In such case, to make the agent learn a model of environment, we need to know the casual relationship among the hidden variables and the relation with other variables so that the structure of a dynamic Bayesian network.

Syllabus Topic : Policy Search

3.5 Policy Search

- The final approach we will consider for reinforcement learning problems is called policy search. The simplest approach followed by this technique is to keep on twiddling the policy till we get the required performance.
- Each Q-function could be a linear function or it could be a nonlinear function such as a neural network. Policy search will then adjust the parameters to improve the policy.
- Notice that if the policy is represented by Q functions, then policy search results in a process that learns Q-functions. This process is not the same as Q-learning, however. In Q-learning with function approximation, the algorithm finds a value of theta such that Qa is "close" to Q_v, the optimal Q-function.
- Policy search, on the other hand, finds a value of parameters that results in grind performance; the values found by the two methods may differ very substantially.
- When the environment or the policy is stochastic, that is randomly changing with time, things get more difficult. Suppose we are trying to do hill climbing, which requires comparing p(theta) and $p(e + A_O)$ for some small A_O.

  The problem is that the total reward on each trial may vary widely, so estimates of the policy value from a small number of trials will be quite unreliable; trying to compare two such estimates will be even more unreliable.
- One solution is simply to run lots of trials, measuring the sample variance and using it to determine that enough trials have been run to get a reliable indication of the direction of improvement for *p(theta)*.
- Unfortunately, this is impractical for many real problems where each trial may be expensive, time-consuming, and perhaps even dangerous.

Syllabus Topic : Applications of Reinforcement Learning

3.6 Applications of Reinforcement Learning

We consider applications in game playing, where the transition model is known and the goal is to learn the utility function, and in robotics, where the model is usually unknown :

1. Game Playing
2. Application to Robot Control

➜ 1. Game Playing

- The checkers program written by Arthur Samuel (1959, 1967) is the first significant application of reinforcement learning.
- It was also the first significant learning program of any kind. The learning model was modified later on and there are significant differences in his program and the current methods, however.
- According to the strategy by Arthur, initial weights were modified by taking the difference between the current state and backed-up values which are generated by the full look ahead in the search tree. This strategy works well, as it takes into account the view of complete state space at a different granularity.
- A second difference was that the program did not use any observed rewards. It ignores the values of terminal states reached in self-play.
- This leads to a possibility that Samuel's program will not converge, or will converge on a strategy designed to lose rather than to win. He managed to avoid this fate by misting that the weight for material advantage should always be positive.
- On the other side, the TD-GAMMON game project was an attempt to learn from self-play alone. The reward signal was given only at the end of each game. A fully connected neural network with a single hidden layer of 40 nodes was used to represent the evaluation function.
- TD-GAMMON learned to play considerably better, even with the input of raw board positions and no pre-computed features. The network took 20000 training games and two weeks of computer time to train itself, which leads to generate only a vanishingly small fraction of the state space.
- After adding the pre-computed features to the input representation, a network with 80 hidden nodes was trained with 300,000 training games, to reach a standard of play comparable to that of the top three human players worldwide.
- Kit Woolsey, a top player and analyst, said in appraisal that "There is no question in my mind that its positional, judgment is far better than mine."

➜ 2. Application to Robot Control

- In this application of reinforcement learning, we have the problem of balancing pendulum which is placed on top of a moving car. The cart can jerk to left or right.
- This is called as cart-pole balancing problem or inverted pendulum. The set up for the problem is as shown in the figure.

- The problem is to keep the pole roughly upright by balancing x, which is the distance between the cart and the reference wall.
- The cart-pole problem differs from the problems described earlier in that the state variables *x*, 0, and θ are continuous. There are discrete actions; jerk left or jerk right, also called as the **bang-bang control** regime.
- The earliest work on learning for this problem was carried out by Michie and Chambers (1968), They could develop the BOXES algorithm which was able to balance the pole for over an hour only after about 30 trials. Moreover, this algorithm was implemented with a real cart and pole; and not simulated as many other subsequent systems.

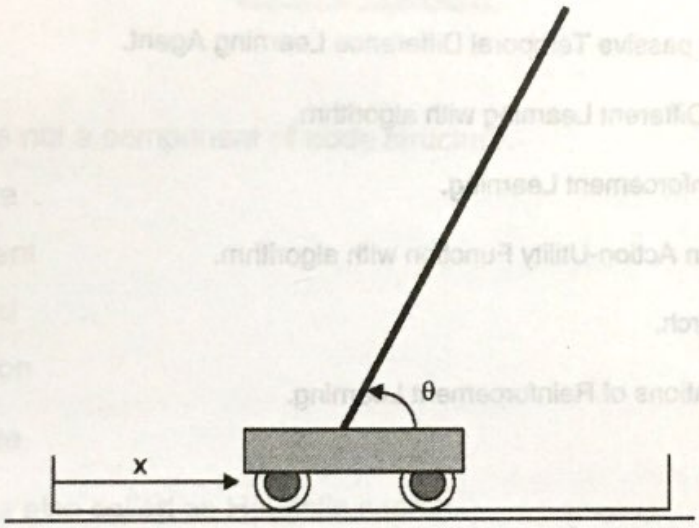


Fig. 3.6.1 : steup for balancing a longpole on top of a moving car

- The algorithm first divided the four-dimensional state space into boxes; hence the name. Then the trials are run till either the pole fell over or the cart hit the wall. Negative reinforcement was associated with the final action in the final box and then propagated hack through the sequence.
- It was observed that the position of apparatus used in training caused discretization problem; as the position were different from the training set.
- This leads to the conclusion that, generalization was not perfect. Improved generalization and faster learning can be obtained using an algorithm that adaptively partitions the state space according to the observed variation in the reward, or by using a continuous-state, nonlinear function approximator such as a neural network. Nowadays, balancing a triple inverted pendulum is a common exercise.

Review Questions

Q. Explain Learning with Complete Data

Q. Explain steps of EM algorithms.

Q. Explain Reinforcement Learning in details.

Q. What do you mean by Passive Reinforcement Learning?

Q. Write Algorithms for Adaptive Dynamic Learning Agent.

Q. Explain Adaptive Dynamic Programming with algorithm.

Q. Write Algorithm for passive Temporal Difference Learning Agent.

Q. Explain Temporal Different Learning with algorithm.

Q. Explain Active Reinforcement Learning.

Q. Explain Learning an Action-Utility Function with algorithm.

Q. Explain Policy Search.

Q. Write down Applications of Reinforcement Learning.

Chapter Ends...

❑❑❑

Appendix A

Solved University Question Paper of Oct. 2018

Artificial Intelligence

Oct. 2018

Q. 1(a) Attempt All **(5 Marks)**

1 _____ is not a component of node structure.

(a) State

(b) Parent

(c) Child

(d) Action

Ans. : State

2. _____ is also called as Heuristic search.

(a) Uninformed search

(b) Informed search

(c) Depth Limited search

(d) Uniform cos search

Ans. : Informed search

3. _____ agent does not maintain internal state.

(a) Model based

(b) Goal based

(c) Simple reflex

(d) Utility based

Ans. : Simple reflex

4. If a hypothesis agrees with all the data, it is called as _____.

(a) Consistent hypothesis

(b) Integral hypothesis

(c) Best hypothesis

(d) Regular hypothesis

Ans. : Consistent hypothesis

5. The most widely used ensemble method is called _____.

(a) Bayesian Learning

(b) Online learning

(c) Boosting

(d) Support vector machine

Ans. : Support vector machine

Q. 1(b) Fill in the blanks. **(5 Marks)**

(Decision List, omniscient, Single, Regularization, Parameter Learning)

1. A decision tree returns a _____ output value.

Ans. : Single

2. _____ is finding the numerical parameters for a probability model whose structure is fixed.

Ans. : Parameter Learning

3. This process of explicitly penalizing complex hypothesis is called _____.

Ans. : Regularization

4. _____ agent knows the actual outcome of its actions and can act accordingly.

Ans. : omniscient

5. _____ consists of series of tests, each of which is a conjunction of literals.

Ans. : Decision List

Q. 1(c)1 What is early stopping ?

Ans. : **(1 Mark)**

Early stopping attempts to remove the need to manually set the value for number of epochs. It can also be considered a type of regularization method in that it can stop the network from over fitting.

The idea behind early stopping is relatively simple :

- Split data into training and test sets
- At the end of each epoch (or, every N epochs) :
  - evaluate the network performance on the test set
  - if the network outperforms the previous best model : save a copy of the network at the current epoch
- Take as final model the model that has the best test set performance.

Q. 1(c)2 Define Error Rate. **(1 Mark)**

Ans. :

Error rates refer to the frequency of errors occurred, defined as "the ratio of total number of data units in wrongly classified to the total number of data units." As the error rate increases, the data network reliability decreases.

Q. 1(c)3 How denote learning rate ? **(1 Mark)**

Ans. :

The amount that the weights are updated during training is referred to as the step size or the "learning rate." Specifically, the learning rate is a configurable hyper parameter used in the training of neural networks that has a small positive value, often in the range between 0.0 and 1.0.

Q. 1(c)4 Define decision boundary. **(1 Mark)**

Ans. :

In a statistical-classification problem with two classes, a decision boundary or decision surface is a hyper surface that partitions the underlying vector space into two sets, one for each class. The classifier will classify all the points on one side of the decision boundary as belonging to one class and all those on the other side as belonging to the other class. A decision boundary is the region of a problem space in which the output label of a classifier is ambiguous.

If the decision surface is a hyper plane, then the classification problem is linear, and the classes are linearly separable. Decision boundaries are not always clear cut. That is, the transition from one class to another is not discontinuous, but gradual.

Q. 1(c)5 What is triangle inequality ? **(1 Mark)**

Ans. :

The single most important inequality in analysis is the triangle inequality. The triangle inequality concerns distance between points and says that the straight line distance between and is less than the sum of the distances from to and from to. It is very much part of our everyday intuition about distances and easy to remember. It is, however, very useful.

For all real numbers x and y, $|x + y| \leq |x| + |y|$.

Q. 2(a) Describe Model based agent. **(Section 1.8.2)** **(5 Marks)**

Q. 2(b) What is PEAS ? Mention it for Part picking robot and Medical Diagnosis system. **(Section 1.7.2)** **(5 Marks)**

Q. 2(c) Explain Artificial Intelligence with Turing Test approach. **(Section 1.2.1)** **(5 Marks)**

Q. 2(d) Describe problem formulation of vacuum world problem. **(Sections 1.10.1 and 1.10.3)** **(5 Marks)**

Q. 2(e) Explain these properties of task environment. **(Section 1.7)** **(5 Marks)**

1. Deterministic vs. Stochastic.
2. Fully observable vs. partially observable.

Q. 2(f) List and explain the categories of definition of AI. **(Section 1.2)** **(5 Marks)**

Q. 3(a) Explain the concept of Locality Sensitive Hashing. **(5 Marks)**

Ans. :

Locality sensitive hashing :

Locality sensitive hashing (LSH) is one such algorithm. LSH has many applications, including :

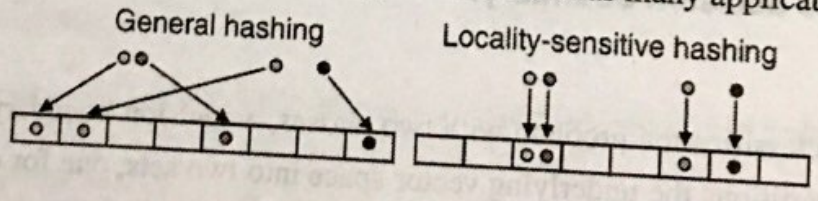


Fig. 1 – Q. 3(a)

- **Near-duplicate detection :** LSH is commonly used to deduplicate large quantities of documents, webpages, and other files.
- **Genome-wide association study :** Biologists often use LSH to identify similar gene expressions in genome databases.
- **Large-scale image search :** Google used LSH along with Page Rank to build their image search technology Visual Rank.
- **Audio/video fingerprinting :** In multimedia technologies, LSH is widely used as a fingerprinting technique A/V data.

LSH refers to a family of functions (known as LSH families) to hash data points into buckets so that data points near each other are located in the same buckets with high probability, while data points far from each other are likely to be in different buckets. This makes it easier to identify observations with various degrees of similarity.

The general idea of LSH is to find a algorithm such that if we input signatures of 2 documents, it tells us that those 2 documents form a candidate pair or not i.e. their similarity is greater than a threshold. Remember that we are taking similarity of signatures as a proxy for Jaccard similarity between the original documents.

Specifically for min-hash signature matrix :

- Hash columns of signature matrix M using several hash functions
- If 2 documents hash into same bucket for at least one of the hash function we can take the 2 documents as a candidate pair.

Now the question is how to create different hash functions. For this we do band partition.

Band partition :

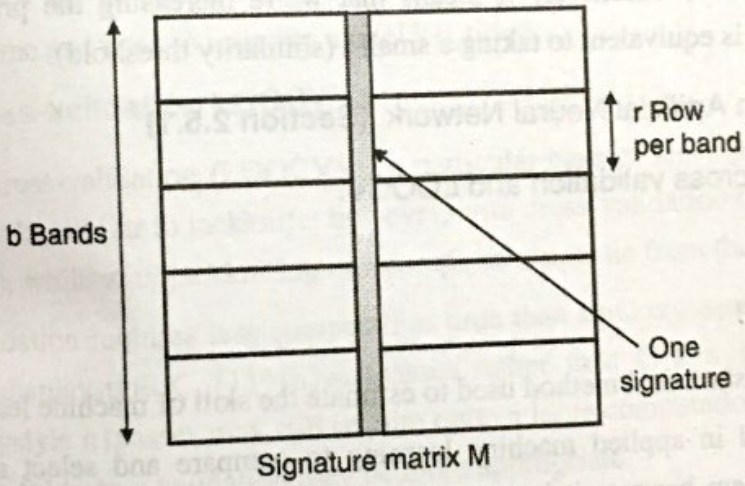


Fig. 2 – Q. 3(a)

Here is the algorithm :

- Divide the signature matrix into b bands, each band having r rows.
- For each band, hash its portion of each column to a hash table with k buckets.
- Candidate column pairs are those that hash to the same bucket for at least 1 band.
- Tune b and r to catch most similar pairs but few non similar pairs.

There are few considerations here. Ideally for each band we want to take k to be equal to all possible combinations of values that a column can take within a band. This will be equivalent to identity matching. But in this way k will be a huge number which is computationally infeasible. For e.g. If for a band we have 5 rows in it. Now if the elements in the signature are 32 bit integers then k in this case will be $(2^{32})^5 \sim 1.4615016e + 48$. You can see what's the problem here. Normally k is taken around 1 million.

The idea is that if 2 documents are similar then they will appear as candidate pair in at least one of the bands.

Choice of b & r :

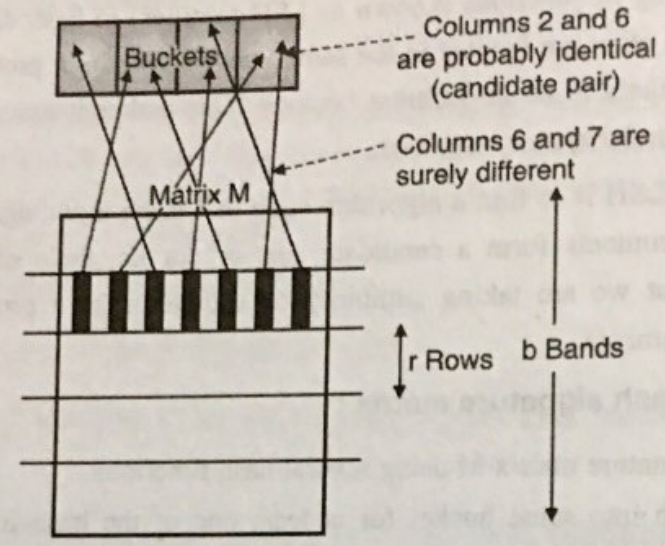


Fig. 3 – Q. 3(a)

If we take b large i.e more number of hash functions, then we reduce r as b*r is a constant (number of rows in signature matrix). Intuitively it means that we're increasing the probability of finding a candidate pair. This case is equivalent to taking a small t (similarity threshold).

Q. 3(b) Write a note on Artificial Neural Network. **(Section 2.6.1)** **(5 Marks)**

Q. 3(c) Explain K-fold cross validation and LOOCV. **(5 Marks)**

Ans. :

K fold cross validation :

Cross-validation is a statistical method used to estimate the skill of machine learning models.

It is commonly used in applied machine learning to compare and select a model for a given predictive modeling problem because it is easy to understand, easy to implement, and results in skill estimates that generally have a lower bias than other methods techniques for assessing how the results of a statistical analysis will generalize to an independent data set. It is mainly used in settings where the goal is prediction, and one wants to estimate how accurately a predictive model will perform in practice. In a prediction problem, a model is usually given a dataset of known data on which training is run (training dataset), and a dataset of unknown data (or first seen data) against which the model is tested (called the validation dataset or testing set). The goal of cross-validation is to test the model's ability to predict new data that was not used in estimating it, in order to flag problems like over fitting or selection bias and to give an insight on how the model will generalize to an independent dataset.

Cross-validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data. That is, to use a limited sample in order to estimate how the model is expected to perform in general when used to make predictions on data not used during the training of the model.

It is a popular method because it is simple to understand and because it generally results in a less biased or less optimistic estimate of the model skill than other methods, such as a simple train/test split.

The general procedure is as follows :

- Shuffle the dataset randomly.
- Split the dataset into k groups
- For each unique group :
  - Take the group as a hold out or test data set
  - Take the remaining groups as a training data set
  - Fit a model on the training set and evaluate it on the test set
  - Retain the evaluation score and discard the model
- Summarize the skill of the model using the sample of model evaluation scores.

Importantly, each observation in the data sample is assigned to an individual group and stays in that group for the duration of the procedure. This means that each sample is given the opportunity to be used in the hold out set 1 time and used to train the model k-1 times.

Leave-one-out cross-validation LOOCV :

Leave-one-out cross-validation (LOOCV) is a particular case of leave-p-out cross-validation with p = 1. The process looks similar to jackknife; however, with cross-validation one computes a statistic on the left-out sample(s), while with jackknifing one computes a statistic from the kept samples only.

LOO cross-validation requires less computation time than LpO cross-validation because there are only C 1 n = n {\displaystyle C_{1}^{n}=n}passes rather than C k n {\displaystyle C_{k}^{n}} However, n {\displaystyle n}passes may still require quite a large computation time, in which case other approaches such as k-fold cross validation may be more appropriate.

k-fold cross-validation [edit] :

In k-fold cross-validation, the original sample is randomly partitioned into k equal sized subsamples. Of the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining k − 1 subsamples are used as training data. The cross-validation process is then repeated *k* times, with each of the *k* subsamples used exactly once as the validation data. The k results can then be averaged to produce a single estimation. The advantage of this method over repeated random sub-sampling (see below) is that all observations are used for both training and validation, and each observation is used for validation exactly once. 10-fold cross-validation is commonly used, but in general *k* remains an unfixed parameter.

For example, setting k = 2 results in 2-fold cross-validation. In 2-fold cross-validation, we randomly shuffle the dataset into two sets d_0 and d_1, so that both sets are equal size (this is usually implemented by shuffling the data array and then splitting it in two). We then train on d_0 and validate on d_1, followed by training on d_1 and validating on d_0.

It is a popular method because it is simple to understand and because it generally results in a less biased or less optimistic estimate of the model skill than other methods, such as a simple train/test split.

The general procedure is as follows :

- Shuffle the dataset randomly.
- Split the dataset into k groups
- For each unique group :
  - Take the group as a hold out or test data set
  - Take the remaining groups as a training data set
  - Fit a model on the training set and evaluate it on the test set
  - Retain the evaluation score and discard the model
- Summarize the skill of the model using the sample of model evaluation scores.

Importantly, each observation in the data sample is assigned to an individual group and stays in that group for the duration of the procedure. This means that each sample is given the opportunity to be used in the hold out set 1 time and used to train the model k-1 times.

Leave-one-out cross-validation LOOCV :

Leave-one-out cross-validation (LOOCV) is a particular case of leave-p-out cross-validation with p = 1. The process looks similar to jackknife; however, with cross-validation one computes a statistic on the left-out sample(s), while with jackknifing one computes a statistic from the kept samples only.

LOO cross-validation requires less computation time than LpO cross-validation because there are only C 1 n = n {\displaystyle C_{1}^{n}=n}passes rather than C k n {\displaystyle C_{k}^{n}} However, n {\displaystyle n}passes may still require quite a large computation time, in which case other approaches such as k-fold cross validation may be more appropriate.

k-fold cross-validation [edit] :

In k-fold cross-validation, the original sample is randomly partitioned into k equal sized subsamples. Of the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining k − 1 subsamples are used as training data. The cross-validation process is then repeated *k* times, with each of the *k* subsamples used exactly once as the validation data. The k results can then be averaged to produce a single estimation. The advantage of this method over repeated random sub-sampling (see below) is that all observations are used for both training and validation, and each observation is used for validation exactly once. 10-fold cross-validation is commonly used, but in general *k* remains an unfixed parameter.

For example, setting k = 2 results in 2-fold cross-validation. In 2-fold cross-validation, we randomly shuffle the dataset into two sets d_0 and d_1, so that both sets are equal size (this is usually implemented by shuffling the data array and then splitting it in two). We then train on d_0 and validate on d_1, followed by training on d_1 and validating on d_0.

Q. 3(d) Write a note on supervised Learning. **(5 Marks)**

Ans. :

Learning agent and teacher both take input from surrounding environment. Teacher has the desired answer key for the given problems.

Whereas, learning agent processes the given input and gets an output. This actual output is subtracted from the desired output by the adder, which shows the error in the actual output.

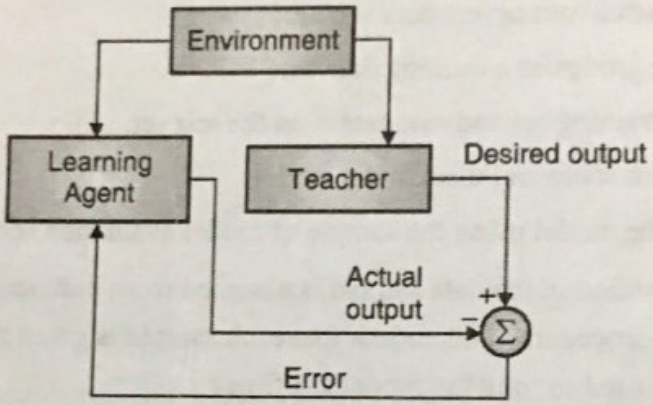


Fig. 1-Q. 3(d) : Supervised Learning

This error is given as input to the learning agent, so that it can learn from this error and while generating output for the next time it can try to remove that error.

As per the name Supervised Learning is performed under supervision of a Teacher. Teacher can be an agent/system which has a correct answer for each example. This answer can be a variable in numeric form, a categorical variable, etc.

Let's takeone example where Learning Agent has to identify living and nonliving things. Supervised Learning will have correct answers see labels shown in Fig. 2-Q. 3(d).

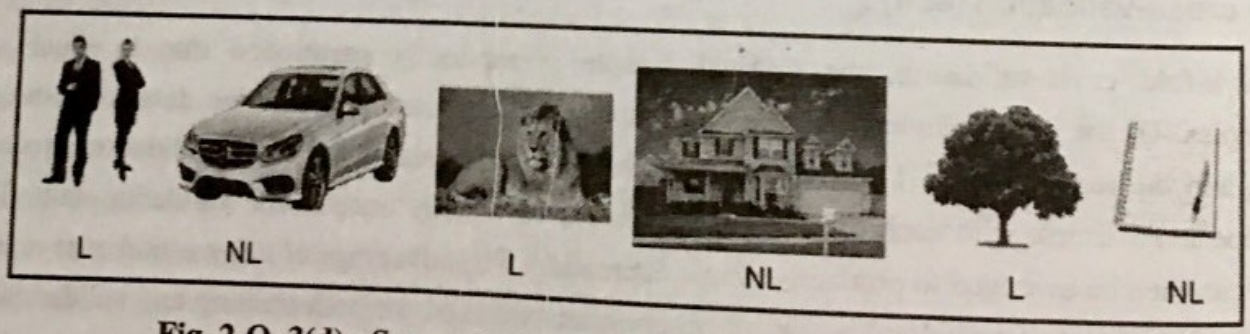


Fig. 2-Q. 3(d) : Supervised Learning Agent with correct answer

In short, the way in real life teacher guides a student to get better results; supervised learning method guides learning agent with the help of teacher to get better results.

Q. 3(e) What is entropy ? How do we calculate it ? **(Section 2.3)** **(5 Marks)**

Q. 3(f) Write a note on Nearest Neighbor model. **(5 Marks)**

Ans. :

K-Nearest Neighbors (K-NN) is one of the simplest machine learning algorithms. When a new situation occurs, it scans through all past experiences and looks up the k closest experiences. Those experiences (or: data points) are what we call the k nearest neighbors.

Like other machine learning techniques, it was inspired by human reasoning. For example, when something significant happens in your life, you memorize that experience and use it as a guideline for future decisions. When a new situation occurs, it scans through all past experiences and looks up the k closest experiences. Those experiences (or: data points) are what we call the k nearest neighbors.

If you have a classification task, for example you want to predict if the glass will break, you take the majority vote of all k neighbors. If k = 5 and in 3 or more of your most similar experiences the glass broke, it will go with the prediction "yes, it will break".

All the computation happens during scoring, i.e. when you apply the model on unseen data points. You need to determine which k data points out of our training set are closest to the data point we want to get a prediction for;

Let's say that our data points look like the following :

$X_{11}, x_{12}, x_{13}...x_{1m}, y_1$:

$X_{n1}, x_{n2}, x_{n3}...x_{nm}, y_n$

We have a table of n rows and $m+1$ columns where the first m columns are the attributes we use to predict the remaining label column (also known as "target"). For now, let's also assume that all attribute values x are numerical, while the label values for y are categorical, i.e. we have a classification problem.

$D(S, Xj) = sqrt\{(S1-Xj1)^2+...+(Sm-Xjm)^2\}$

We can now define a distance function which calculates the distance between data points. It should find the closest data points from our training data for any new point. The Euclidean distance is often a good choice for a distance function if the data is numerical. If our new data point has attribute values s1 to sm, we can calculate the distance d(s, xj) between point s to any data point xj by

The k data points with the closest value for this distance become our k neighbors. For a classification task, we now use the most frequent of all values y from our k neighbors. For regression tasks, where y is numerical, we use the average of all values y from our k neighbors. The k data points with the closest value for this distance become our k neighbors. For a classification task, we now use the most frequent of all values y from our k neighbors. For regression tasks, where y is numerical, we use the average of all values y from our k neighbors.

Q. 4(a) Explain the concept of Passive Reinforcement Learning. **(Section 3.2.1)** **(5 Marks)**

Q. 4(b) Write a note on Statistical Learning. **(Section 3.1)** **(5 Marks)**

Q. 4(c) Explain Hidden Markov Model. **(5 Marks)**

Ans. :

Hidden Markov Model (HMM) is a statistical Markov model in which the system being modeled is assumed to be a Markov process with unobservable (i.e. hidden) states.

The hidden Markov model can be represented as the simplest dynamic Bayesian network. The mathematics behind the HMM were developed by L. E. Baum and coworkers. HMM is closely related to earlier work on the optimal nonlinear filtering problem by Ruslan L. Stratonovich, who was the first to describe the forward-backward procedure.

In simpler Markov models, the state is directly visible to the observer, and therefore the state transition probabilities are the only parameters, while in the hidden Markov model, the state is not directly visible, but the output (in the form of data or "token" in the following), dependent on the state, is visible. Each state has a probability distribution over the possible output tokens. Therefore, the sequence of tokens generated by an HMM gives some information about the sequence of states; this is also known as pattern theory.

The adjective hidden refers to the state sequence through which the model passes, not to the parameters of the model; the model is still referred to as a hidden Markov model even if these parameters are known exactly.

Hidden Markov models are especially known for their application in reinforcement learning and temporal pattern recognition such as speech, handwriting, recognition, part, musical score following, partial and bioinformatics.

A hidden Markov model can be considered a generalization of a mixture model where the hidden variables (or latent variables), which control the mixture component to be selected for each observation, are related through a Markov process rather than independent of each other. Recently, hidden Markov models have been generalized to pair wise Markov models and triplet Markov models which allow consideration of more complex data structures and the modeling of non-stationary data.

Q. 4(d) Briefly explain the concept of direct utility estimation. **(Section 3.2.1.1)** **(5 Marks)**

Q. 4(e) What are the applications of Reinforcement Learning ? **(Section 3.2)** **(5 Marks)**

Q. 4(f) Explain the concept of EM algorithm. **(Section 3.1.2.1)** **(5 Marks)**

Q. 5(a) Explain Breadth First Search strategy along with its pseudo code. **(Section 1.11.2)** **(5 Marks)**

Q. 5(b) Write a note on decision Tree. Also describe its pruning technique. **(Section 2.3)** **(5 Marks)**

Q. 5(c) Explain Naive Bayes Model. **(Section 3.1.1.2)** **(5 Marks)**

Q. 5(d) Explain the concept of Goal Based Agent. **(Section 1.8.3)** **(5 Marks)**

Q. 5(e) Write a note on over fitting in decision tree. **(5 Marks)**

Ans. :

Decision Tree – Over fitting :

Over fitting is a significant practical difficulty for decision tree models and many other predictive models. Over fitting happens when the learning algorithm continues to develop hypotheses that reduce training set error at the cost of an increased test set error. There are several approaches to avoiding over fitting in building decision trees.

- Pre-pruning that stop growing the tree earlier, before it perfectly classifies the training set.
- Post-pruning that allows the tree to perfectly classify the training set, and then post prune the tree.
- Practically, the second approach of post-pruning over fit trees is more successful because it is not easy to precisely estimate when to stop growing the tree.
- The important step of tree pruning is to define a criterion be used to determine the correct final tree size using one of the following methods :

1. Use a distinct dataset from the training set (called validation set), to evaluate the effect of post-pruning nodes from the tree.
2. Build the tree by using the training set, then apply a statistical test to estimate whether pruning or expanding a particular node is likely to produce an improvement beyond the training set.
   - Error estimation
   - Significance testing (e.g., Chi-square test)
3. Minimum Description Length principle: Use an explicit measure of the complexity for encoding the training set and the decision tree, stopping growth of the tree when this encoding size (size (tree) + size(misclassifications(tree)) is minimized.
4. The first method is the most common approach. In this approach, the available data are separated into two sets of examples: a training set, which is used to build the decision tree, and a validation set, which is used to evaluate the impact of pruning the tree. The second method is also a common approach.

❑❑❑

Artificial Intelligence

Oct. 2018

Q. 1(a) Attempt All (Each of 5 Marks) **(15 Marks)**

1. _____ is not a component of node structure.
   - (a) State
   - (b) Parent
   - (c) Child
   - (d) Action

2. _____ is also called as Heuristic search.
   - (a) Uninformed search
   - (b) Informed search
   - (c) Depth Limited search
   - (d) Uniform cos search

3. _____ agent does not maintain internal state.
   - (a) Model based
   - (b) Goal based
   - (c) Simple reflex
   - (d) Utility based

4. If a hypothesis agrees with all the data, it is called as _____.
   - (a) Consistent hypothesis
   - (b) Integral hypothesis
   - (c) Best hypothesis
   - (d) Regular hypothesis

5. The most widely used ensemble method is called _____.
   - (a) Bayesian Learning
   - (b) Online learning
   - (c) Boosting
   - (d) Support vector machine.

Q. 1(b) Fill in the blanks.

(Decision List, omniscient, Single, Regularization, Parameter Learning)

1. A decision tree returns a _____ output value.
2. _____ is finding the numerical parameters for a probability model whose structure is fixed.
3. This process of explicitly penalizing complex hypothesis is called _____.
4. _____ agent knows the actual outcome of its actions and can act accordingly.
5. _____ consists of series of tests each of which is a conjunction of literals.

Q. 1(c) Short Answers

1. What is early stopping ?
2. Define Error Rate.
3. How denote learning rate ?
4. Define decision boundary.
5. What is triangle inequality ?

Q. 2 Attempt the following (Any Three) (Each of 5 Marks) (15 Marks)

(a) Describe Model based agent.

(b) What is PEAS ? Mention it for Part picking robot and Medical Diagnosis system.

(c) Explain Artificial Intelligence with Turing Test approach.

(d) Describe problem formulation of vacuum world problem.

(e) Explain these properties of task environment.

1. Deterministic vs. Stochastic.
2. Fully observable vs. partially observable.

(f) List and explain the categories of definition of AI.

Q. 3 Attempt the following (Any Three) (Each of 5 Marks) (15 Marks)

(a) Explain the concept of Locality Sensitive Hashing.

(b) Write a note on Artificial Neural Network.

(c) Explain K-fold cross validation and LOOCV.

(d) Write a note on supervised Learning.

(e) What is entropy ? How do we calculate it ?

(f) Write a note on Nearest Neighbor model.

Q. 4 Attempt the following (Any Three) (Each of 5 Marks) **(15 Marks)**

(a) Explain the concept of Passive Reinforcement Learning.

(b) Write a note on Statistical Learning.

(c) Explain Hidden Markov Model.

(d) Briefly explain the concept of direct utility estimation.

(e) What are the applications of Reinforcement Learning ?

(f) Explain the concept of EM algorithm.

Q. 5 Attempt the following (Any Three) (Each of 5 Marks) **(15 Marks)**

(a) Explain Breadth First Search strategy along with its pseudo code.

(b) Write a note on decision Tree. Also describe its pruning technique.

(c) Explain Naive Bayes Model.

(d) Explain the concept of Goal Based Agent.

(e) Write a note on over fitting in decision tree.

❑❑❑