□□□

1 INTRODUCTION

Syllabus

Introduction to Artificial Intelligence, Foundations of Artificial Intelligence, History of Artificial Intelligence, State of the Art, Risks and Benefits of AI, Intelligent Agents, Agents and Environments, Good Behavior : Concept of Rationality, Nature of Environments, Structure of Agents.

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.

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

Q. Explain Turing test designed for satisfactory operational definition of AI.

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.

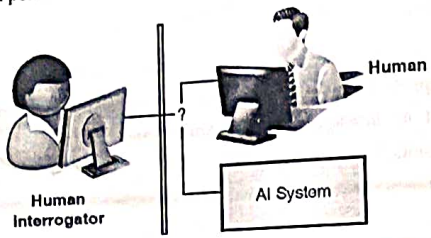


Fig. 1.2.1 : Turing Test Environment

- 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.

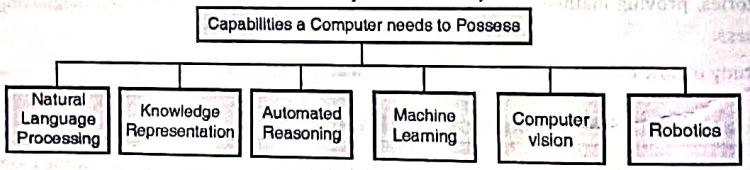


Fig. 1.2.2 : Capabilities a Computer needs to possess

- 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 : *"AI ... 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.

1.3 Categorization of Intelligent Systems

University Question

Q. Explain different definitions of Artificial Intelligence according to different categories. **SPPU : Dec. 19, 5 Marks**

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 :

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.

1.4 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 : (Refer Fig. 1.4.1)

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

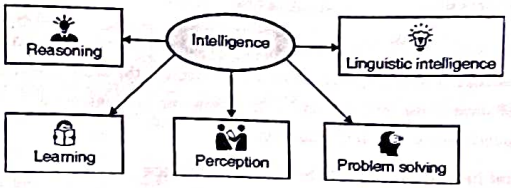


Fig. 1.4.1 : Components of AI

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.

1.4.1 Computational Intelligence vs. Artificial Intelligence

Sr. No.	Computational Intelligence (CI)	Artificial Intelligence (AI)
1.	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.
2.	CI involves numbers and computations.	AI involves designs and symbolic knowledge representations.
3.	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.
4.	CI concentrates on low level cognitive function implementation.	AI concentrates of high level cognitive structure design.

1.5 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 :**
  - **Game - Checkers :** Computer played as an opponent,
  - **Education - Algebra :** For solving word problems,
  - **Education - Math :** Proving logical theorems,
  - **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 :
  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.

1.5.1 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.

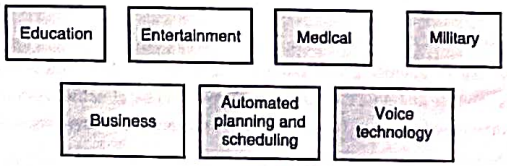


Fig. 1.5.1 : Fields of AI Application

- Fig. 1.5.1 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.

1.6 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. 1.6.1 shows these areas in Intelligent Systems.

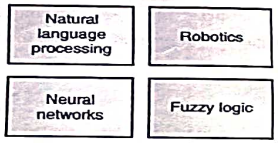


Fig. 1.6.1 : Sub-areas in Intelligent Systems

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!").

1.6.1 Benefits of Artificial Intelligence

1) Reduction in Human Error :

- The phrase **"human error"** was born because humans make mistakes from time to time. Computers, however, do not make these mistakes if they are programmed properly. With Artificial intelligence, the decisions are taken from the previously gathered information applying a certain set of algorithms. So errors are reduced and the chance of reaching accuracy with a greater degree of precision is a possibility.
- **Example :** In Weather Forecasting using AI they have reduced the majority of human error.

2) Takes risks instead of Humans :

- This is one of the biggest advantages of Artificial intelligence. We can overcome many risky limitations of humans by developing an AI Robot which in turn can do the risky things for us. Let it be going to mars, defuse a bomb, explore the deepest parts of oceans, mining for coal and oil, it can be used effectively in any kind of natural or man-made disasters.
- **Example :** Have you heard about the **Chernobyl** nuclear power plant explosion in Ukraine? At that time there were no AI-powered robots that can help us to minimize the effect of radiation by controlling the fire in early stages, as any human went close to the core was dead in a matter of minutes. They eventually poured sand and boron from helicopters from a mere distance.
- AI Robots can be used in such situations where intervention can be hazardous.

3) Available 24 × 7 :

- An Average human will work for 4-6 hours a day excluding the breaks. Humans are built in such a way to get some time out for refreshing themselves and get ready for a new day of work and they even have weekly offed to stay intact with their work-life and personal life. But using AI we can make machines work 24 × 7 without any breaks and they don't even get bored, unlike humans.
- **Example :** Educational Institutes and Helpline centers are getting many queries and issues which can be handled effectively using AI.

4) **Helping in Repetitive Jobs :**

- In our day-to-day work, we will be performing many repetitive works like sending a thanking mail, verifying certain documents for errors and many more things. Using artificial intelligence we can productively automate these mundane tasks and can even remove "boring" tasks for humans and free them up to be increasingly creative.
- **Example :** In banks, we often see many verifications of documents to get a loan which is a repetitive task for the owner of the bank. Using AI Cognitive Automation the owner can speed up the process of verifying the documents by which both the customers and the owner will be benefited.

5) **Digital Assistance :**

- Some of the highly advanced organizations use digital assistants to interact with users which saves the need for human resources. The digital assistants also used in many websites to provide things that users want. We can chat with them about what we are looking for. Some chatbots are designed in such a way that it's become hard to determine that we're chatting with a chatbot or a human being.
- **Example :** We all know that organizations have a customer support team that needs to clarify the doubts and queries of the customers. Using AI the organizations can set up a Voice bot or Chatbot which can help customers with all their queries. We can see many organizations already started using them on their websites and mobile applications.

6) **Faster Decisions :**

- Using AI alongside other technologies we can make machines take decisions faster than a human and carry out actions quicker. While taking a decision human will analyze many factors both emotionally and practically but AI-powered machine works on what it is programmed and delivers the results in a faster way.
- **Example :** We all have played Chess games in Windows. It is nearly impossible to beat CPU in the hard mode because of the AI behind that game. It will take the best possible step in a very short time according to the algorithms used behind it.

7) **Daily Applications :**

- Daily applications such as Apple's **Siri**, Window's **Cortana**, Google's **OK Google** are frequently used in our daily routine whether it is for searching a location, taking a selfie, making a phone call, replying to a mail and many more.
- **Example :** Around 20 years ago, when we are planning to go somewhere we used to ask a person who already went there for the directions. But now all we have to do is say **"OK Google** where is Visakhapatnam". It will show you Visakhapatnam's location on google map and the best path between you and Visakhapatnam.

8) **New Inventions :**

- AI is powering many inventions in almost every domain which will help humans solve the majority of complex problems.
- **Example :** Recently doctors can predict breast cancer in the woman at earlier stages using advanced AI-based technologies.

1.6.2 Risks of Artificial Intelligence

1) **High Costs of Creation :**

As AI is updating every day the hardware and software need to get updated with time to meet the latest requirements. Machines need repairing and maintenance which need plenty of costs. It' s creation requires huge costs as they are very complex machines.

2) **Making Humans Lazy :**

AI is making humans lazy with its applications automating the majority of the work. Humans tend to get addicted to these inventions which can cause a problem to future generations.

3) **Unemployment :**

As AI is replacing the majority of the repetitive tasks and other works with robots,human interference is becoming less which will cause a major problem in the employment standards. Every organization is looking to replace the minimum qualified individuals with AI robots which can do similar work with more efficiency.

4) **No Emotions :**

There is no doubt that machines are much better when it comes to working efficiently but they cannot replace the human connection that makes the team. Machines cannot develop a bond with humans which is an essential attribute when comes to Team Management.

5) **Lacking out of Box Thinking :**

Machines can perform only those tasks which they are designed or programmed to do, anything out of that they tend to crash or give irrelevant outputs which could be a major backdrop.

1.7 State of the Art

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.

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.

1.8 Intelligent Agents

1.8.1 What is an Agent ?

Q. Define in your own words, the following terms :
1. Agent
2. Agent function
3. Agent program
4. Autonomy

- Agent is something that perceives its environment through sensors and acts upon that environment through effectors or actuators. Fig. 1.8.1 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.

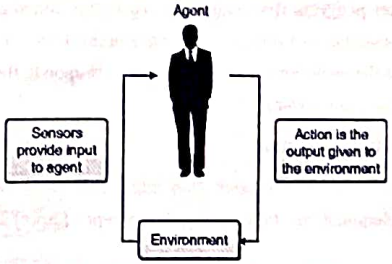


Fig. 1.8.1 : Agent and Environment

- Fig. 1.8.2 shows generic robotic agent structure.

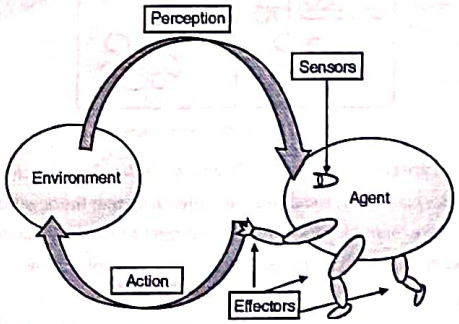


Fig. 1.8.2 : Generic Robotic Agent Architecture

- 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.

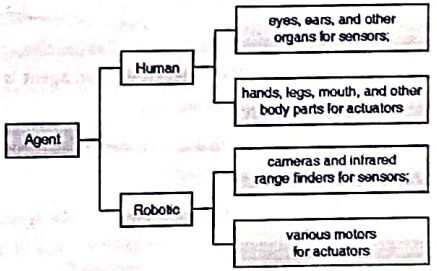


Fig. 1.8.3 : Sensors and Actuators in Human and Robotic Agent

- 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.

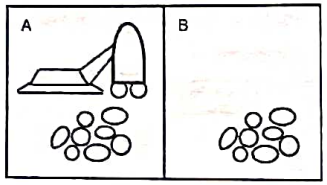


Fig. 1.8.4 : Vacuum cleaner Agent

- As shown in Fig. 1.8.4, 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.

1.8.2 Definition 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".
- 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".

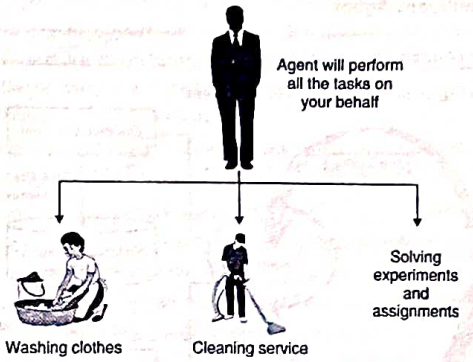


Fig. 1.8.5 : Interactive Intelligent Agent

- From above definitions we can understand that an agent is : (As per Terziyan, 1993)
  - Goal-oriented
  - Creative
  - Adaptive
  - Mobile
  - Social
  - Self-configurable

1.8.3 Definition of Intelligent Agent

University Question

Q. What is intelligent agent ? **SPPU : May 18, 3 Marks**

- 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.

1.8.3(A) Structure of Intelligent Agent

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

- Fig. 1.8.6 shows the general structure of an intelligent agent.

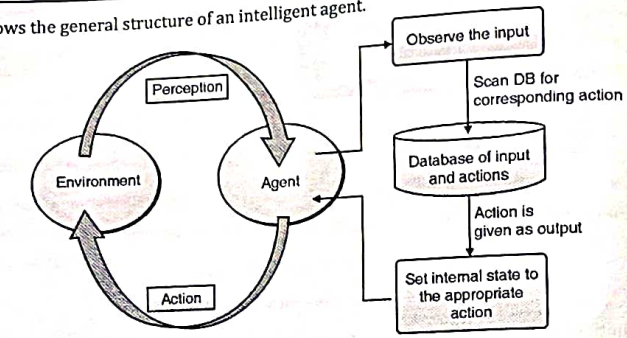


Fig. 1.8.6 : General Structure of Intelligent Agent

- From Fig. 1.8.6 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.
  - **Self-Learning :** An intelligent agent changes its behaviour based on its previous experience. This agent keeps updating its knowledge base all the time.
  - **Movable/Mobile :** An Intelligent agent can move from one machine to another while performing actions.
  - **Self-governing :** An Intelligent agent has control over its own actions.

1.9 Rational Agent

University Question

Q. Define rationality and rational agent. Give an example of rational action performed by any intelligent agent.

SPPU : Dec. 15, 5 Marks

- 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. 1.9.1

- **For example**, see Fig. 1.9.1. 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.

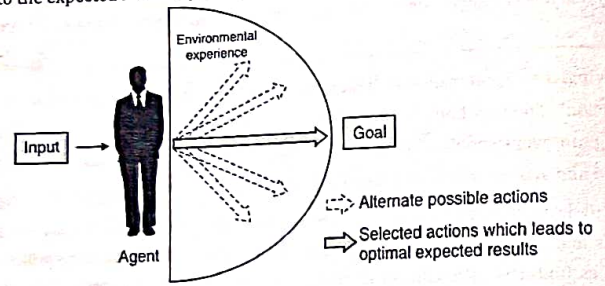


Fig. 1.9.2 : Rational Agent

1.10 Environments Types and PEAS Properties of Agent

1.10.1 Environments Types

University Question

Q. Write short note on : properties of Agent task environments ? **SPPU : Dec. 15, May 19, 5 Marks**

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.

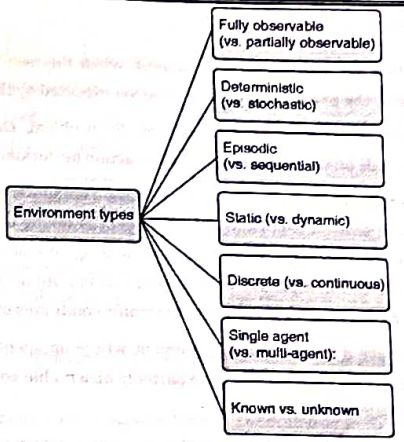


Fig. 1.10.1 : Environment types

- 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 types 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 1.10.1 summarizes few task environment and their characteristics.

Table 1.10.1 : Task Environments

Task environment	Car driving	Part – Picking Robot	Cross word puzzle	Soccer game	Checkers with clock
Observable	Partially	Partially	fully	partially	Fully
Agents	Multi agent (cooperative)	Single agent	single	Multi agent (competitive)	Multi agent (competitive)

Task environment	Car driving	Part – Picking Robot	Cross word puzzle	Soccer game	Checkers with clock
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

1.10.2 PEAS Properties of Agent

Q. What are PEAS descriptor ? Give PEAS descriptors for Part - picking Robot.

Q. Give PEAS description for a robot soccer player. Characterize its environment.

- **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.

1.11 Types of Agents

Q. Give the classification of agents.

Q. Explain various types of intelligent agents, state limitations of each and how it is overcome in other type of agent.

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. 1.11.1.

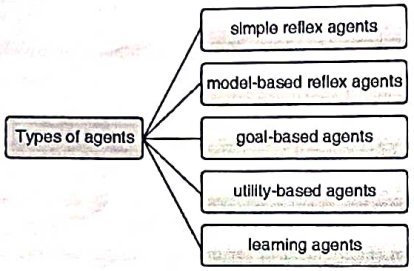


Fig. 1.11.1 : Types of agents

Let us understand these agent types one by one.

1.11.1 Simple Reflex Agents

Q. Explain simple reflex agent architecture.

- An agent which performs actions based on the current input only, by ignoring all the previous inputs is called as **simple reflex agent.**

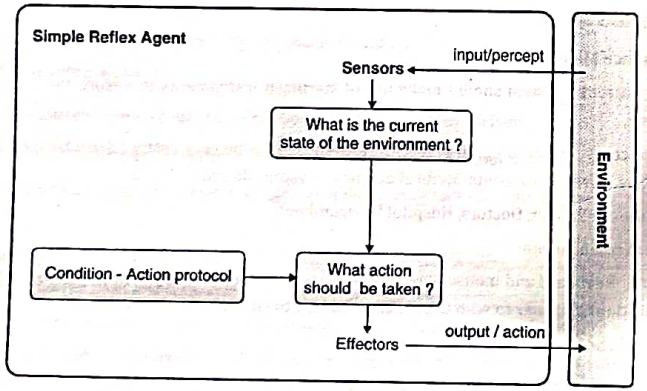


Fig. 1.11.2 : Simple reflex agents

- 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 1.11.1.
- Few possible input sequences and outputs for vacuum cleaner world with 2 locations are considered for simplicity.

Table 1.11.1

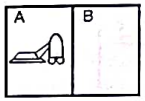
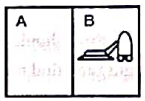
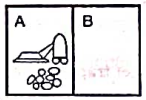
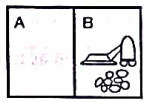
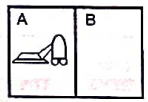
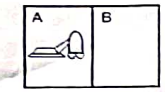
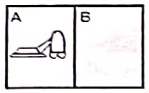
Figure	Input sequence {location, content}	Output / action Right, left, suck, no-op
	{A, clean}	Right
	{B, clean}	Left
	{A, dirt}	Suck
	{B, dirt}	Suck
	Input sequence {location, content }	Output / action Right, left, suck, no-op

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

- 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.

1.11.2 Model-based Reflex Agent

University Question

Q. Explain Model-based Reflex Agent with block diagram. **SPPU : May 15, May 16, Dec. 16, Dec. 19, 5 Marks**

- 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. 1.11.3 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.

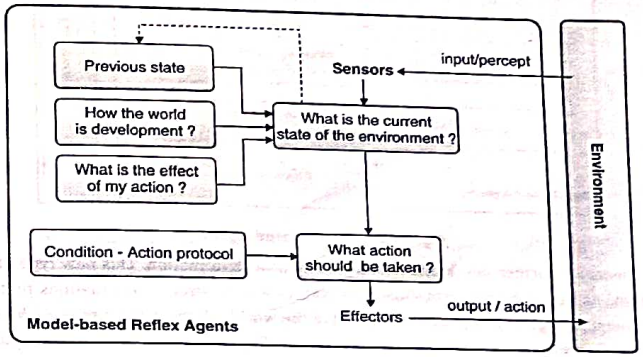


Fig. 1.11.3 : Model-based reflex agent

- 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.

1.11.3 Goal-based Agent

University Question

Q. Explain Model-based Reflex Agent with block diagram. **SPPU : May 17, Dec. 17, Dec. 18, 5 Marks**

Q. Explain Goal based agent.

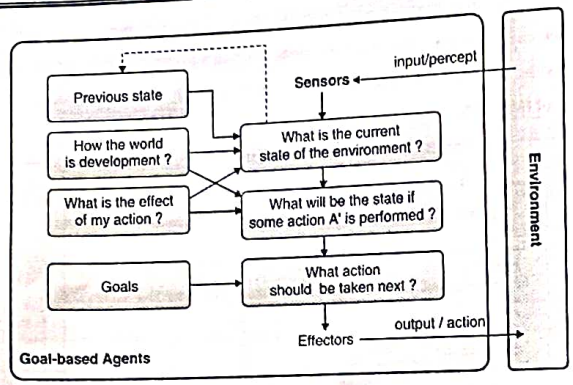


Fig. 1.11.4 : Goal-based agent

- 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.

1.11.4 Utility-based Agent

University Question

Q. Explain utility based agent with the help of neat diagram.

SPPU : May 15, May 16, Dec. 16, May 17, Dec. 17, Dec. 18, Dec. 19, 5 Marks

- Utility function is used to map a state to a measure of utility of that state. We can define a measure for determining how advantageous a particular state is for an agent. To obtain this measure utility function can be used.
- The term utility is used to depict how "happy" the agent is to find out a generalized performance measure, various world states according to exactly how happy they would make an agent is compared.
- Take one example; you might have used Google maps to find out a route which can take you from source location to your destination location in least possible time. Same logic is followed by utility based automatic car driving agent.

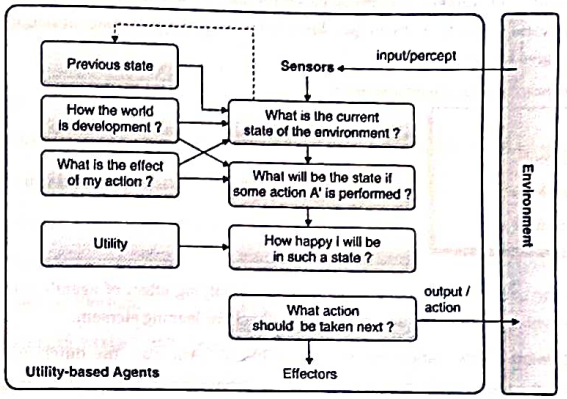


Fig. 1.11.5 : Utility-based agent

- Goals utility based automatic car driving agent can be used to reach given location safely within least possible time and save fuel. So this car driving agent will check the possible routes and the traffic conditions on these routes and will select the route which can take the car at destination in least possible time safely and without consuming much fuel.

1.11.5 Learning Agents

Q. Explain the structure of learning agent architecture. What is role of critic in learning ?

Q. What are the basic building blocks of learning agent ? Explain each of them with a neat block diagram.

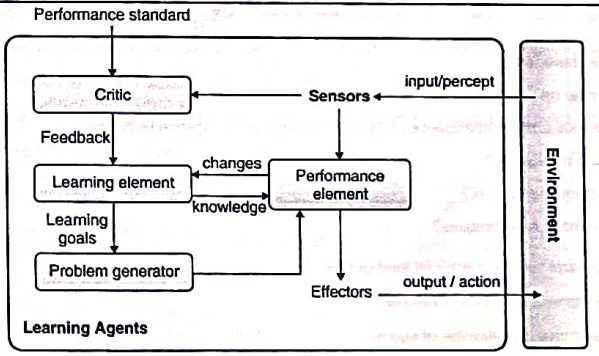


Fig. 1.11.6 : Learning agents

- 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.

Review Questions

Q. 1 Define artificial intelligence.

Q. 2 Write a short note on : Applications of artificial intelligence.

Q. 3 Explain the various artificial intelligence problems and artificial intelligence techniques.

Q. 4 What is artificial intelligence?

Q. 5 What are the components of AI?

Q. 6 What are the various AI techniques?

Q. 7 Explain various applications of Artificial Intelligence.

Q. 8 Explain PEAS representation with example.

Q. 9 Define agent and give classification of agents.

Q. 10 What is intelligent agent?

Q. 11 Write a short note on: Rational agent.

Q. 12 Write a short note on : Structure of Intelligent agents.

Q. 13 Give types of agents.

Q. 14 What are various agent environments? Give PEAS representation for an agent.

Q. 15 Define in your own words, the following terms :

1. Agent
2. Agent function
3. Agent program
4. Autonomy

Q. 16 Explain various types of intelligent agents, state limitations of each and how it is overcome in other type of agent.

Q. 17 What do you mean by PEAS? Explain properties of task environment.

Q. 18 Explain detail architecture of goal based agent.

Q. 19 Explain Simple reflex agent architecture.

Q. 20 Explain learning agent architecture.

Q. 21 What are PEAS descriptor ? Give PEAS descriptors for Part - picking Robot.

Q.2 2 Explain utility based agents with the help of neat diagram.

Q. 23 Explain the learning agent with the help of suitable diagram.

Q. 24 Describe different types of environments applicable to AI agents.

Q. 25 Explain the structure of learning agent architecture. What is role of critic in learning.

Q. 26 Discuss structure of learning agent.

Q. 27 What are PEAS descriptor ? Give PEAS descriptors for Part – picking Robot.

Q. 28 Describe different types of environments applicable to AI agents.

Q. 29 Explain the structure of learning agent architecture. What is role of critic in learning.

Q. 30 Define rationality and rational agent. Give an example of rational action performed by any intelligent agent.

Q. 31 What are the basic building blocks of learning agent ? Explain each of them with a neat block diagram.

Q. 32 Give PEAS description for a robot soccer player. Characterize its environment.

Q. 33 What are the basic building blocks of learning agent ? Explain each of them with a neat block diagram.

Q. 34 Explain Turing test designed for satisfactory operational definition of AI.

□□□

2 PROBLEM-SOLVING

Syllabus

Solving Problems by Searching, Problem-Solving Agents, Example Problems, Search Algorithms, Uninformed Search Strategies, Informed (Heuristic) Search Strategies, Heuristic Functions, Search in Complex Environments, Local Search and Optimization Problems.

Introduction

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.

2.1 Solving Problems by Searching

- Now let us see how searching play a vital role in solving AI problems. Given a problem, 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.

2.2 Formulating Problems

Q. Explain how you will formulate search problem.

- 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.

2.2.1 Components of Problems Formulation

Q. What are the components of problem formulation?

Q. Explain steps in problem formulation with example.

- 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.

2.2.2 Example of 8-Puzzle Problem

Q. Formulate 8-puzzle problem.

- Fig. 2.2.1 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.

1	2	3
4	8	–
7	6	5

Initial State

1	2	3
4	5	6
7	8	–

Goal State

Fig. 2.2.1 : A scenario of 8-Puzzle Problem

- This problem can be formulated as follows :
- States : States can represented by a 3 × 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. 2.2.1, 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

2.2.3 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) → (2,2,0) →(3,2,1) → (3,0,0) → (3,1,1) → (1,1,0) → (2,2,1) → (0,2,0) → (0,3,1) → (0,1,0) → (0,2,1) → (0,0,0)

Cost = 11 crossings

2.2.4 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. 2.2.2 indicating all possible states with actions and the next state.

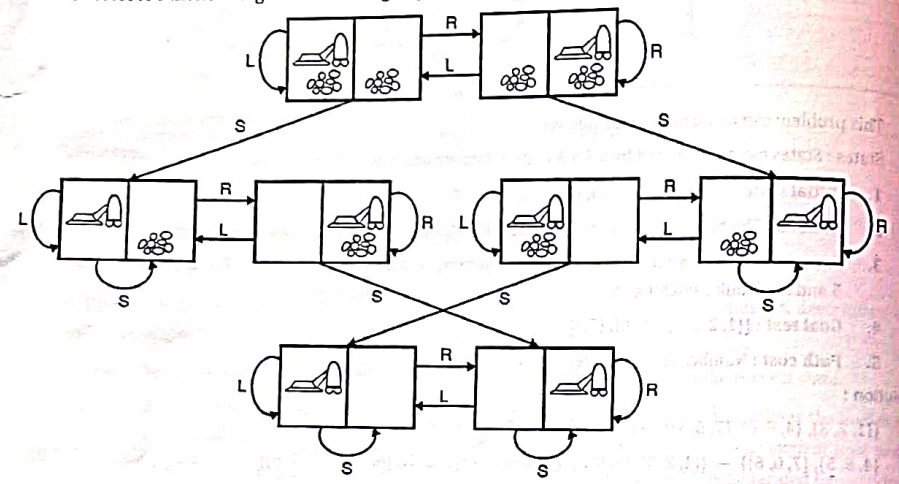


Fig. 2.2.2 : The state space for vacuum world

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.

2.2.5 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, Travelling 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.

2.3 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.

2.4 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. 2.4.1 shows the representation of a tree node as a data structure in 8-puzzle problem.

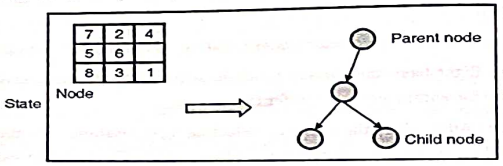


Fig. 2.4.1 : Node representation of state in searching

- 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.

2.5 Uninformed Search Strategies

University Question

Q. Explain with example uniformed search strategies. **SPPU : May 17, 10 Marks**

- Why is it called uninformed search? What is not been informed about the 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".

2.5.1 Depth First Search (DFS)

Q. Explain Depth First Search Technique with an example.

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. 2.5.1.

- 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.

Process

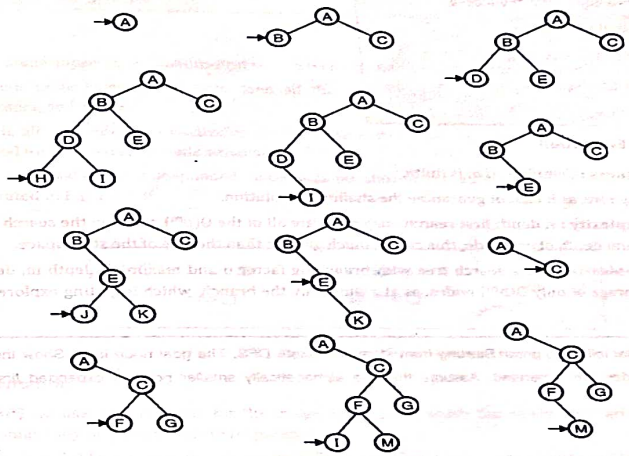


Fig. 2.5.1 : Working of Depth first search on a binary tree

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.

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;
```

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 $O(b^m)$ 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 $O(b^m)$ nodes, as at a time only the branch, which is getting explored, will reside in memory.

Ex. 2.5.1 : 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. :

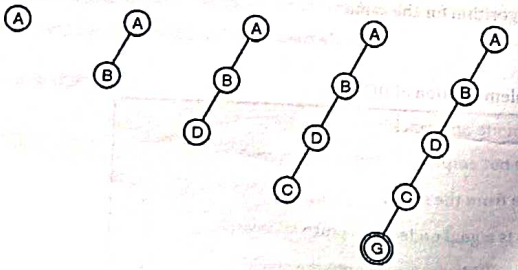


Fig. P. 2.5.1

2.5.2 Breadth First Search (BFS)

Q. Write a note on BFS.
Q. Explain breadth first algorithm.

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. 2.5.2.

2. Process

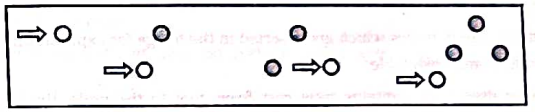


Fig. 2.5.2 : Working of BFS on binary tree

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.

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;
```

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 :** $O(b^d)$, number of nodes in the fringe.
- **Space complexity :** $O(b^d)$, total number of nodes explored.

2.5.3 Uniform Cost Search (UCS)

Q. Write a note on : Uniform cost search.

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.

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.

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".

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 ϵ. Then the algorithm's worst-case time and space complexity is $O(b^{C^*/\epsilon})$, which can be much greater than bd.
- **Space complexity :** $O(b^{C^*/\epsilon})$, indicating number of node in memory at execution time.

2.5.4 Depth Limited Search (DLS)

Q. How the drawbacks of DFS are overcome by DLS and IDDFS?

Q. Compare and contrast DFS, DLS and IDDFS.

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. 2.5.3.

2. Process

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

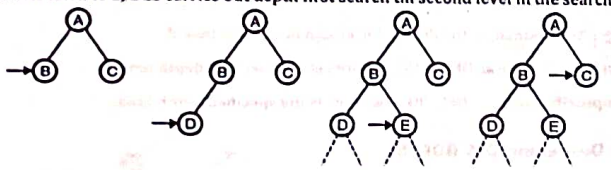


Fig. 2.5.3 : DIS working with depth limit

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.

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.

5. Pseudo Code

```
boolean DLS (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;
}
```

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 (b^l), where l is the specified depth limit.
- **Space complexity :** Same as DFS, O(b^l), where l is the specified depth limit.

2.5.5 Iterative Deepening DFS (IDDFS)

Q. Explain Iterative deepening search algorithms based on performance measure with justification : complete optimal, time complexity & space complexity. **SPPU : Dec. 18, 10 Marks**

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. 2.5.4.

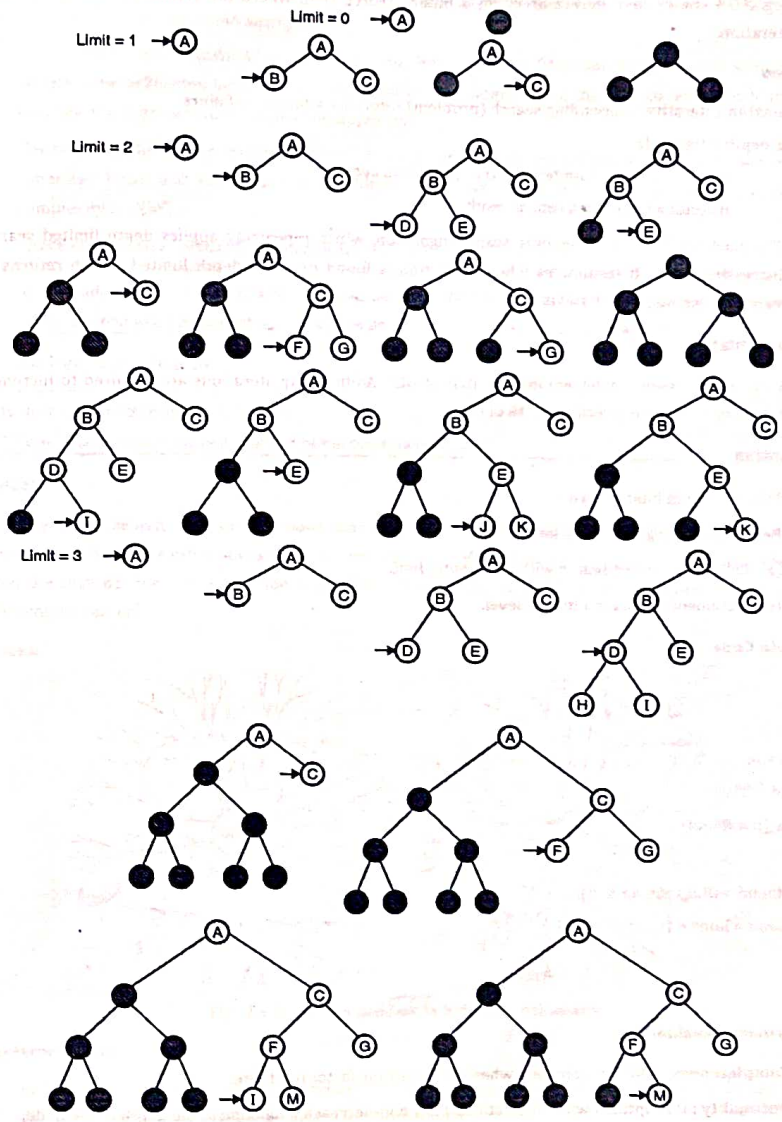


Fig. 2.5.4 : Search process in IDDFS

- Fig. 2.5.4 shows four iterations of on a binary search tree, where the solution is found on the fourth iteration.

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. 2.5.4 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.

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.

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.

5. Pseudo Code

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

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 $O(b^d)$.
- **Space complexity :** Memory requirements of IDDFS are modest, i.e. $O(b^d)$.

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.

2.5.6 Bidirectional Search

Q. Write short note on bidirectional search.
Q. Differentiate between unidirectional and bidirectional search.

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. 2.5.5 shows the general search process in bidirectional search.

2. Process

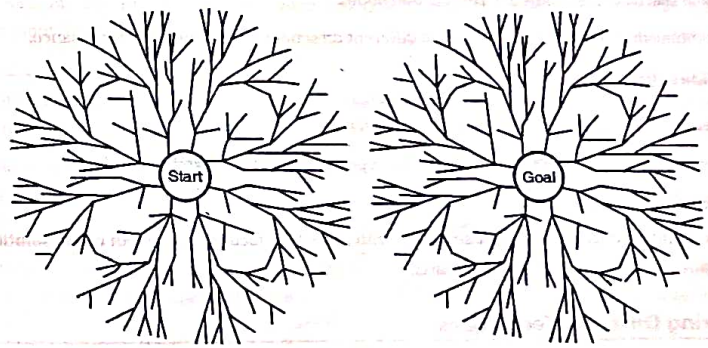


Fig. 2.5.5 : Search process in bidirectional search

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.

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

5. Pros of Bidirectional Search

- It is much more efficient.
- Reduces space and time requirements as, we perform two $b^{d/2}$ searches, instead of one b^d search.

Example :

Suppose b = 10, d = 6. Breadth first search will examine $10^6 = 1,000,000$ nodes.

Bidirectional search will examine $2 \times 10^3 = 2,000$ nodes.

One can combine different search strategies in different directions to avail better performance.

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 $b^{d/2}$ for each search.

2.6 Comparing Different Techniques

Q. Compare and contrast all the un-informed searching techniques.

Q. Write a note on comparative analysis of searching techniques.

- Table 2.6.1 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 2.6.1 : 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	$O(b^d)$	$O(b^{C^*/\epsilon})$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$
Space Complexity	$O(b^d)$	$O(b^{C^*/\epsilon})$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$

2.6.1 Difference between Unidirectional and Bidirectional Search

Q. Differentiate 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.
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.

Sr. No.	Unidirectional search method	Bidirectional search method
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

2.6.2 Difference between BFS and DFS

Q. Compare DFS and BFS with example.

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.
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.

Sr. No.	BFS	DFS
11.	Example :  A, B, C, D, E, F	Example :  A, B, D, C, E, F

2.7 Informed Search Techniques

Q. Write short note on Informed search.

- 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.

2.8 Heuristic Function

University Question

Q. What is heuristic function ? **SPPU : Dec. 16, May 17, May 18, Dec. 18, 5 Marks**

- 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. 2.8.1 shows the general representation of heuristic function.

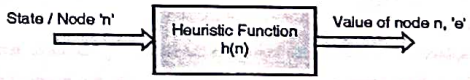


Fig. 2.8.1 : General representation of Heuristic function

- 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.

2.8.1 Example of 8-puzzle Problem

University Question

Q. Give an example Heuristic Function for 8-puzzle Problem. **SPPU : May 17, 2 Marks**

- 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. 2.8.2 : A scenario of 8-puzzle problem

- Two simple heuristic functions are :
  - h_1 = the number of misplaced tiles. This is also known as the Hamming Distance. In the Fig. 2.8.2 example, the start state has $h_1 = 8$. Clearly, h_1 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?
  - h_2 = 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. 2.8.2, the start state has $h_2 = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18$. Clearly, h_2 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 ($h_1 + h_2$). Additionally, it is easy to see from the definitions of the heuristic functions that for any given state, h_2 will always be greater than or equal to h_1. Thus, we can say that h_2 dominates h_1.

2.8.2 Example of Block World Problem

University Question

Q. Find the heuristics value for a particular state of the blocks world problem. **SPPU : Dec. 15, May 17, 5 Marks**

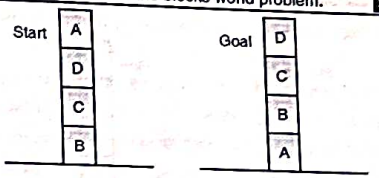


Fig. 2.8.3 : Block Problem

- Fig. 2.8.3 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. 2.8.4

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.

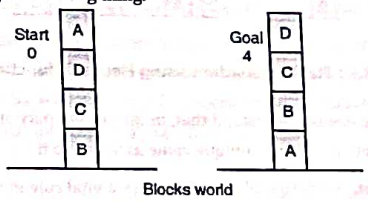


Fig. 2.8.4 : Definition of Heuristic Function "h_1"

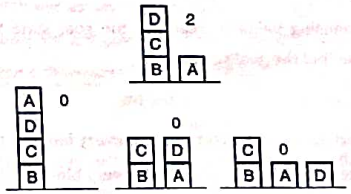


Fig. 2.8.5 : State evaluations using Heuristic function "h_1"

- Fig. 2.8.5 shows the heuristic values generated by heuristic function "h_1" 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. 2.8.6 is depicting a new heuristic function "h_2" 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.

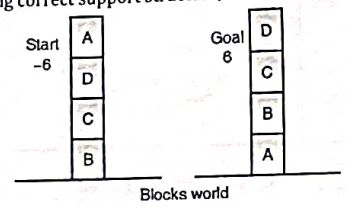


Fig. 2.8.6 : Definition of heuristic function "h_2"

- As we observe in Fig. 2.8.7, the same states are considered again as that of Fig. 2.8.5, but this time using h_2, each one of the state is assigned a unique value generate according to heuristic function h_2.

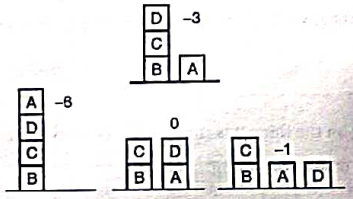


Fig. 2.8.7 : State evaluations using Heuristic function "h_2"

- 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

- 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??

2.8.3 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.

2.9 Best First Search

Q. Write algorithm for Best first search and specify its properties.
Q. What is the difference between best first and greedy best first search? Explain with example.
Q. Explain search strategy to overcome drawbacks of BFS and DFS.

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. 2.9.1 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.

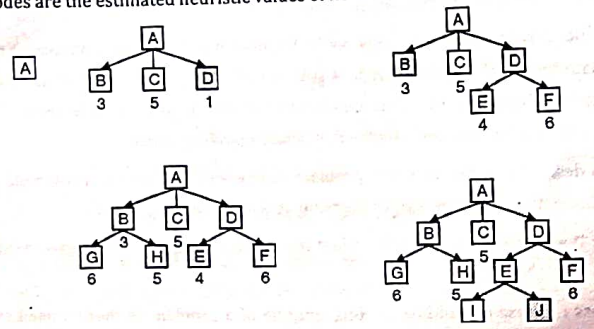


Fig. 2.9.1 : Best first search tree expansion scenario

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.

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.

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).

2.9.1 Greedy Best First Search

Q. What is the difference between best first and greedy best first search ? Explain with example.

- 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. 2.9.2. 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.

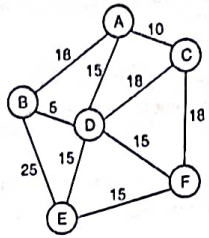


Fig. 2.9.2 : Travelling Salesmen Problem example

- 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.

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. **Time Complexity :** $O(b^m)$, but a good heuristic can give dramatic improvement.
4. **Space Complexity :** $O(b^m)$, It needs to keep all nodes in memory.

2.10 A* Search

Q.	Compare Best First Search and A* Search with an example.
Q.	Explain A* Algorithm. What is the drawback of A* ? Also shows that A* is optimally efficient.
Q.	Describe A* algorithm with merits and demerits.
Q.	Explain A* algorithm with example.

2.10.1 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.

2.10.2 Implementation

A* does also use both OPEN and CLOSED list.

2.10.3 Algorithm (A*)

1. Initialization OPEN list with initial node; CLOSED= ϕ; g = 0, f = h, **Found = false;**
2. While (OPEN ≠ ϕ 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

2.10.4 Behaviour of A* Algorithm

Q. Write short note on behavior of A* in case of underestimating and overestimating Heuristic.

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

- 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.

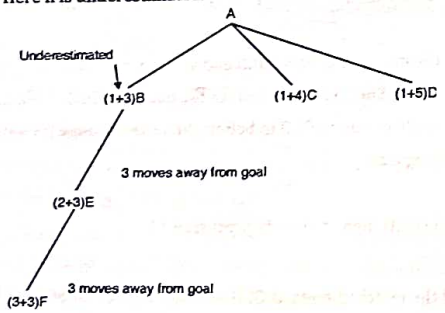


Fig. 2.10.1

- 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

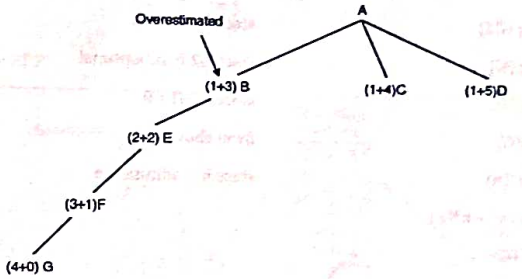


Fig. 2.10.2

- 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.

2.10.5 Admissibility of A*

Q. Discuss admissibility of A* in case of optimality.

Q. What do you mean by admissible heuristic function? Explain with example.

Q. Write short note on 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.

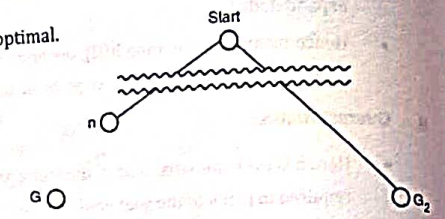


Fig. 2.10.3 : Optimality of A*

$f(G2) = g(G2)$ 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

$h(n) \leq h^*(n)$ since h is admissible

$g(n) + h(n) \leq g(n) + h^*(n)$

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

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

2.10.6 Monotonicity

Q. Prove that A* is admissible if it uses Monotone heuristic.

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

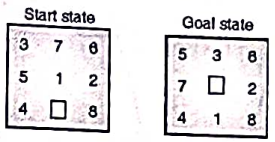
$$h(X_i) - h(X_j) \leq \text{cost}(X_i, X_j) \quad \text{where,}$$

- cost (X_i, X_j)actual cost of going from X_i to X_j 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.

2.10.7 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 :** $O(b^m)$, as the number of nodes grows exponentially with solution cost.
4. **Space Complexity :** $O(b^m)$,as it keeps all nodes in memory.

2.10.8 Example : 8 Puzzle Problem using A* Algorithm



Evaluation function - f for EPP

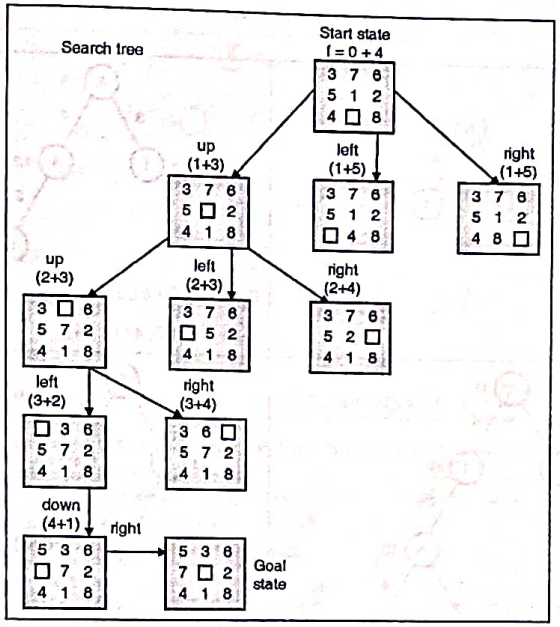


Fig. 2.10.4 : Solution of 8-puzzle using A*

- 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. 2.10.1 : Consider the graph given in Fig. P. 2.10.1. Assume that the initial state is 8 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(8) = 15.

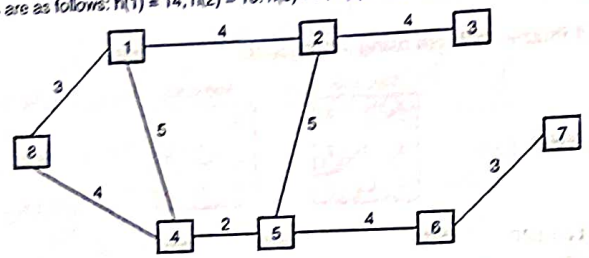
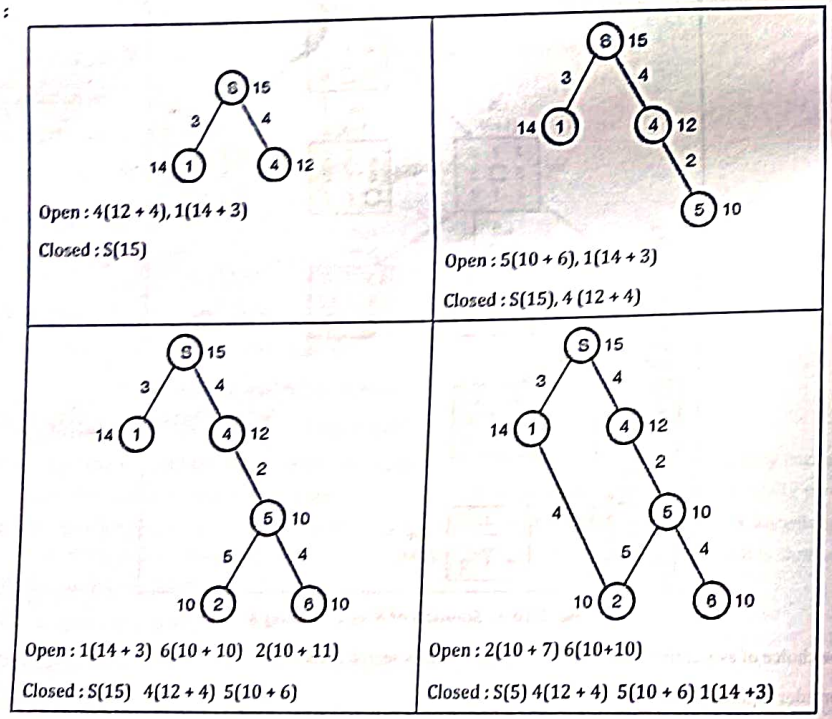
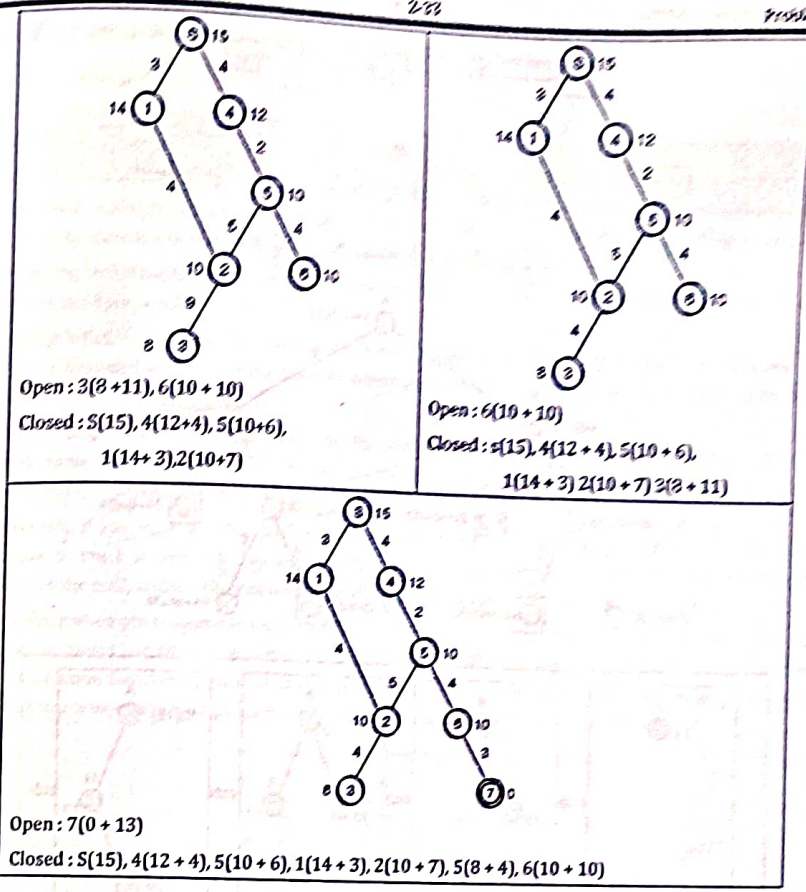


Fig. P. 2.10.1

Soln. :

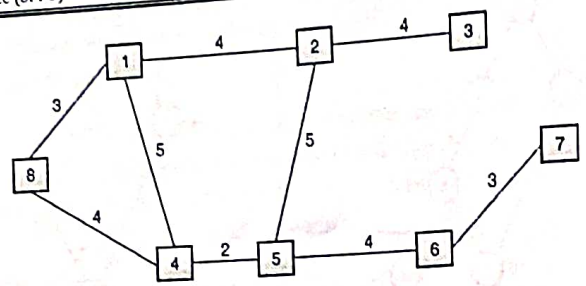


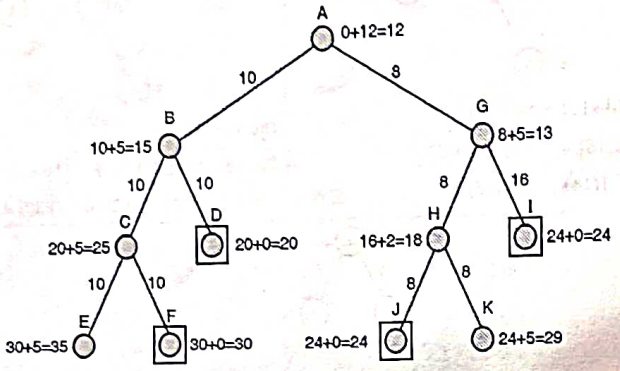


2.10.9 Comparison among Best First Search, A* Search and Greedy Best First Search

Q. Compare following informed searching algorithms based on performance measure with justification : Complete, Optimal, Time complexity and space complexity.
(a) Greedy best first, (b) A* (c) recursive best-first(RBFS)

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)





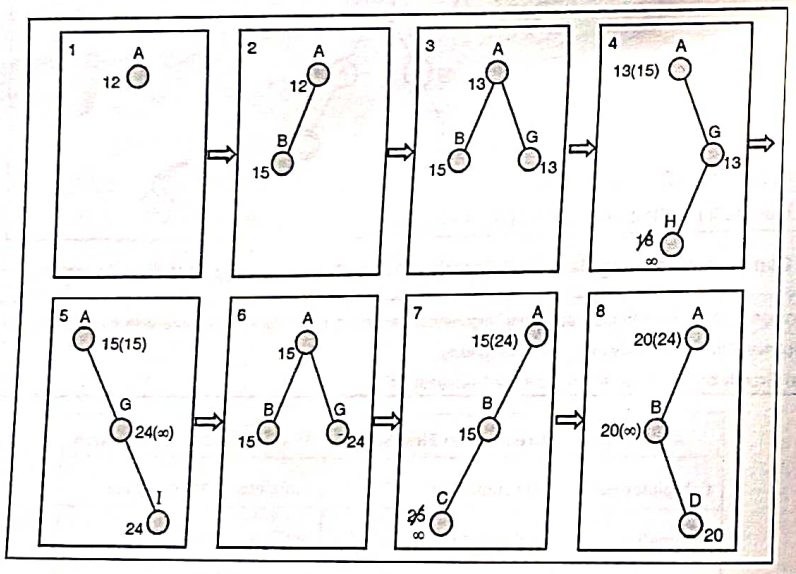


Fig. 2.10.5 : Process of SMA* with memory size of 3 nodes

2.11 Local Search Algorithms and Optimization Problems

2.11.1 Hill Climbing

Q. Describe Hill Climbing algorithm. What are it's limitations ?
Q. Explain Hill- Climbing algorithm with example.

- 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. 2.11.1 depicts the typical hill climbing scenario, where multiple paths are available to reach to the hill top from ground level.

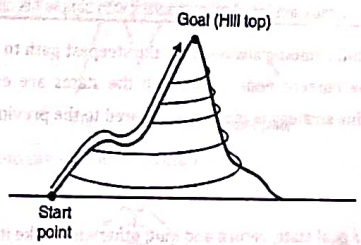


Fig. 2.11.1 : Hill Climbing Scenario

- 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.

2.11.1(A) 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.

2.11.1(B) Steepest Ascent Hill Climbing

Q. Write algorithm of steepest ascent hill climbing. And compare it with simple 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.

2.11.1(C) Limitations of Hill Climbing

Q. Explain limitations of Hill Climbing in brief.
Q. Write short note on Limitations of Hill-climbing algorithm.

- Now let's see what can be the impact of incorrect design of heuristic function on the hill climbing techniques.
- 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. 2.11.2 : Local Maximum

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

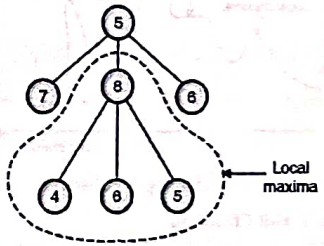


Fig. 2.11.3 : Local maxima in Search Tree

- **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.

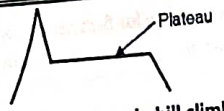


Fig. 2.11.4 : Plateau in hill climbing

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

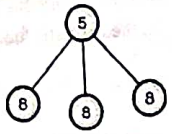


Fig. 2.11.5

- **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. 2.11.6 : Ridge in Hill Climbing

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

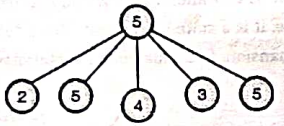


Fig. 2.11.7

- Fig. 2.11.8 depicts all the different situations together in hill climbing.

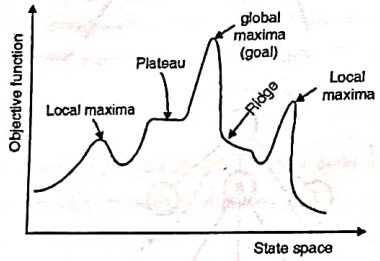


Fig. 2.11.8 : Hill climbing problems scenario

2.11.1(D) Solutions on Problems in Hill Climbing

Q. Explain solution in brief for Problems in Hill Climbing.

- In order to overcome these problems we can try following techniques. At times combination of two 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 a 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.

2.11.2 Simulated Annealing

Q. Write a note on : 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)
     - $\Delta E < 0$ → 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.

2.11.2(A) Comparing Simulated Annealing with Hill Climbing

Q. Compare and contrast 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.

2.11.3 Local Beam Search

Q. Write short note on local beam search.

Q. What is the significance of local minima in neuron learning ?

- 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
```

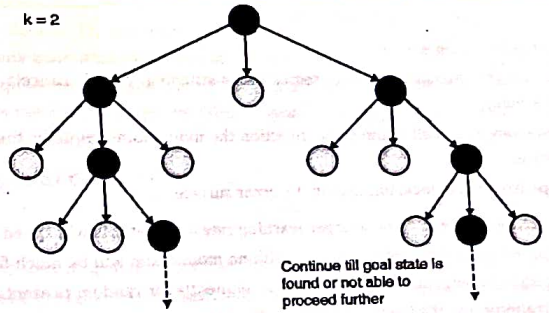


Fig. 2.11.9 : Process of local Beam Search

- As shown in Fig. 2.11.9, 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.

2.11.4 Genetic Algorithms

Q. Write a short note on genetic algorithm.

Q. Explain how genetic algorithm can be used to solve a problem by tacking a suitable example ?

- GAs are adaptive heuristic search algorithms based on the evolutionary ideas of natural selection and genetics. As such they represent an intelligent exploitation of a random search used to solve optimization problems. Although randomized, GAs are by no means random, instead they exploit historical information to direct the search for better performance within the search space. The basic techniques of the GAs are designed to simulate processes in natural systems necessary for evolution, especially those following the principles of "survival of the fittest" laid down by Charles Darwin.
- Genetic algorithms are implemented as a computer simulation in which a population of abstract representations (called chromosomes or the genotype or the genome) of candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem evolves towards better solutions. The solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible. The evolution usually starts from a population of randomly generated individual and occurs in generations. In each generation, the fitness of every individual in the population is evaluated, multiple individuals are stochastically selected from the current population (based on their fitness), and modified to form a new population. The new population is then used in the next iteration of the algorithm.

2.11.4(A) Terminologies of GA

Gene

Gene is the smallest unit in genetic algorithm. The gene represents the smallest unit of information in the problem domain and can be thought of as the basic building block for a possible solution. If the problem context were, for example, the creation of a well-balanced investment portfolio, a gene might represent the number of shares of a particular security to purchase.

Chromosome

Chromosome is a series of genes that represent the components of one possible solution to the problem. The chromosome is represented in computer memory as a bit string of binary digits that can be "decoded" by the genetic algorithm to determine how good a particular chromosome's gene pool solution is for a given problem. The decoding process simply informs the genetic algorithm what the various genes within the chromosome represent.

Encoding

Encoding of chromosomes is one of the problems, to start solving problem with GA. Encoding depends on the type of the problem. There are various types of encoding techniques like binary encoding, permutation encoding, value encoding, etc.

Population

A population is a pool of individuals (chromosomes) that will be sampled for selection and evaluation. The performance of each individual will be computed and a new population will be reproduced using standard genetic operators.

Reproduction

Reproduction is the process of creating new individuals called off-springs from the parents population. This new population will be evaluated again to select the desired results. Reproduction is done basically using two genetic operators: crossover and mutation. However, the genetic operators used can vary from model to model, there are a few standard or canonical operators : crossover and recombination of genetic material contained in different parent chromosomes, random mutation of data in individual chromosomes and domain specific operations, such as migration of genes.

2.11.4(B) Genetic Operators

Selection

In this process, chromosomes are selected from the population to be the parents for the crossover. The problem is how to select these chromosomes. According to Darwin's evolution theory the best ones should survive and create new offspring. There are many methods how to select the best chromosomes, for example roulette wheel selection, Boltzman selection, tournament selection, rank selection, steady state selection, etc.

Crossover

Crossover involves the exchange of gene information between two selected chromosomes. The purpose of the crossover operation is to allow the genetic algorithm to create new chromosomes that shares positive characteristics while simultaneously reducing the prevalence of negative characteristics in an otherwise reasonably fit solution. Types of crossover techniques include single-point crossover, two-point crossover, uniform crossover, mathematical crossover, tree crossover, etc.

Mutation

Mutation is another refinement step that randomly changes the value of a gene from its current setting to a completely different one. The majority of the mutations formed by this process are, as is often the case in nature, less fit than more so. Occasionally, however, a highly superior and beneficial mutation will occur. Mutation provides the genetic algorithm with the opportunity to create chromosomes and information genes that can explore previously uncharted areas of the solution space, thus increasing the chances for the discovery of an optimal solution. There are various types of mutation techniques like bit inversion, order changing, value encoding.

2.11.4(C) The Basic Genetic Algorithm

1. [Start] Generate random population of n chromosomes (suitable solutions for the problem)
2. [Fitness] Evaluate the fitness f(x) of each chromosome x in the population
3. [New population] Create a new population by repeating following steps until the new population is complete
4. [Selection] Select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to be selected)
5. [Crossover] With a crossover probability cross over the parents to form a new offspring (children). If no crossover was performed, offspring is an exact copy of parents.
6. [Mutation] With a mutation probability mutate new offspring at each locus (position in chromosome).
7. [Accepting] Place new offspring in a new population
8. [Replace] Use new generated population for a further run of algorithm
9. [Test] If the end condition is satisfied, stop, and return the best solution in current population
10. [Loop] Go to step 2

2.11.4(D) Example of Genetic Algorithm

Let's consider following pair of chromosomes encoded using permutation encoding technique and are undergoing the complete process of GA. Assuming that they are selected using rank selection method and will be applied, arithmetic crossover and value encoding mutation techniques.

Parent chromosomes

Chromosome A	1 5 3 2 6 4 7 9 8
Chromosome B	8 5 6 7 2 3 1 4 9

Child chromosome after arithmetic crossover: i.e adding bits of both chromosomes.

Child chromosome :

Chromosome C	9 0 9 9 8 7 8 3 7

After applying value encoding mutation i.e. adding or subtracting a small value to selected values. e.g. subtracting 1 to 3rd and 4th bit.

Child chromosome

Chromosome C	9 0 8 8 8 7 8 3 7

It can be observed that the child produced is much better than both parents.

Review Questions

Q. 1 Why is it called uninformed search? What is not been informed about the search?

Q. 2 Write a note on BFS.

Q. 3 Write a note on : Uniform cost search.

Q. 4 How the drawbacks of DFS are overcome by DLS and IDDFS?

Q. 5 Compare and contrast DFS, DLS and IDDFS.

Q. 6 Write short note on bidirectional search.

Q. 7 Differentiate between unidirectional and bidirectional search.

Q. 8 Compare and contrast all the un-informed searching techniques.

Q. 9 Write a note on comparative analysis of searching techniques.

Q. 10 Write a short note on Interative deepening search.

Q. 11 Compare DFS and BFS with example.

Q. 12 What is heuristic function? What are the qualities of a good heuristic ?

Q. 13 Write a short note on : Properties of Heuristic function and its role in AI.

Q. 14 Write algorithm for Best first search and specify its properties.

Q. 15 What is the difference between best first and greedy best first search? Explain with example.

Q. 16 Discuss admissibility of A* in case of optimality.

Q. 17 Compare and contrast A*, SMA* and IDA*.

Q. 18 Explain solution in brief for Problems in Hill Climbing.

Q. 19 Write a note on : Simulated Annealing.

□□□