UNIT I

CHAPTER 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.

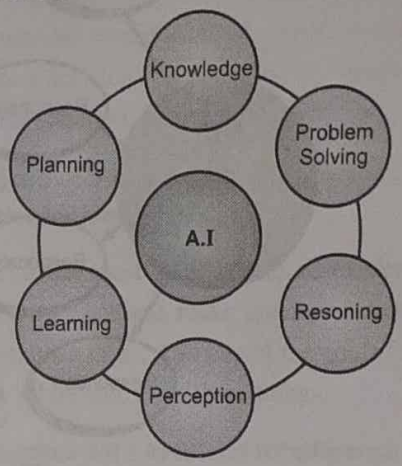
1.1 INTRODUCTION

- The field of Artificial Intelligence (AI) and Machine Learning (ML) has exploded in last decade. The roots of these words originate from multiple disciplines and not just computer science.
- These disciplines include pure mathematics, electrical engineering, statistics, signal processing, and communications along with computer science to name the top few.
- Along with the wide variety of origins, the field also finds applications in even greater number of industries ranging from high-tech applications like image processing, natural language processing to online shopping to non-destructive testing of nuclear power plants and so on.

1.1.1 What is AI ?

GQ. What is Artificial Intelligence?

- Alan Turing **defined Artificial Intelligence** as follows: "If there is a machine behind a curtain and a human is interacting with it (by whatever means, e.g. audio or via typing etc.) and if the human feels like he/she is interacting with another human, then the machine is artificially intelligent."
- There are two aspects to AI as viewed from human-like behavior standpoint. One is where the **machine is intelligent** and is **capable of communication with humans**, but does not have any locomotive aspects. The other aspect involves having physical interactions with human-like locomotion capabilities, which refers to the field of robotics.
- Artificial intelligence (AI, also machine intelligence, MI) is intelligence demonstrated by machines, in contrast to the natural intelligence (NI) displayed by humans and other animals. The term "artificial intelligence" is applied when a machine mimics "cognitive" functions that humans associate with other human minds, such as "learning" and "problem solving".
- We can use AI in many special domains and functions like understanding/learning, Planning, Problem solving, Decision making (Fig. 1.1.1). These all are very helpful now days in various industries.



(1A1)**Fig. 1.1.1 : Different phases under AI**

Artificial Intelligence is a term, which consists of two words.

1.1.1.1 Artificial

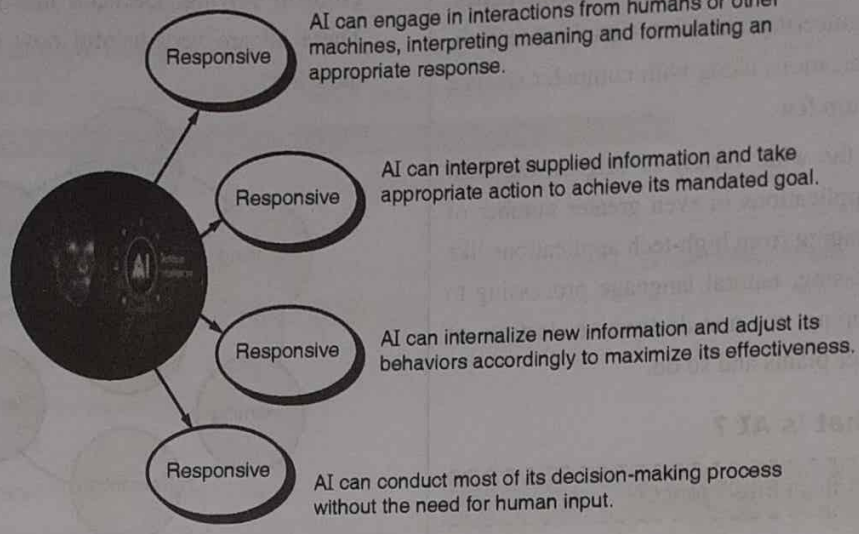
- Artificial is somewhat that is not real and is simulated. Artificial is not only used in the context of food. Artificial turf is a grass like surface used for sports playing fields.
- Artificial diamonds are those that are manufactured, as opposed to those that have developed through natural processes in the earth. Artificial flowers are often used as decorations.

1.1.1.2 Intelligence

- Defining and classifying intelligence is extremely complicated. Theories of intelligence range from having one general intelligence, to certain primary

mental abilities, and to multiple category-specific intelligences.

- Intelligence is defined in many different ways like logic, understanding, self-awareness, learning, emotional knowledge, planning, creativity and problem solving.
- Day by day lot of innovations takes place in today's world. New technologies, new machines with greater efficiency performances are the need of upcoming Techies and this thing is growing gradually exponentially.



(1A2)**Fig. 1.1.2 : Artificial Intelligence**

- Tremendous efforts in every domain has to be carried out as a need of future world that may be faster processing speed, reducing memory overhead lesser in size and more intelligent.
- Now the question arises, can we make machine intelligent? Answer is a Yes and this has been possible because of AI (Artificial Intelligence).
- As a result of this, we can make the system, and train the system build the system which can mimic the behavior and multiple properties and function of human being.

1.1.2 Objectives of AI

GQ. What are different objectives of AI ?

Below are the eight aims and objectives of artificial intelligence :

☞ Objective #1 : Artificial intelligence solves problems

- When it comes to artificial intelligence, there is a strong urge to create AI programs that look, act, and feel like real humans.
- However, many scientists now understand that the real goal is not to make a human-like robot. Instead, they would rather create a robot that works to make our lives easier, no matter what it looks or sounds like.
- Moving forward, it is likely that we will see some serious work being put into the ability for AI to learn and understand, and less on forcing them to act like real humans. That will probably just come with time.

☞ **Objective #2 : Artificial intelligence completes multiple tasks**

- Completing multiple tasks is another aims and objectives of artificial intelligence. One of the largest difficulties to overcome has been making it possible for an AI program or a "robot" to do more than one task. It is very easy to program a system to complete a certain task. For instance, it can bring an item from point A to point B.
- However, if you want the program to understand that it must pick up the item and then either bring it to point A or throw it in the trash based on arbitrary rules that a human would know that's a different story. In simpler terms, it might be a while before your housemaid is a robot.

☞ **Objective #3 : Artificial intelligence shapes the future of every company**

- AI is quickly becoming a crucial tool for all companies. They are using this technology to streamline their processes.
- It's no secret that the goal is to continue this trend for as many low-level tasks as possible. It ultimately saves the companies money in the long run, and it allows them to up productivity in other areas.

☞ **Objective #4 : Artificial intelligence prepares for a boom in big data**

- Big data has already taken the world by storm. Big data is the large-scale, and sometimes even random, collection of data about people's lives, habits, conversations and more.
- AI will be able to do much more for the analysis of this data than humans ever did, so data-driven research, advertisements, and content are going to explode.

☞ **Objective #5 : Artificial intelligence creates synergy between humans and AI**

One of the key goals in AI is to develop a strong synergy between AI and humans, so that they can work together to enhance the capabilities of both.

☞ **Objective #6 : Artificial intelligence is good at problem-solving**

So far, AI is unable to employ advanced problem-solving abilities. That is, it can tell you a factual answer, but cannot analyze a specific situation and make a decision based on the very specific context of that situation.

☞ **Objective #7 : Artificial intelligence helps with planning**

One of the most human traits in existence is the ability to plan and make goals and subsequently accomplish them. And one of the goals for AI is to have AI be able to do these things.

☞ **Objective #8 : Artificial intelligence performs more complex tasks**

- The key goal is this: to develop AI programs that can complete more and more complex tasks. Already the abilities are shocking, although not yet widespread.
- However, over time these will develop and ultimately, scientists hope, be able to do basically the same things humans can do.

1.2 INFORMATION, KNOWLEDGE AND INTELLIGENCE

GQ. Define information, knowledge and intelligence. What is the comparison between artificial and human intelligence.

GQ. Explain Intelligence and Artificial Intelligence. How does conventional computing differ from the intelligence computing?

(1) Information

- All data are information. However, there is some part of information that is not considered as a data. Such **distinguished information** can be considered as a processed data, which makes decision making easier. Processing involves an **aggregation** of data, **calculations** of data, **corrections** on data, etc. in such a way that it generates the flow of messages.
- Information usually has some meaning and purpose that is; data within a context can be considered as information.

(2) Knowledge

Knowledge is a justified true belief. Knowledge is a **store of information** proven useful for a capacity to act.

(3) Intelligence

- Unlike belief and knowledge, intelligence is not information: it is a process, or an **innate capacity to use information** in order to respond to ever-changing requirements.
- It is a capacity to **acquire, adapt, modify, extend** and **use information** in order to solve problems. Therefore, intelligence is the ability to cope with unpredictable circumstances.

(A) Human intelligence

- Human intelligence is the intellectual capacity of humans, which is characterized by perception, consciousness, self-awareness, and volition.
- Intelligence enables humans to remember description of things and use those descriptions in future behaviours. It is a **cognitive process**.
- It gives humans the cognitive abilities to learn from concepts, understand, and reason, including the capacities to recognize patterns, comprehend ideas, plan, solve problems, and use language to communicate. Intelligence enables humans to experience and think.

(B) Artificial intelligence

- Artificial intelligence (or AI) is both the intelligence of machines and the branch of computer science which aims to create it, through "the study and design of intelligent agents" or "rational agents", whereas; an intelligence agent is a system that perceives its environment and takes actions which maximize its chances of success.
- Achievements in artificial intelligence include constrained and well-defined problems such as **games, crossword-solving and optical character recognition** and a few more general problems such as **autonomous cars**. General intelligence or strong AI has not yet been achieved and is a long-term goal of AI research.

1.2.1 Differences between Conventional Computing and Intelligence Computing

Sr. No.	Parameters	Intelligence computing	Conventional computing
1.	Solution	Does not guarantee a solution to a given problem.	Guarantees a solution to a given problem.
2.	Results	Produces results that may not be reliable or consistent.	Produces results that are consistent and reliable.
3.	Instructions	Solves the problem without specific program instructions.	Solves the given problem according to the programmer's exact instructions (algorithm).

1.3 AI TECHNIQUES

(1) An artificial intelligence technique is a method that exploits knowledge that is represented so that, the knowledge captures generalizations and situations that share properties which can be grouped together, rather than being allowed in a separate representation. It can be understood by people who must provide the knowledge; although for many programs the bulk of the data may come automatically, such as from readings.

(2) There are various techniques that have evolved and can be applied to a variety of artificial intelligence tasks. These techniques are concerned with how we represent, manipulate and reason with knowledge in order to solve problems.

(3) In short AI technique is a manner to organize and use the knowledge efficiently in such a way that –

- It should be perceivable by the people who provide it.
- It should be easily modifiable to correct errors.
- It should be useful in many situations though it is incomplete or inaccurate.

(4) AI techniques elevate the speed of execution of the complex program it is equipped with.

☞ Following are three important artificial intelligence techniques

- **Search :** Provides a way of solving problems for which no more direct approach is available.
- **Use of knowledge :** Provides a way of solving **complex problems** by exploiting the structures of the objects that are involved.
- **Abstraction :** Provides a way of separating important features and variations from many unimportant ones that would otherwise overwhelm any process.

(1) **Knowledge representation :** Knowledge representation is crucial. One of the clearest results of artificial intelligence research so far is that; solving even apparently simple problems requires lots of knowledge. Really understanding a single sentence requires extensive knowledge of both language and the context. Really understanding a visual scene similarly requires knowledge of the kinds of objects in the scene. Solving problems in a particular domain generally requires knowledge of the objects in the domain and knowledge of how to reason in that domain. Both these types of knowledge must be represented efficiently, and in a meaningful way.

(2) **Efficiency :** It is important, as it would be impossible (or at least impractical) to explicitly represent every fact that we might ever need.

(3) **Search :** Another crucial general technique required while writing an artificial intelligence programs is a search. Often there is no direct way to find a solution to some problem. However, we know how to generate possibilities.

For example, in solving a puzzle you might know all the possible moves, but not the sequence that would lead to a solution. When working out how to get somewhere one might know all the roads/buses/trains, just not the best route to get our destination quickly. Developing good ways to search through these possibilities for a good solution is therefore vital.

- **Brute force techniques**- where we generate and try out every possible solution that may work, but are often very inefficient, as there are just too many possibilities to try.
- **Heuristic techniques** are often better, where we only try the options, which we think (based on our current best guess) are most likely to lead to a good solution.

1.4 INTELLIGENT SYSTEM (IS)

(1) **Intelligent System** (IS) can be defined as the **system** that incorporates intelligence into applications being handled by machines.

(2) **Intelligent systems** perform search and optimization along with learning capabilities.

(3) Different types of machine learning such as supervised, unsupervised and reinforcement learning can be modeled in designing **intelligent systems**.

(4) **Intelligent systems** also perform complex automated tasks which are not possible by traditional computing paradigm.

(5) Various diagnostic, robotics and engineering **systems** are results of **intelligent** procedures implemented in **Intelligent System** Design.

1.4.1 Categorization of Intelligent Systems

Sr. No.	Intelligence	Description	Example
1.	Linguistic intelligence	The ability to speak, recognize, and use mechanisms of phonology (speech sounds), syntax (grammar), and semantics (meaning).	Narrators, Orators
2.	Musical intelligence	The ability to create, communicate with, and understand meanings made of sound, understanding of pitch, rhythm.	Musicians, Singers, Composers
3.	Logical-mathematical intelligence	The ability of use and understand relationships in the absence of action or objects. Understanding complex and abstract ideas.	Mathematicians, Scientists
4.	Spatial intelligence	The ability to perceive visual or spatial information, change it, and re-create visual images without reference to the objects, construct 3D images, and to move and rotate them.	Map readers, Astronauts, Physicists
5.	Bodily-Kinesthetic intelligence	The ability to use complete or part of the body to solve problems or fashion products, control over fine and coarse motor skills, and manipulate the objects.	Players, Dancers
6.	Intra-personal intelligence	The ability to distinguish among one's own feelings, intentions, and motivations.	Gautam Buddhha
7.	Interpersonal intelligence	The ability to recognize and make distinctions among other people's feelings, beliefs, and intentions.	Mass Communicators, Interviewers

You can say a machine or a system is **artificially intelligent** when it is equipped with at least one and at most all intelligences in it.

1.5 DIFFERENT DEFINITIONS OF ARTIFICIAL INTELLIGENCE

UQ. Explain different definitions of artificial intelligence according to different categories.
(Q. 1(a), Dec. 19, 5 Marks)

(I) **Strong AI :** This concept was put forward by John Searle in 1980 in his article "Minds, Brains and programs". Strong form AI provides theories for developing some form of computer based AI that can truly reason and solve problems. A strong form of AI is said to be sentient or self-aware.

Strong AI can be categorized as :

1. Human-like AI-In which the computer program thinks and reasons much like a human-mind.
2. Non-human-like AI-IN which the computer program develops a totally nonhuman sentience, and a non-human way of thinking and reasoning.

(II) **Weak AI :** Weak artificial intelligence research deals with the creation of some form of computer-based AI that cannot truly reason and solve problems. They can reason and solve problems only in a limited domain, such a machine would, in some ways, act as if it were intelligent, but it would not possess true intelligence.

There are several fields of weak AI, one of which is natural language. Much of the work in this field has been done with computer simulations of intelligence based on predefined sets of rules. Very little progress has been made in strong AI. Depending on how one defines one's goals, a moderate amount of progress has been made in weak AI.

1.6 MAJOR COMPONENTS OF ARTIFICIAL INTELLIGENCE

Artificial intelligence (AI) refers "Developing computer programs to store complex problems by application of process that analogues to human reasoning."

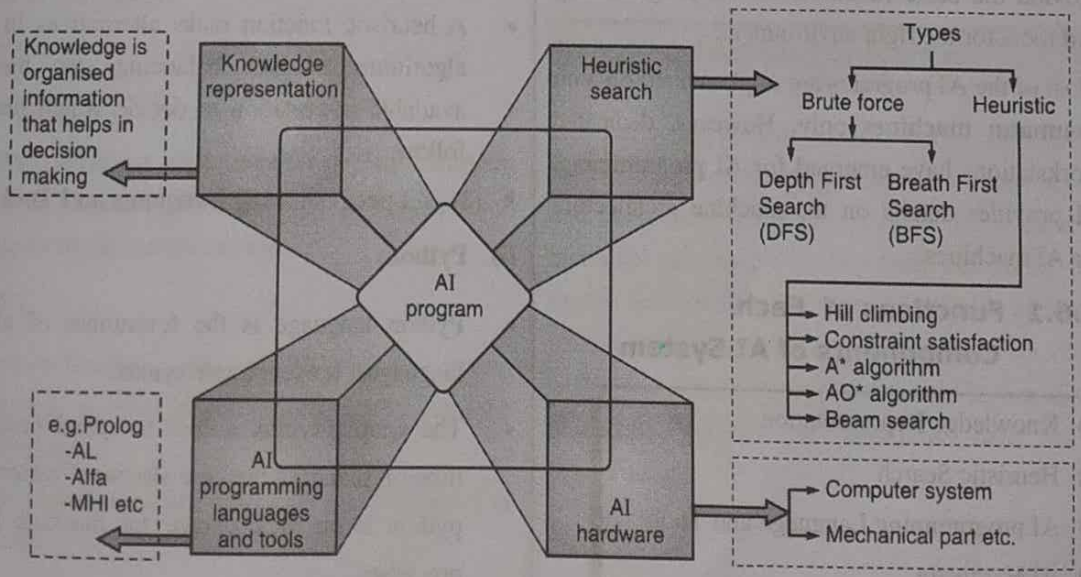


Fig. 1.6.1 : Major components of an AI system

- Any AI system has four major components :
  1. Knowledge representation.
  2. Heuristic search.
  3. AI programming languages and tool.
  4. AI hardware.
- **Definition :** Artificial intelligence (AI) refers to a "**Developing computer programs to store complex problems by application of process that analogues to human reasoning.**" The quality of the results depends on how much knowledge the system possesses. The available knowledge must be represented in a very efficient way. Hence **knowledge representation** is a vital component of the system.
- It is not merely enough that knowledge is represented efficiently. The **inference process** should also be equally good for satisfactory results. The inference process is broadly divided into **brute and heuristic search procedures.**
- Today just like we have specialized languages and programs for data-processing and scientific applications, we encounter specialized languages and tools for AI programming. AI languages provide the basic functions for AI programming and tools for the right environment.
- Most of the AI programs are implemented on **Von Neumann machines** only. However, dedicated workstations have emerged for AI programming. AI provides details on the machine architecture for AI machines.

1.6.1 Functions of Each Components of AI System

(1) Knowledge Representation
(2) Heuristic Search
(3) AI programming Language and Tool
(4) AI Hardware

(1) Knowledge Representation

- Knowledge representation in AI describes the representation of knowledge. It expresses how the beliefs, intentions, and judgements of an intelligent agent can be expressed suitably for automated reasoning. One of the primary purposes is modelling intelligent behaviour for an agent.
- It represents information from the real world for a computer to understand and then utilise this knowledge to solve complex real-life problems, like communicating with human-beings in natural language.
- It stores data in a database, and allows a machine to learn from that knowledge and behave intelligently like a human being.

(2) Heuristic Search

- In mathematical optimisation and computer science, heuristic is a technique designed for solving a problem more quickly when classic methods fail to find any exact solution.
- This is achieved by trading optionality, completeness, accuracy, or precision for speed. In a way, it can be considered a short-cut.
- A heuristic function ranks alternatives in search algorithms at each balancing step based on available information to decide which branch to follow.

(3) AI programming Language and Tool

(i) Python

- Python language is the forerunner of all other languages. It uses simple syntax.
- The simple syntax allows to spend much more time on planning the core structure, which is why python is an ideal choice for machine learning processes.

(ii) Lisp

- It is the oldest language used for AI processes. Lisp is considered as a tool in AI with its enlarged scope of turning thoughts into reality.
- The language differentiates itself from other AI languages by eying precision.

(iii) R

- This is highly efficient programming language.
- Packages like G models, RODBC, one R and Tm allow huge support for machine learning processes.

(iv) Prolog

- It is completely designed by logic. Prolog requires three important factors : rules, facts and the desired result.
- Once these requirements are provided, the language will figure out the link between the three and design an AI solution.

(v) C++

- One of the major advantages of having C++ as programming language is its processing speed.
- To run complex automated solutions efficiently, C++ is very useful.

(vi) Javascript

- For versatility, Javascript is ahead of Java.
- With continuous developments, multiple domain growth, backend use, ease of use, efficiency etc. Support the above statement.

(vii) Java

- The best benefit of using the JAVA programming language is the presence of virtual machine technology.
- Java virtual machine eases the implementation process.

(viii) Haskell

- This programming language is well-known for resolving errors, during the compilation process and even before that.
- The features like build-in memory and code reusability increase the time allotted for planning the process.

(ix) Julia

- The programming language is well-known for numerical analysis.The best feature of Julia is dynamic type system, which allows you to use the language for literally any process.
- Other features involve in-built package manager, macro programming abilities, multiple dispatch support and suitability with C functions.

(4) AI Hardware

- An AI accelerator is a specialised hardware accelerator or computer system designed to accelerate AI and machine learning applications, including neural networks and machine vision.
- Applications include algorithms for robotics, internet of things and other data-intensive or sensor-driven tasks.

1.7 FOUNDATIONS OF AI

The foundation of AI does not occur in isolation. Fields such as philosophy, linguistics, psychophysics, and theoretical computer science have exercised a historical influence over the field and today there is much discussion over the new field of cognitive science. One consequence of discussion (or dialogue) is that the criticism of positions held in one discipline frequently applies to positions held in other disciplines.

Of these, five stand out as particularly fundamental.

(1) Pre-eminence of knowledge and conceptualisation
(2) Disembodiment
(3) Kinematics of cognition are language-like
(4) Learning can be added later
(5) Uniform architecture
(6) Are knowledge and conceptualisation at the heart of AI ?
(7) Linguistic concept
(8) The logicist concept of concept
(9) Inferential abilities
(10) Knowledge and perception
(11) Growth of knowledge

(1) Pre-eminence of knowledge and conceptualisation

Intelligence requires declarative knowledge and some form of reasoning like computation is called 'cognition'. Core AI is the study of the conceptualizations of the world presupposed and used by intelligent systems during cognition.

(2) Disembodiment

Cognition and knowledge it presupposes can be studied largely in abstraction from the details of perception and motor control.

(3) Kinematics of cognition are language-like

It is possible to describe the trajectory of knowledge states or informational states created during cognition using a vocabulary very much like English or some regimented logico-mathematical version of English.

(4) Learning can be added later

The kinematics of cognition and the domain knowledge needed for cognition can be studied separately from the study of concept learning, psychological development, and evolutionary change.

(5) Uniform architecture

There is a single architecture underlying virtually all cognition.

Remark : By 'cognition' is meant to refer to computational processes that resemble both reasoning in a classical sense and computational processes that are more 'peripheral' than reasoning, such as language recognition and object identification, where the representations are not about the entities and relations, but which may usefully be construed as rules operating on representations.

(6) Are knowledge and conceptualisation at the heart of AI ?

A theory in AI is a specification of the knowledge under planning a cognitive skill. A cognitive skill is the information - based control mechanism regarding performance in some domain. It is meant to cover information sensitive activities such as problem solving, language use, decision making, routine activity, perception and some elements of motor control.

(7) Linguistic concept

- In the case of linguistics and higher vision these basic knowledge units tend more generally to be about theoretical entities. Only occasionally will there be pre-existing terms in English for them. Thus, noun phrase, sphere, pyramid and other shapes are commonsense concepts having familiar English names, but governing domain, animate movements, causal launchings and most shape representations are, for most people, novel ideas that are not part of common parlance.

- The basic knowledge units of cognition the conceptualizations under planning cognitive skills-may range, from the familiar to the exotic and theoretical.
- The basic idea that knowledge and conceptualization lie at the heart of AI stems from the seductive view that cognition is inference. Intelligent skills are composed of two parts a declarative knowledge base and an inference engine.
- The inference engine is relatively uncomplicated; it is a domain-independent program that takes as input a set of statements about the current situation plan, a fragment of the declarative knowledge base, it produces as output a stream of knowledge - based programs.
- Logicists are not unmindful of the need to explain for a system to understand a proposition, or to grasp the concepts which constitute propositions. But the party line is that this job can be pursued independently from the designer's main task of inventing conceptualizations.
- The two activities inventing conceptualization and grounding concepts-are modular. Hence the grounding issue has not historically been treated as posing a challenge that might overturn the logicist program.

(8) The logicist concept of concept

- A concept is a modular component of knowledge. If we say that kishor knows 'the pen is on the desk', and we mean this to imply that kishor grasps the fact of their being a particular pen on a particular desk, we assume that he has distinct concepts for 'pen', 'desk' and 'on'.
- We assume this because we believe that kishor must know what it is for something to be a pen, a desk, and something to be an something else.
- If concepts and conceptual schemes seem to play enough of an explanatory role at the disembodied level to be seen as robust entities, then we can study their structure without concern for their grounding.

(9) Inferential abilities

- It is the capacity of an agent to draw inferences. For instance, given the premises that the pen on the desk, is matt-black pen on the desk then a knowledgeable agent ought to be able to infer that matt-black pen is on the desk.
- Thus as long as it makes sense to view agents to be 'something' drawing inferences about a domain, or performing reason - like operations, it makes sense to suppose they have a network of concepts which structures their knowledge.

(10) Knowledge and perception

- Immanuel Kant, a german philosopher, once said, 'sensation without conception is blind'. What he meant is that 'I do not know what I am seeing, if I have no concept to categorise my experience'. It is hard to imagine how we could identify entities if we did not have concepts. The reason is because object identification is an active process.
- Perception is not a passive system. It is a method for systematically gathering evidence about the environment. We can think of it as a guide offering answers to questions about the external world. Not direct answers but perceptual answers, that serve as an evidence for an agent certain perceptual 'conjectures'. One job of the perceptual system is to ask the right questions. Our eyes jump about an image looking for clues of identity; then search for confirmation of conjectures.

(11) Growth of knowledge

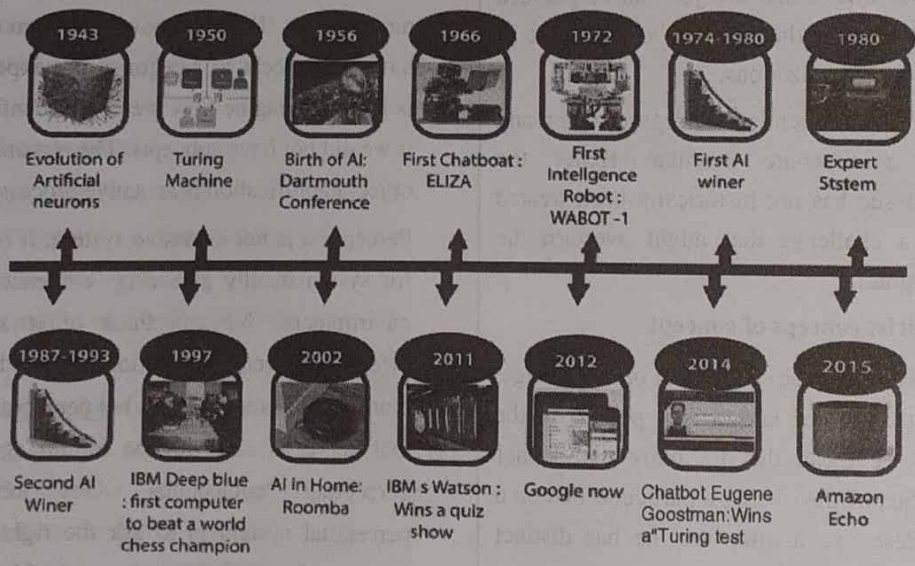
- A third feature of rational intelligence - learning - can be partially explained if we give to a system a set of disembodied concepts. Domain knowledge is much like a theory, it is a system of axioms relating basic concepts. Some axioms are definitional. Hence learning is movement along a trajectory of theories. It is a conceptual advance.
- This approach brings us closer to understanding the principles of learning, but a theory of intelligence which did not mention concepts would have to explain learning as a chance in capacities behaviourally or functionally classified. Disembodied concepts serve well enough.

1.8 HISTORY OF AI

GQ. Write a short note on : History of Artificial Intelligence ?

Artificial Intelligence is much older than you would imagine. Even there are the myths of Mechanical men in Ancient Greek and Egyptian Myths.

Following are some milestones in the history of AI which defines the journey from the AI generation to till date development.



(1A4)**Fig. 1.8.1 History of AI**

- Maturation of Artificial Intelligence (1943-1952)
  - **Year 1943 :** The first work which is now recognized as AI was done by Warren McCulloch and Walter pits in 1943. They proposed a model of **artificial neurons**.
  - **Year 1949 :** Donald Hebb demonstrated an updating rule for modifying the connection strength between neurons. His rule is now called **Hebbian learning**.
  - **Year 1950 :** The Alan Turing who was an English mathematician and pioneered Machine learning in 1950. Alan Turing publishes **"Computing Machinery and Intelligence"** in which he proposed a test. The test can check the machine's ability to exhibit intelligent behavior equivalent to human intelligence, called a **Turing test**.
- The birth of Artificial Intelligence (1952-1956)
  - **Year 1955 :** An Allen Newell and Herbert A. Simon created the "first artificial intelligence program "Which was named as **"Logic Theorist"**. This program had proved 38 of 52 Mathematics theorems, and find new and more elegant proofs for some theorems.
  - **Year 1956 :** The word "Artificial Intelligence" first adopted by American Computer scientist John McCarthy at the Dartmouth Conference. For the first time, AI coined as an academic field.
  - At that time high-level computer languages such as FORTRAN, LISP, or COBOL were invented. And the enthusiasm for AI was very high at that time.
- The golden years-Early enthusiasm (1956-1974)
  - **Year 1966 :** The researchers emphasized developing algorithms which can solve mathematical problems. Joseph Weizenbaum created the first Chatbots in 1966, which was named as ELIZA.
  - **Year 1972 :** The first intelligent humanoid robot was built in Japan which was named as WABOT-1.
- The first AI winter (1974-1980)
  - The duration between years 1974 to 1980 was the first AI winter duration. AI winter refers to the time period where computer scientist dealt with a severe shortage of funding from government for AI researches.
  - During AI winters, an interest of publicity on artificial intelligence was decreased.
- A boom of AI (1980-1987)
  - **Year 1980 :** After AI winter duration, AI came back with "Expert System". Expert systems were programmed that emulate the decision-making ability of a human expert.
  - In the Year 1980, the first national conference of the American Association of Artificial Intelligence was held at Stanford University.
- The second AI winter (1987-1993)
  - The duration between the years 1987 to 1993 was the second AI Winter duration.
  - Again Investors and government stopped in funding for AI research as due to high cost but not efficient result. The expert system such as XCON was very cost effective.
- The emergence of intelligent agents (1993-2011)
  - **Year 1997 :** In the year 1997, IBM Deep Blue beats world chess champion, Gary Kasparov, and became the first computer to beat a world chess champion.
  - **Year 2002 :** for the first time, AI entered the home in the form of Roomba, a vacuum cleaner.
  - **Year 2006 :** AI came in the Business world

till the year 2006. Companies like Facebook, Twitter, and Netflix also started using AI.

- Deep learning, big data and artificial general intelligence (2011-present)
  - **Year 2011 :** In the year 2011, IBM's Watson won jeopardy, a quiz show, where it had to solve the complex questions as well as riddles. Watson had proved that it could understand natural language and can solve tricky questions quickly.
  - **Year 2012 :** Google has launched an Android app feature "Google now", which was able to provide information to the user as a prediction.
  - **Year 2014 :** In the year 2014, Chatbot "Eugene Goostman" won a competition in the infamous "Turing test."
  - **Year 2018 :** The "Project Debater" from IBM debated on complex topics with two master debaters and also performed extremely well.
  - Google has demonstrated an AI program "Duplex" which was a virtual assistant and which had taken hairdresser appointment on call, and lady on other side didn't notice that she was talking with the machine.

Now AI has developed to a remarkable level. The concept of Deep learning, big data, and data science are now trending like a boom. Nowadays companies like Google, Facebook, IBM, and Amazon are working with AI and creating amazing devices. The future of Artificial Intelligence is inspiring and will come with high intelligence.

1.9 STATE OF ART. RISKS AND BENEFITS OF A.I.

The State of Art :

AI has so many activities in so many subfields. We mention here some of the main applications :

(a) Robotic Vehicles : A driverless robotic car named Stanley speed at 22 miles per hour and finished 132-mile course in 2005.

Stanley is a Volkswagen Touareg outfitted with cameras, radar and laser rangefinders to sense the environment and onboard software to command the steering, braking and acceleration.

(b) Speech Recognition : A traveller calling Air India to book a flight can have the entire conversation guided by an automated speech recognition.

(c) Autonomous planning and scheduling : NASA's Remote agent program became the first on-board autonomous planning program to control the scheduling of operations for a spacecraft. Successor program plans the daily operations for NASA's Mars exploration Rovers, did mission planning-both logistics and science planning.

(d) Game Playing : IBM's DEEP BLUE became the first computer program to defeat the world champion in a chess match Garry Kasparo's by a score of 3.5 to 2.5 in an exhibition match.

(e) Logistic Planning : During the Persian gulf crisis of 1991, U.S. forces deployed a dynamic Analysis and Replanning Tool, DART, to do automated logistics planning and scheduling for transportation. This involved upto 50,000 vehicles, cargo and people at a time, and had to account for starting points, desfinations, routes, and conflict resolution among all parameters. The AI planning techniques generated in hours a plan that would have taken weeks with older methods.

(F) **Robotics :** The i Robot corporation has sold over two million Roomba robotic vacuum cleaners for home use.

(g) **Machine Translation :** A computer program automatically translates from Marathi-to-English. The program uses a statistical model built from examples of Marathi-to-English, translations.

1.10 RISKS AND BENEFITS OF AI

1.10.1 Benefits of AI

1. AI machines use machine learning algorithms to reduce the cognitive abilities of human beings and solve a simple or complex problem.
2. AI-powered machines are great at doing a particular repetitive task with tremendous efficiency.

   They remove human errors from their tasks to achieve accurate results every time they do that specific task.
3. Machines can work 24 × 7, unlike humans. AI-powered chat assistants can answer customer queries and provide support to visitors every minute of the day and boost the sales of a company.
4. **Work with high accuracy :** Scientists are working to teach artificial intelligence powered machines to solve complex equations and perform critical tasks on their own so that the results obtained have higher accuracy as compared to their human counterparts.
5. **Reduce cost of training and operation :** AI uses machine learning algorithms like Deep learning and neural networks to learn new things like humans do. This way they eliminate the need to write new code every time we need them to learn new things.

1.10.2 Risks of Artificial Intelligence

(I) AI is unsustainable

Intelligent machines have characteristically high computing powers. These computer chips have rave earth materials like **selenium** as a major constituent. Also Batteries run on **Lithium**, which is again a rave element in earth's crust. They consume huge amounts of power to function, that is putting severe pressure on our power plants and again harming the environment.

(II)Lesser Jobs

Machines do routine and repeatable tasks much better than humans. Many businesses would prefer machines instead of humans to increase their profitability, thus reducing jobs that are available for the human workfore.

(III) Elon Musk

Elon Musk stated publicly that AI is the biggest threat to human civilisation in the future. Also **Stephen Hawking** has shown his disagreement with the advancement in the field of AI.

(IV) The biggest risk associated with AI is that machines may turn against humans in case they go rogue.

Conclusion

The rise of AI powered machines has eased our lives in many applications. But there is a need to emphasise on creating ethical codes and policies to ensure that the risks associated with AI are reduced to minimum.

1.11 INTELLIGENT AGENT

In artificial intelligence, an **intelligent agent (IA)** is an autonomous entity which observes through sensors and acts upon an environment using actuators (i.e. it is an agent) and directs its activity towards achieving goals.

- Intelligent agents may also learn or use knowledge to achieve their goals.
- They may be very simple or very complex, **Example:** a reflex machine such as a thermostat is an intelligent agent.
- An agent is anything that can perceive its environment through sensors and acts upon that environment through effectors.

Agent's structure can be viewed as :

(1) Agent = Architecture + Agent program.

(2) Architecture = the machinery that an agent executes on.

(3) Agent program = an implementation of an agent function.

- IA like Rahul and Gopal are examples of intelligence as they use sensors to perceive a request made by the user and automatically collect the data from the internet without the user's help.
- That can be used to gather information about its perceived environment such as weather and time. Thus, an intelligent agent is an autonomous entity which acts upon an environment using sensors and actuates for achieving goals. An intelligent agent may learn from the environment to achieve their goals.
- The term '**percept**' means the agent's perceptual inputs at any given instant. An agent's percept sequence is the complete history of everything that the agent has perceived. We illustrate this idea in the Fig. 1.11.1.

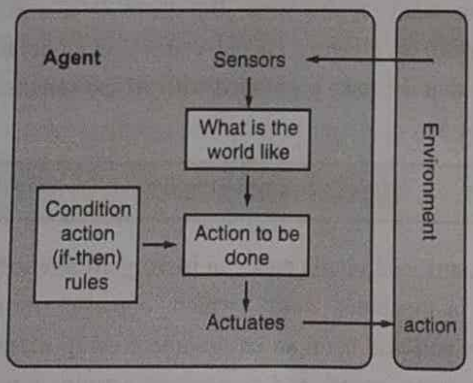


Fig. 1.11.1 : Agent's interact with environments through sensors and actuators

Thus, an agent's behaviour can be described by the '**agent function**' that maps any given percept sequence to an action.

- An intelligent agent is a programme that can make decisions or perform a service based on its environment, user input and experiences.
- These programs can be used autonomously to gather information on a regular, programmed schedule or when prompted by the user in real time.
- IA may be simple or complex - a **thermostat** is considered as an example of an intelligent agent, as is a human being, as is any system that meets the definition, such as firm, a state or total quantity.

1.12 ENVIRONMENT

The natural environment or natural world encompasses all living and non-living things occurring naturally, meaning that they are not artificial.

1.12.1 Definition of Environment

1. The circumstances, objects or conditions by which one is surrounded.
2. The complex of physical, chemical or biotic factors (**such as climate, soil and living things**) that act upon an organism or an ecological community and ultimately determine its form and survival.
3. The sum of the total of the elements, factors and conditions in the surroundings which may have an impact on the development, action or survival of an organism or group of organisms.

1.12.2 Three Types of Environment

The three types of environment are as follows :

(I) The physical environment

(II) Social environment

(III) Cultural environment.

(I) Physical Environment

(1) It consists of all components provided by nature and hence can be called as the natural environment. It is also referred to as physical environment, as it pertains to the physical requirements of life.

(2) These physical conditions are not dependent on the existence of humans. It includes natural resources, the earth's surface, mountains, plains, land, water, deserts, storms, cyclones, volcanoes, oceans, climatic factors, and so on.

(3) It also refers to biological situations such as complexities associated with plants and animals.

(II) Social Environment

(1) Social environment is the immediate environment or surroundings associated with an individual.

(2) Social environment includes all the social practices, government, occupational structure, family, friends, acquaintances, religious beliefs. It also refers to trends and developments in changes in attitudes, behaviour, and values in society.

(3) It is closely related to population, life-style, culture, tastes, customs and traditions. Social environments allow us to receive feedback, comments, and suggestions in real time concerning our developmental goals and the observations of our behaviours.

(4) Timely feedback is a critical component for personal development and provides for assessing developmental gaps and progress.

(III) Cultural Environment

(1) Cultural environments are environments shaped by human activities, such as cultural landscapes in the countryside, forests, urban areas and cities, fixed archaeological structures on land or water, constructions and built environments from different ages, along with bridges, roads and power lines.

(2) Cultural environment influences the personality traits of people in the community as well as their ideologies.

(3) The factors of cultural environment are language, social norms, religion, ethics, socio-economics, traditions, social regulations, nationalism aethetics, material culture, attitudes, values and social organisation.

Now we consider the types based on the way the environment appears to the agent.

1.12.3 The Task Environment

In task environment, we consider PEAS (Performance, Environment, Actuators, Sensors) descriptions. In designing an agent, the first step must be to specify the task environment.

Let us consider a problem : an automated taxi-driver. We mention the PEAS-description for the taxi's task environment.

1. Performance Measure

The desirable qualities are :

(i) correct destination

(ii) minimising violations of traffic laws,

(iii) minimising disturbances to other drivers,

(iv) maximising safety and passenger comfort,

(v) maximising profits,

(vi) minimising fuel consumption,

(vii)getting to correct destination.

Some of the goals may conflict, in that case tradeoff is required.

2. Environment

The agent will have to deal with a variety of roads; rural lanes, urban free ways, express highways. The roads contain traffic, pedestrians, police cars, stray animals, potholes. It could drive on right as well as on left.

But the design problem is comparatively easy if the environment is restricted.

3. Actuators

The actuators include control over the engine through steering, accelerator and braking. Also output to a display screen or voice synthesizer to communicate with other vehicles and also passengers and also to communicate in rough language, if so required.

4. Sensors

Sensors include to detect distances to other cars and obstacles, to control the vehicle on curves. It will also need engine, fuel and electrical system sensors.

1.12.4 Properties of Task-Environment

UQ. Write short notes on: Properties of Agent Task Environment.
(Q. 6(a), May 19, 5 Marks, Dec. 15, 3 Marks, Q. 5(b), Dec. 15, 5 Marks)

The task-environment in AI is vast. So, we categorise the task-environments in a small number of dimensions. First we enlist the dimensions :

(I) Fully observable versus partially observable
(II) Single agent Vs. Multiagent
(III) Static Vs-Dynamic
(IV) Deterministic Vs. Stochastic
(V) Episodic Vs. Sequential
(VI) Discrete Vs Continuous
(VII) Known Vs Unknown
(VIII) Accessible Vs Inaccessible

(I) Fully observable versus partially observable

(May 19, Dec. 15)

- If the agent sensors describes the environment at each point in detail, then the environment is fully observable. Fully observable environments are convenient because agent need not maintain track of the world.
- An environment may be partially observable because an automated taxi cannot see what other drivers are thinking. A fully observable environment is also termed as accessible environment, while partially observable environment is termed as maccessible environment.

(II) Single agent Vs. Multiagent

(May 19, Dec. 15)

- An agent playing on a violine is a single-agent environment while an agent playing a chess is in a two-agent environment. The agent design problems in the multi-agent environment are different from single agent environment.
- Playing tennis against the ball is a single agent environment. The vacuum cleaning environment is single agent environment while chess is a two-agent environment.
- The expert agent where an agent acts like an expert assistant to a user attempting to fulfil some tasks on a computer is multiple-agent environment.

(III) Static Vs-Dynamic

(May 19, Dec. 15)

- If the environment can change itself while an agent is deliberating then such environment is called a dynamic environment, else it is called a static environment.
- Static environment are easy to deal because an agent does not need to continue looking at the world while deciding for an action.
- But for dynamic environment, agents need to keep looking at the world at each action.
- Taxi driving is an example of a dynamic environment whereas playing on Piano is an example of a static environment. If the environment does not change with the passage of time, but the agent performance changes by time, then it is semi-dynamic.

Playing card-games is a perfect example of partially observable environment where a player is not aware of the card in the opponent's hand.

(IV) Deterministic Vs. Stochastic

(May 19, Dec. 15)

- If an agent's current state and selected action can completely determine the next state of the environment, then such environment is called a deterministic environment.
- A stochastic environment is random in nature and cannot be determined completely by an agent.
- In a deterministic, fully observable environment, agent does not need to worry about uncertainty.

For a given current state and action executed by agent, the next state or outcome cannot be exactly determined, e.g., if agent kicks the ball in a particular direction, then the ball may or may not be stopped by other players.

Chess : the board is fully observable, so are the opponent's moves.

Driving : the environment is partially observable because what is around the corner is not known.

(V) Episodic Vs. Sequential (May 19, Dec. 15)

- In an episodic environment, there is a series of one-shot actions, and only the current percept is required for the action. But in sequential environment, an agent requires memory of past actions to determine the next best actions.
- Episodic is an environment where each state is independent of each other. The action on a state has nothing to do with the next state.
- The sequential environment is an environment where the next state is dependent on the current action. So agent current action can change all of the future states of the environment.
- An AI that looks at radiology images to determine if there is a sickness, is an example of an episodic environment. One image has nothing to do with the next state. Chess with a clock is fully sequential.

(VI) Discrete Vs Continuous (May 19, Dec. 15)

- If in an environment there are a finite number of percepts and actions that can be performed within it, then such an environment is called a discrete environment else it is called continuous environment.
- A chess game comes under discrete environment as there is a finite number of moves that can be performed.
- A self-driving car is an example of a continuous environment. Input from digital cameras is discrete.

(VII) Known Vs Unknown (May 19, Dec. 15)

- Known and unknown are not actually a feature of an environment, but it is an agent's state of knowledge to perform an action.
- In a known environment, the results for all actions are known to the agent.
- While in unknown environment, agent needs to learn how it works in order to perform an action.
- It is quite possible that a known environment to be partially observable

For example, in Rummy card game I know the rules of the game but unable to see the card that is not turned over. An unknown environment is fully observable in a video game, the entire game is visible on the screen, but I do not know what the buttons **do till I try them.**

(VIII) Accessible Vs Inaccessible

(MU - May 19, Dec. 15)

- If an agent can obtain complete and accurate information about the state's environment then such an environment is called an Accessible environment else it is called inaccessible.
- An empty room whose state can be defined by its temperature is an example of an accessible environment.

Now ; we enlist the properties of a number of familiar environments :

1.12.5 Examples of Task Environments and their Characteristics

Task environment	Observable	Agents	Deterministic	Episodic	Static	Discrete
Chess with a clock	Fully	Multi	Deterministic	Sequential	Semi	Discrete
Taxi driving	Partially	Multi	Stochastic	Sequential	Dynamic	Continuous
Image analysis	Fully	Single	Deterministic	Episodic	Semi-dynamic	Continuous
Chess without clock	Fully	Multi	Stochastic	Sequential	Static	Discrete

1.13 RATIONAL AGENT IN AI

UQ. Define Rationality and Rational Agent. Give an example of rational action performed by any intelligent agent. **(Q. 1(c), Dec. 2015, 5 Marks)**

Rational agent in AI is an agent which has **clear preference, models uncertainty**, and acts in a way to **maximize its performance measure** with all possible actions. AI is about crating rational agents to use for same theory and decision theory for various real-world phenomenons.

Intelligent agents can be grouped into the following classes based on their **degree of perceived intelligence** and capability.

Agent	Example
Simple reflex agents	The vacuum promises to sense dirt and debris on floors and clean those areas accordingly. This is an example of a simple reflex agent that operates on the condition (dirty floors) to initiate an action (vacuuming).
Model-based agent example	Some examples of items with model-based agents aboard include **Roomba vacuum cleaner** and the **autonomous car** known as **Waymo**. Both interact with their environments by using what they know, an internal model of the world and their on-board sensors as well, to make **moment-to-moment decisions** about their actions.
Goal-based agent example	Google's **Waymo driverless cars** are good examples of a goal-based agent when they are programmed with an end destination, or goal, in mind. The car will then 'think' and make the right-decisions in order to deliver the passenger where they intend to go.

1.13.1 Concept of Rationality

The concept of rationality depends upon the following four things :

(i) The performance measure that defines the criterion of success.

(ii) Prior knowledge of the environment

(iii) Agent can perform the action

(iv) The percept sequence of the agent.

Example of Vacuum cleaner Agent

Let us consider the above example of a vacuum-cleaner agent. It cleans a square if it is dirty and moves to the next square if not;

(i) The performance measure is that it cleans a square if it is dirty, at each time.

(ii) The area of space of the floor or **'geography'** of the environment is known as **'a priori'**. Clean

squares stay clean and sucking cleans the current square. The **'Left'** and **'Right'** actions move the agent left and right.

(iii) Left, right and suck are the only available actions of the agent.

(iv) The agent correctly perceives its location, if there is dirt. Under these circumstances, we say that the agent is **'rational'**.

1.13.2 Omniscience and Rationality

(1) An omniscient's agent knows the exact outcome of its actions and act accordingly, but in practice it is impossible. In omniscience there is a total perfection. Rationality is not same as perfection. Rationality tries to maximise the expected performance while perfection gives actual total performance.

(2) Suppose I apply urgent brakes to my BMW car, as I want to take a halt. Car will come to halt instantly but the other car behind me may hit my car.

Does the definition of rationality say that it is OK to apply sudden brakes? First, it is not rational to apply sudden brakes, given the **uninformative percent sequence**; the risk of accident without looking behind is too great. Second, a rational agent will mention **'looking behind'** before applying the brakes, as it may help maximize the expected performance. This information gathering i.e. doing actions in order to modify future precepts is an important part of rationality.

(3) A rational agent is not only supposed to gather information but also to learn what it perceives. As the agent gains experience, its knowledge may be modified and augmented. If the agent relies on 'prior' knowledge of its designer and not on its own precepts, then the agent lacks autonomy.

(4) A rational agent must be autonomous - it is supposed to learn what it can, in order to compensate for partial prior knowledge. For example, a vacuum-cleaning agent that can learn to locate when and where additional dirt will appear works better than one does not learn. After having sufficient experience of its environment, the behaviour of a rational agent can become effectively independent of its prior knowledge.

1.14 INTELLIGENT AGENT

An agent is just something that acts (agent comes from the Latin agree, to do).

In artificial intelligence, an **intelligent agent (IA)** is an autonomous entity which observes through sensors and acts upon an environment using actuators (i.e. it is an agent) and directs its activity towards achieving goals.

- Intelligent agents may also learn or use knowledge to achieve their goals.
- They may be very simple or very complex, **Example :** a reflex machine such as a thermostat is an intelligent agent.
- An agent is anything that can perceive its environment through sensors and acts upon that environment through effectors.

Agent's structure can be viewed as :

(1) Agent = Architecture + Agent program.

(2) Architecture = the machinery that an agent executes on.

(3) Agent program = an implementation of an agent function.

- IA like Rahul and Gopal are examples of intelligence as they use sensors to perceive a request made by the user and automatically collect the data from the internet without the user's help.
- That can be used to gather information about its perceived environment such as weather and time. Thus, an intelligent agent is an autonomous entity which acts upon an environment using sensors

and actuates for achieving goals. An intelligent agent may learn from the environment to achieve their goals.

- The term '**percept**' means the agent's perceptual inputs at any given instant. An agent's percept sequence is the complete history of everything that the agent has perceived. We illustrate this idea in the Fig. 1.14.1.

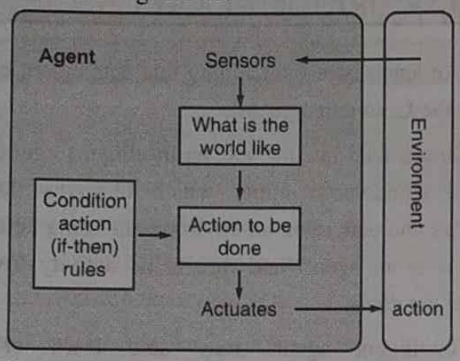


Fig. 1.14.1 : Agent's interact with environments through sensors and actuators

Thus, an agent's behaviour can be described by the '**agent function**' that maps any given percept sequence to an action.

- An intelligent agent is a programme that can make decisions or perform a service based on its environment, user input and experiences.
- These programs can be used autonomously to gather information on a regular, programmed schedule or when prompted by the user in real time.
- IA may be simple or complex - a **thermostat** is considered as an example of an intelligent agent, as is a human being, as is any system that meets the definition, such as firm, a state or total quantity.

1.14.1 Characteristics of Intelligent Agent

UQ. Define Intelligent Agent. What are the characteristics of Intelligent Agent ?
(Q. 1(a), May 18, 5 Marks)

1. **Mobility :** Using computer-Network, Intelligent agents engaged in e-commerce gather information until search parameter are complete.
2. **Goal-oriented :** Intelligent agents carry out the particular task provided by **'user statement of goals'**. It moves around from one machine to another and can react in response to their environment and takes initiative to exhibit goal directed behaviour.
3. **Independent :** Intelligent agent is self-dependent, in the sense that it functions an its own without human intervention. It makes decisions on its own and initiate them. It communicates independently with data or information and other agents and achieves the objectives and tasks an behalf of the user.
4. **Intelligent :** Intelligent agent can collect data more intelligently. They can reason out things based on the existing knowledge of its user and environment on past experiences intelligently. To evaluate condition in the external environmental intelligent agents follow present rules.
5. **Reduce net traffic :** Agents communicate and co-operate with other agents quickly. This way, they can perform the tasks, such as information searches quickly and efficiently. And network traffic gets reduce thereby.
6. **Multiple tasks :** Multiple tasks can be performed by an intelligent agent simultaneously. This helps the human from monotonous clerical work.

1.15 STRUCTURE OF INTELLIGENT AGENTS

(1) The IA structure consists of three main parts: architecture, agent function and agent programme. Architecture refers to machinery or devices that consist of **actuators and sensors**. The IA executes on this machinery. The tool allows the adjusting of image details and clarifies. Using this tool we can obtain great detail or get a smoother picture with less detail. This is the main tool to increase the contrast of the image and visualize more details of the image.

(2) A software agent has file contents, received network packages which act as sensors and display on the screen, files, sent network packets acting as actuators. A human agent has eyes, ears and other organs which act as sensors, and hands, legs, mouth and other body parts acting as actuators.

(3) The learning element is responsible for improvements. This can make a change on any of the knowledge components in the agent. One way of learning is to observe parts of successive steps in the percept sequence; from this the agent can learn how the world evolves.

(4) Agency is the capacity of individuals to act independently and to make their own free choices. The structure versus agency debate may be understood as an issue of socialization against autonomy in determining whether an individual acts as a free agent or in a manner dictated by social structure.

So far we have discussed the behaviour of the agent, i.e. the **action that is performed after a given sequence of precepts**. Now we begin to study how the inside of the agent works. The job of 'Artificial intelligence' is to design an agent program that designs the agent function-the function from precepts to action. This is 'architecture' or 'structure' of intelligent agent.

We mention four basic kinds of agent programs that cover the principles of almost all intelligent systems.

They are :

(1) Simple reflex agents
(2) Model-based reflex agents
(3) Goal-based agents; and
(4) Utility-based agents

Each of these programs combine particular component in particular ways to generate actions. Then we shall see how to convert all these agents into learning agents. This will improve the performance of their components so as to have better actions. Then we shall also describe variety of ways in which the components themselves can be represented within the agent.

1.15.1 Simple Reflex Agent

(1) In artificial intelligence, a simple reflex agent is a **'type of intelligent agent'** that performs actions based solely on the current situation, with an intelligent agent generally being one that perceives its environment and then acts. The agent cannot learn or take past percepts into account to modify its behavior.

(2) The 'simple reflex agent' works on **condition-action rule**, which means it **maps the current state to action**. Since 'simple reflex agent' is based on the present condition so it is called as condition-action rule. Problems with simple reflex agents are: very limited intelligence. No knowledge of non-perceptual parts of the state is required.

(3) Usually too big to generate and store. The simple reflex based agent is designed only to respond to the currently occurring problem. If the knowledge of the entire environment is given, then simple reflex agent is perfectly rational.

(4) This agent selects actions based on the agent's current perception, and not based on past perceptions. **For example**, if a mars lander found a rock in a specific place it needed to collect then it would collect it, **if** it was a 'simple reflex agent' **then** if it found the same rock in a different place, it would still pick it up, as it does not take into account that it has already picked it up.

(5) This is useful when a **quick automated response** is needed. Humans have a very similar reaction to fire for example; our brain pulls our hand away without thinking about any possibility that there could be danger in the path of your arm this is called as a **reflex action**.

(6) This kind of connection-action rule, written as **if** hand is in fire, **then** pulls it away. The 'simple reflex agent' has a library of such rules so that if a certain situation should arise, it is in the set of condition-action rules, the agent will know how to react with **minimal reasoning**. These agents are simple to work with but have very limited intelligence, such as picking up 2 rock samples. Refer Fig. 1.15.1.

(rectangles) : To represent the current internal state of the agent's decision process.

(ovals) : To represent the background information used in the process.

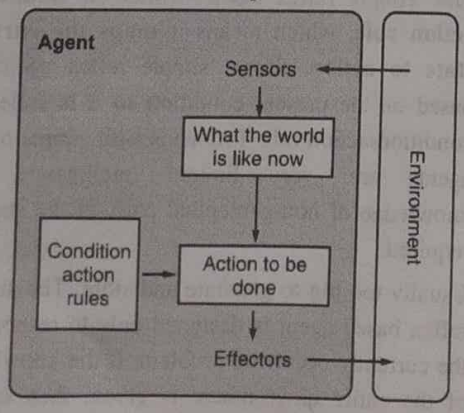


Fig. 1.15.1 : Simple Reflex Agent in AI

Function : SIMPLE–REFLEX–AGENT (percent) returns on action.

Static : Rules, a set of condition – action rules.

State ← INTERPRET – INPUT (percept)

Rule ← RULE – MATCH (State, rules)

Action ← RULE – ACTION (rule)

Return Action

1.15.2 Model-based Reflex Agent

UQ. Explain Model based Reflex agent with block diagram.

(Q. 2(a), Dec. 19, 10 Marks, Q. 2(c), Dec. 16, 5 Marks, Q. 5(b), May 16, 5 Marks)

(1) A model-based reflex agent needs **memory for storing the precept history**; it uses the percept-history to help to reveal the current unobservable aspects of the environment. An example of this IA class is the **self-screening mobile-vision**, where it is necessary to check the percept history to fully understand how the world is evolving.

(2) Model-based reflex agents are made to deal with **partial accessibility**; they do this by keeping track of the world it can see now. It does this by keeping an internal state that depends on what it has seen before, so it holds information on the unobserved aspects of the current state.

(3) A simple reflex agent selects actions based on the **agent's current perception of the world and not based on past perception**. A model based reflex agent is designed to deal with partial accessibility. They do this by keeping track of the part of the world it can see now.

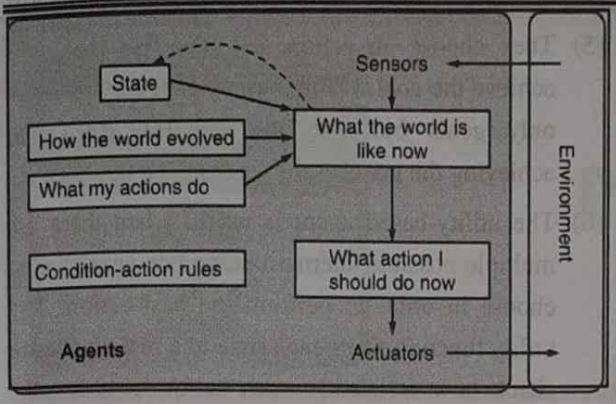


Fig. 1.15.2 : Model-based reflex agent in AI

Function REFLEX-AGENT WITH-STATE (percept)

Returns an action.

Static : State, a description of the current world state

rules, a set condition-action rules.

action, the most recent action, initially none.

State ← UPDATE – STATE (State, action, percept)

Rule ← RULE – MATCH (State, rules)

Action ← RULE – ACTION (rule)

Return Action

1.15.3 A Goal-based Reflex Agent

UQ. Explain Goal Based with block diagram. **(Q. 2(b), Dec. 18, Q. 2(B), Dec. 17, 10 Marks, Q. 1(d), May 17, 4 Marks)**

(1) A goal-based reflex agent has a goal and has a strategy to reach that goal. All actions are taken to reach this goal.

(2) More precisely, from a set of possible actions, it selects the one that improves the progress towards the goal (not necessarily the best one).

(3) A goal-based agent has an agenda. Unlike a simple reflex agent that makes decisions based solely on the current environment, a goal-based agent is capable of thinking beyond the present moment to decide the best actions to take in order to achieve its goal.

(4) Goal-based agents expand the capabilities of the model-based agent by having the **'goal'** information. They choose an action, so that they can achieve the goal.

(5) These agents may have to consider a long sequence of possible actions, before deciding whether the goal is achieved or not.

(6) A goal-based algorithm uses **searching and planning** to act in the most-efficient solution to achieve the goal.

Remark : Conclusion is a statement to Goal-based agent, but is not considered as goal-based agent.

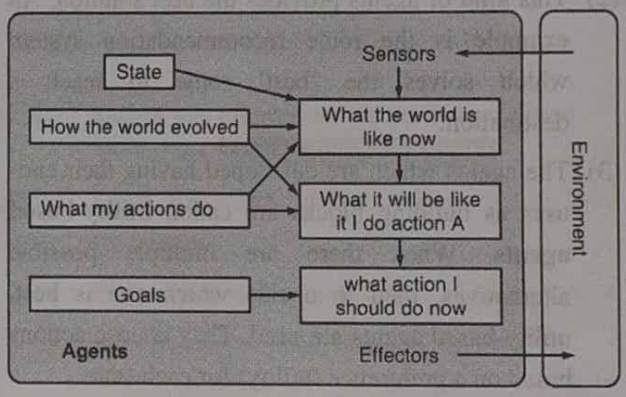


Fig. 1.15.3 : Goal-based agent diagram

1.15.4 An Utility-based Reflex Agent

UQ. Explain Utility based agent with block diagram. **(Q. 2(a), Dec. 19, 10 Marks, Q. 2(c), Dec. 16, Q. 5(b), May 16, 5 Marks, Q. 2(b), Dec. 18, Q. 2(B), Dec. 17, Q. 1(d), May 17, Q. 2(a), Dec. 19, 10 Marks)**

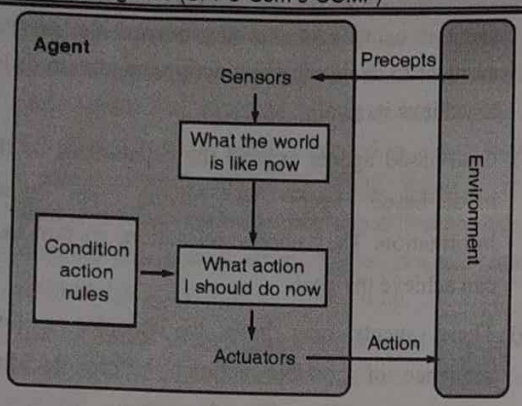


Fig. 1.15.4

(1) An utility-based reflex agent is like the goal-based agent but with measure of **'how much happy'** an action would make it rather than the **goal-based binary feedback [happy, unhappy].**

(2) This kind of agents provides the best solution. An example is the route recommendation system which solves the 'best' route to reach a destination.

(3) The agents which are developed having their end-uses as building blocks are called **utility-based agents**. When there are multiple possible alternatives, then to decide which one is best, utility-based agents are used. They choose actions based on a preference (utility) for each state.

(4) Goal-based agents are important as they are used to expand the capabilities of the model-based agent by having the 'goal' information.

(5) They choose an action, in order that they will achieve the goal. Utility based agent-act based not only goals but also the samples, thanks to achieving the goal.

(6) The utility-based agent is useful when there are multiple possible alternatives, and an agent has to choose in order to perform the best action. The utility function maps each state to a real number to check how efficiently each action achieves the goals.

(7) In artificial intelligence, utility function **assigns values to certain actions** that the AI can take. An AI agent's preferences over all possible outcomes can be captured by a function that maps the outcomes to a utility value, the higher the number, the more that agent likes that outcome.

(8) In Economics, the utility function **measures the welfare or satisfaction of a consumer as a function of the consumption of real goods**, such as food or clothing. Utility function is widely used in rational choice theory to analyze human behavior.

(9) Utility theory bases its beliefs upon individual's preferences. It is a theory postulated in economics to explain behavior of individuals based on premise; people can consistently rank order their choices depending upon their preferences. We can state that individual's preferences are intrinsic.

Chapter Ends...

❑❑❑

UNIT II

CHAPTER 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.

2.1 PROBLEM SOLVING

GQ. Briefly explain problem solving.

Problem solving consists of using generic or ad hoc methods, in an orderly manner, for finding solutions to problems. Some of the problem-solving techniques use artificial intelligence, computer science, engineering, mathematics, or psychoanalysis.

Problems can also be classified into two different types: **ill-defined and well-defined,** from which appropriate solutions are to be made.

1. **Ill-defined** problems are those that **do not have clear goals**, solution paths, or expected solution.
2. **Well-defined** problems **have specific goals**, clearly defined solution paths, and clear expected solutions. These problems also allow for more initial planning than ill-defined problems.

   Being able to solve problems sometimes involves dealing with **pragmatics (logic)** and **semantics** (interpretation of the problem). It requires the ability to understand what the goal of the problem is and what rules could be applied to represent the key to solving the problem. Sometimes the problem requires some abstract thinking and coming up with a creative solution.

Problem types

1. **Deterministic, fully observable** ⇒ single-state problem

   Agent knows exactly which state the problem will be in; it's solution is a sequence.
2. **Non-observable** ⇒ conformant problem

   Agent may have no idea where it is; solution (if any) is a sequence.
3. Nondeterministic and/or partially observable ⇒ contingency problem percepts provide new information about current state.
4. **Unknown state space** ⇒ exploration problem ("online") Solution is a tree or policy often interrelated search, and execution.

2.1.1 Problem Solving Agent

GQ. Describe problem solving agent.

Problem-solving agents : A goal formulation, based on the current situation and the performance measure is required for problem solving.

- **Problem formulation** is the process of deciding what actions and states to consider, given a goal. In general, an agent with several options for action of unknown value can decide what to do by first examining different possible sequences of actions (that lead to states of known value) and then choosing the best sequence.
- A search algorithm takes a problem as input and returns a solution in the form of an action sequence.

2.1.2 Simple Problem-Solving Agent

```
Function SIMPLE-PROBLEM-SOLVING- AGENT
(percept) returns an action Persistent:
        seq an action sequence, initially empty,
        state, some description of the current world state,
        goal; a goal initially null,
        problem, a problem formulation,
        state ← UPDATE-STATE(state percept)
    If seq. is empty then
        goal ← FORMULATE-GOAL(State)
        problem ← FORMULATE-PROBLEM
                (State, goal)
        seq. ← SEARCH( problem}
    If seq. = failure then return a null action action
← FIRST(seq.)
    seq. ← REST(seq.) return action
```

Unit II In Sem.

2.1.3 Types of Performance Measures

(a) Workload or output measures. These measures indicate the amount of work performed or number of services received.

(b) Efficiency measures.

(c) Effectiveness or outcome measures.

(d) Productivity measures.

2.2 SEARCHING

GQ. What is searching ?

- Search plays a major role in solving many artificial intelligence (AI) problems. Search is a **universal problem-solving mechanism** in AI.
- In many problems, sequence of steps required to solve a problem is not known in advance but must be determined by systematic trial-and-error exploration of alternatives.
- **Search techniques** try to "pre-play" the game by evaluating the future states (game tree search) and may use also heuristic to prone bad choices or speed things up. They theoretically can make an **exact and perfect** choice, but are **slow**.
- The problems that are addressed by AI search algorithms fall into three general- classes:
  1. Single-agent path-finding problems
  2. Two players games
  3. Constraint- satisfaction problems

2.2.1 Node Representation in Search Tree

A binary search tree (BST) is a tree in which all the nodes follow the below mentioned properties

(i) The value of the key of the left sub-tree is less than the value of its parent (root) node's key.

(ii) The value of the key of the right sub-tree is greater than or equal to the value of its parent (root) node's key.

Thus, BST divides all its sub-trees into two segments; the left sub-tree and the right sub-tree and it can be defined as left-sub-tree (keys) < node (key) ≤ right_sub-tree (keys)

☞ Representation

BST is a collection of nodes arranged in a way where they maintain BST properties. Each node has a key and has an associated value.

While searching, the desired key is compared to the keys in BST and if found, the associated value is retrieved.

We mention below a pictorial representation of BST

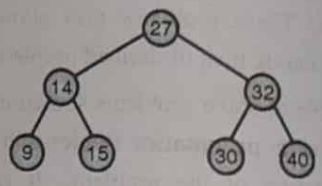


Fig. 2.2.1

Note that the root node key (27) has all less-valued keys on the left sub-tree and the higher valued keys on the right sub-tree.

☞ Basic Operation

The basic operations of a tree

(i) **Search :** Searches an element in a tree

(ii) **Insert :** Inserts an element in a tree

(iii) **Pre-order Traversal :** Traverses a tree in a pre-order manner

(iv) **In-order Traversal :** Traverses a tree in an in-order manner

(v) **Post-order Traversal :** Traverse a tree in a post-order manner.

(vi) **Remark :** There must be no duplicate nodes.

☞ **Nodes representing in search tree**

Remark

Let us suppose we want to search for the number :

(i) We start at the root

(ii) We compare the value to be searched with the value of the root.

(iii) It is equal, we complete the search.

(iv) If it is lesser, we need to go to the left sub-tree, since in a binary subtree all the elements in the left subtree are lesser and all the elements in the right subtree are greater.

(v) In this traversal, at each step we discard one of the sub-trees.

(vi) We go on reducing like this till we find the element or till our search is reduced to only one node.

(vii) The search here is a binary search and hence is called as Binary search tree.

Note : After reaching the end, just insert that node at left (it less than current), else right.

2.3 PROBLEM SOLVING AGENTS

(1) The problem - solving agent performs by defining problems and several solutions. Problem solving agent employees a number of techniques such as a tree, heuristic algorithms to solve a problem.

(2) The reflex agent of AI directly maps states into action.

When the state of mapping is too large, then the problem gets dissolved and sent to a problem - solving domain. It breaks the large stored problem into smaller storage area and resolves one by one. The final action will be the desired outcome.

(3) Intelligent agent organises finite number of steps to formulate a target or goal which require some action to achieve the goal.

(4) Problem formulation is one of the most important steps of problem-solving. Agent has to decide what action to be taken to achieve the goal.

Components to formulate the associated problem :

(i) **Initial state :** In this state, new methods also initialise problem solving by a specific class.

(ii) **Action :** Here the agent performs all possible actions with a specific class taken from the initial state

(iii) **Transition :** Here the agent integrates the actual action by the previous action stage and collets the final stage to forward to the next stage.

(iv) **Goal test :** When the goal is achieved, action steps and then forwards into the next stage to determine the cost to achieve the goal.

(v) **Path costing :** Here the agent calculates all hardware, software and human working cost.

Thus, a problem - solving agent is a goal - driven agent and focuses on satisfying the goal.

2.3.1 Simple Problem-Solving Agent

Function SIMPLE-PROBLEM-SOLVING-AGENT (percept) returns an action **Persistent:**

seq. an action sequence, initially empty,

state, some description of the current world state,

goal; a goal initially null,

problem, a problem formulation,

state ← UPDATE-STATE(state percept)

If seq. is empty then

goal ← FORMULATE-GOAL(State)

problem ← FORMULATE-PROBLEM (State, goal)

seq. ← SEARCH(problem}

If seq. = failure then return a null action action ← FIRST(seq.)

seq. ← REST(seq.) return action

Unit II In Sem.

Method

(i) Formulate a problem and a goal.

(ii) Formulate a sequence of actions and execute the sequence of action one - by - one.

(iii) When this is complete, form another goal and proceed.

(iv) Agent uses search-procedure to arrive at a goal.

(v) Agent then uses the solution to guide its actions. Then agent formulates a **new goal**.

Consider an example of a car from Pune to Surat (Fig. 2.3.1)

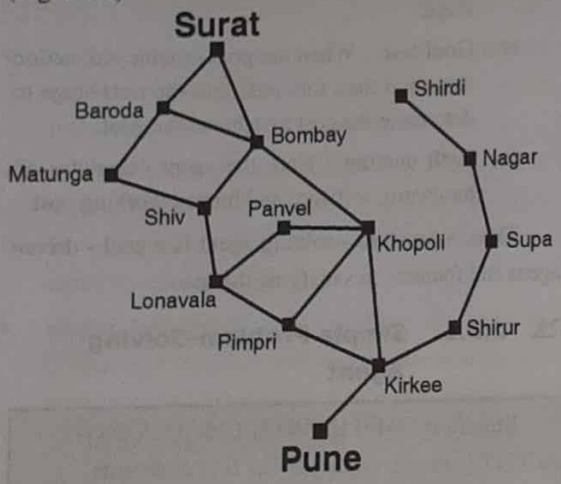


Fig. 2.3.1

2.3.2 Well - Defined Problems and Solutions

Problem - solving consists of using generic or ad-hoc methods in an orderly manner.

Problems can be classified into two different types : ill-defined and well-defined.

1. **Ill defined problems** are those that do not have clear goals, solution paths, or expected Solutions.
2. **Well - defined problems** have specific goals, clearly defined solution paths, and clear expected solutions.

These problems also allow for more initial planning than ill-defined problems.

2.3.2(A) A Problem can be Defined by Five Components

(i) The initial state

It is the state where the agent starts in For example, the initial state for agent is Pune, can be written as In (Pune)

(ii) Action

A description of the possible actions available to the agent. Given a particular state p, ACTIONS (p) returns the set of actions that can be executed in p. For example, from the state (Pune), the actions could be

{Go (Kirkee), Go (Lonavala), Go (Panvel)}

(iii) Transition model

The formal name for the description of each action is 'transition model'. It is specified by Result (p, a), that results the state from doing action a in p.

We use the term successor to refer to any state reachable from a given state by a single action.

For example, Result {In (Pune), Go (Kirkee)} = In (Kirkee). The initial state, actions and transition model define the **state space** of the problem. It is the set of all states reachable from the initial state by any sequence of actions.

The state space forms a **directed network** or **graph** in which the **nodes** are **states** and **links** between **nodes** are **actions**. A path in the **state space** is a sequence of states connected by a sequence of actions.

(iv) The goal test

It determines whether a given state is a goal-test. There may be an explicit set of possible goal - states, and the test checks whether the given state is one of them.

Sometimes the goal is specified by an abstract property. For example, in chess, the goal is to reach a state called "**checkmate**", where the opponent's king is under attack and cannot escape.

(v) Path cost

It is a function which assigns numeric value or cost to each path. For the agent going to Surat, time is the essence, so the cost of path may be its length in kilometres. Here the cost of the path is the sum of the costs of individual actions along the path.

The **step cost** of taking action a in state p to reach state p′ is denoted by c(p, a, p′). The step costs are non-negative.

2.4 EXAMPLE PROBLEMS

We enlist some of the best known problems, distinguishing between **toy** and **real-world** problems.

(1) Toy Problem

A toy problem is intended to illustrate various problem-solving methods. Here we mention exact description of the problem and hence is usable to compare the performance of algorithm.

(2) Real-world problem

The solution of these problems are useful and people actually care about these problems.

These problems cannot have a unique solution but we can give the general flavour of their formulations.

2.4.1 Toy-problem Example

(a) Vacuum-cleaner World

- This world-problem is very simple so that we can invent many variations in it.
- This world has just two locations : squares A and B. The vacuum agent perceives which square it is in and whether there is dirt in the square. It can move left, move right, suck up the dirt, or do nothing.
- One very simple agent function is : if the current square is dirty, then suck; otherwise, move to the other square.

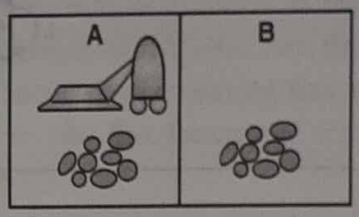


Fig. 2.4.1

We formulate the problem as follows :

(1) **States :** The state is determined by the agent location and the dirt locations. The agent is in one of the two locations. And each of them may or may not contain dirt.

Thus, there are $2 \cdot 2^2 = 8$ possible world states.

For n locations, it will have $n \cdot 2^n$ states

(2) **Initial state :** Any state can be taken as the initial state

(3) **Actions :** Each state has three actions : **Left, Right, Suck**

(4) **Transition Model :** The actions have their expected effects, except that moving left in the leftmost square, moving right in the rightmost square and sucking in a clean square have no effect.

(5) **Goal Test :** It checks whether all squares are clean.

(6) **Path cost :** Each step costs 1, so the path cost is the number of steps in the path.

Unit II In Sem.

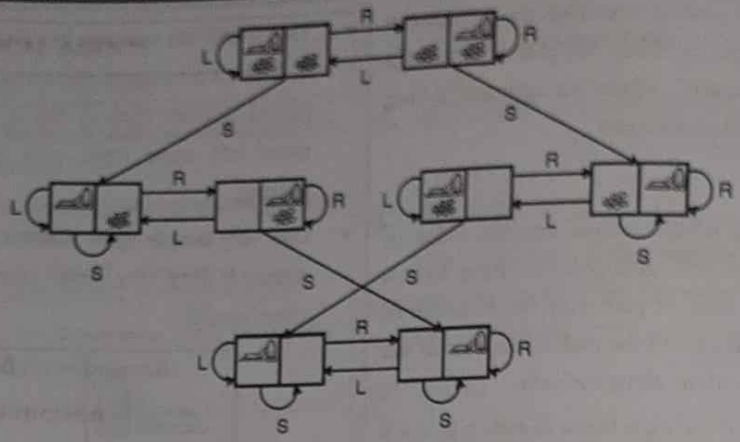
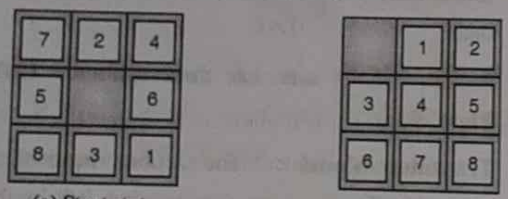


Fig. 2.4.2

This toy problem has discrete locations, discrete dirt, reliable cleaning and it never gets dirtier.

2.4.2 The 8 Puzzle Problem

- It consists of a 3 × 3 board with eight numbered tiles and a blank space.
- A tile adjacent to the blank space can slide into the space.
- The object is to reach a specified goal state, such as shown on the right of the figure.



(a) Start state (b) Goal state

Fig. 2.4.3

We form the standard **formulations.**

(1) **States :** The location of each of the eight tiles and the blank in one of the nine squares is described by a state.

(2) **Initial state :** Any state can be taken as the initial state.

(3) **Actions :** The actions as movements of the blank space Left, Right, Up or Down are defined by the simplest formulation.

(4) **Transition Model :** Given a state and action, this returns the resulting state. For example, if we apply **right** to the start state then the resulting state has 6 and the blank state is switched.

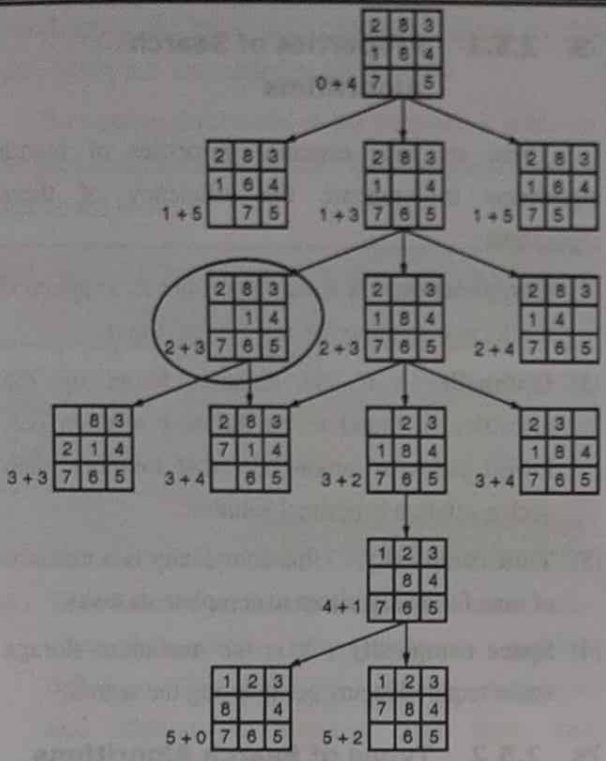
(5) **Goal Test :** The test checks whether the state matches the goal configuration.

(6) **Path cost :** Each step cost 1, so the path cost is the number of steps in the path.

Here we are interested in the description of the rules of the puzzle, avoiding all the details of physical manipulation.

The 8th puzzle problem

The puzzle can be solved by moving the tiles one by one in the single empty space and achieving the goal state. One can take only one step at a time and no diagonal move.



2.4.3 8-Queens Problem

The goal of 8-Queens Problem is to place eight queens on a chessboard such that no queen attacks any other. A queen can attack any piece in the same row, column or diagonal.

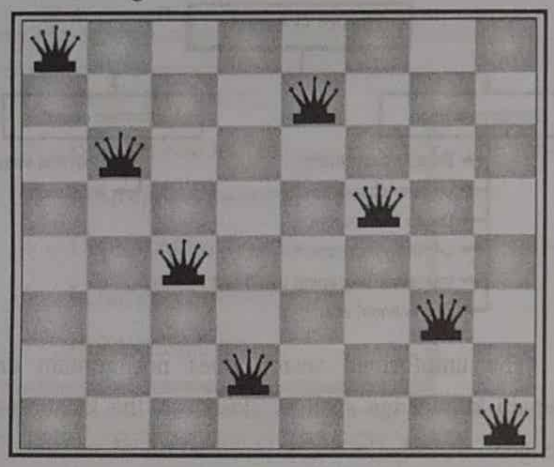


Fig. 2.4.4

In Fig. 2.4.4 the queen in the rightmost column is attacked by the queen at the top left.

(This attempted solution is a failure.)

Here there are two main kinds of formulations.

(1) An incremental formulations

This involves operators that **augment** -the state description, beginning with an empty state ; for this problem, it means that each action adds a queen to the state.

(2) A complete state formulation

This starts with all 8 queens on the board and moves them around. In either case, the path cost is of no interest because only the final state counts. We form the first incremental formulation as follows :

(i) **States :** Any arrangement of 0 to 8 queens on the board is a state.

(ii) **Initial state :** No queens on the board.

(iii) **Actions :** Add a queen to any empty square.

(iv) **Transition model :** Returns the board with a queen added to the specified square.

(v) **Goal test :** 8 Queens are on the board, no one is attacked.

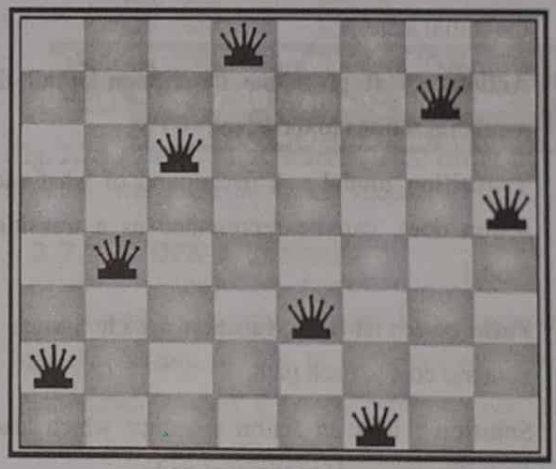


Fig. 2.4.5

In this formulation, there are 64 · 63 57 = 1.8×10^{14} possible sequences. Here the Queens are safe; no queen can attack the other.

2.5 SEARCH ALGORITHMS

Search algorithms are one of the most important areas of Artificial intelligence. We mention below search algorithms in AI.

Search Algorithm Terminologies

Search

Searching is a step by step procedure to solve a search-problem in a given search space. A search problem can have three main factors :

(a) **Search space :** Search space represents a set of possible solutions, which a system may have.

(b) **Start state :** It is a state from where agent begins the search.

(c) **Goal Test :** It is a function which observe the current state and returns whether the goal test is achieved or not.

- **Search Tree :** A tree-representation of search-problem is called search tree. The root of the search tree is the root node which corresponds to the initial state.
- **Actions :** It gives the description of all the available actions to the agent.
- **Transition model :** A description of what each action does, can be represented as a transition model.
- **Path cost :** It is a function which assigns a numeric cost to each path.
- **Solution :** It is an action sequence which leads from the start node to the goal node.
- **Optimal solution :** A solution having the lowest cost among all solutions.

2.5.1 Properties of Search Algorithms

There are four essential properties of search algorithms to compare the efficiency of these algorithms :

(1) **Completeness :** A search algorithm is complete if it returns a solution for any random input.

(2) **Optimality :** If the solution found by the algorithm is found to be the best solution (i.e. lowest path cost) among all other solutions, then such a solution is optimal solution.

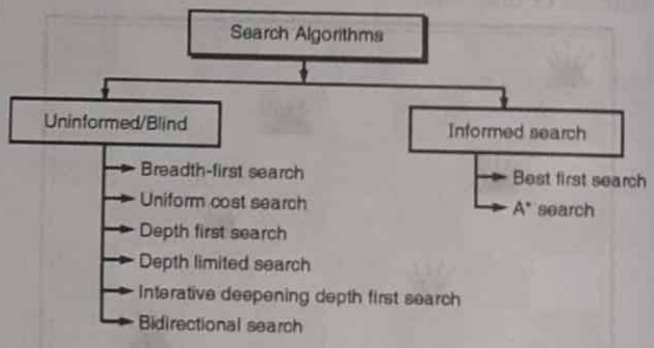
(3) **Time complexity :** Time complexity is a measure of time for an algorithm to complete its tasks.

(4) **Space complexity :** It is the maximum storage space required at any point during the search.

2.5.2 Types of Search Algorithms

Depending on the search problems, we classify the search algorithms into :

1. Uninformed (Blind search) algorithms and
2. Informed search (Heuristic search) algorithms.



The uninformed search does not contain any domain knowledge such as closeness, the location of the goal. It only includes information about how to traverse the tree and how to identity leaf and goal nodes.

Uninformed search applies a way in which search tree is searched without any information about the

search space like initial state operators and test for the goal, hence it is also called as blind search.

It examines each node of the tree until it achieves the goal node. It can be divided into **five** main types as mentioned above.

2.6 UNINFORMED SEARCH (BLIND SEARCH)

GQ. What is blind or uninformed search ?

UQ. Explain with example various uninformed search techniques.

(Q. 3(a), May 17, 10 Marks)

(1) They have no additional information about states other than those provided in the problem definition. They can only **generate successors and distinguish between goal state and non- goal state**.

(2) These are commonly used search procedures which explore all the alternating options during the search process. They do not have any dome in specific knowledge. All they need are the initial state, the final state and a set of legal operators.

Most important search techniques are as follows :

1. Depth first search.
2. Breadth first search.
3. Uniform-cost search.
4. Depth-limited search.
5. Iterative deepening search.
6. Bidirectional search.

2.7 DEPTH FIRST SEARCH (DFS)

(1) DFS search is the **distributive file system**. The distributive file function provides the ability to logically group shares on multiple servers and to transparently link shares into a single namespace. DFS organizes shared resources on a network in a tree-like structure.

(2) DFS is a file-system with data stored on a server. The data is accessed and processed on if it was stored on the local client machine.

(3) The DFS makes it convenient to share information and files among users on a network in a controlled and authorised way.

(4) DFS is used in topological **sorting, scheduling problems, cycle detection in graphs and solving puzzles**, other applications involve analysing networks **e.g.** testing, if a graph is bipartite

(5) Depth First Search (DFS) is an algorithm for traversing or searching tree or graph data structures.

(6) The algorithm starts at the root node (in case of a graph, selecting some arbitrary node as the root node) and explores as far as possible along each branch before backtracking. Refer Fig. 2.7.1.

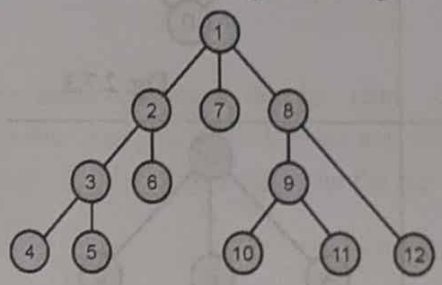


Fig. 2.7.1 : Depth First Search (DFS) order in which the nodes are visited

2.7.1 DFS Algorithm

- DFS algorithm traverses a graph in a depthward motion and uses a stack to remember to get the next vertex to start a search, when a dead end occurs in any iteration.
- Refer Fig. 2.7.2. In this example, DFS **algorithm** traverses from S to A to D to G to E to B first, then to F and lastly to C. It follows the following rules :

Unit II In Sem.

- **Rule 1 :** Visit the adjacent unvisited vertex. Mark it as visited and display it.

↓

- **Rule 2 :** If no adjacent vertex is found, choose the vertex from the stack. (There will appear all the vertices from the stack, which do not have adjacent vertices).

↓

- **Rule 3 :** Repeat Rule 1 and Rule 2 till the stack becomes empty.

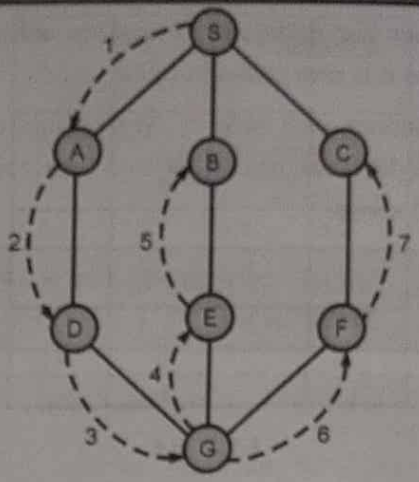
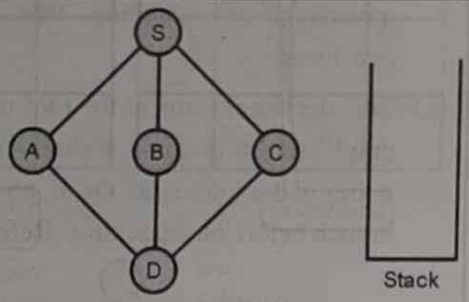
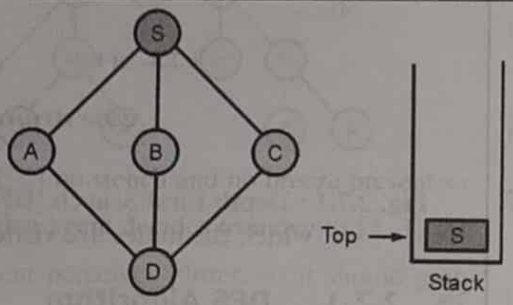
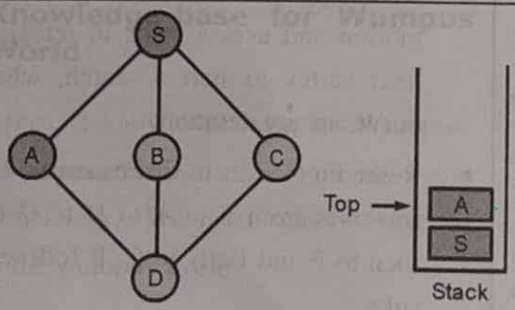
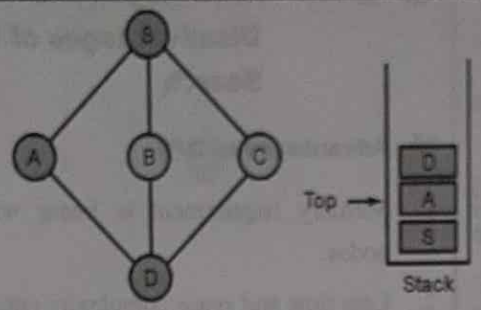
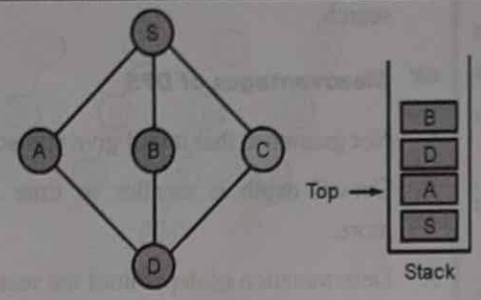
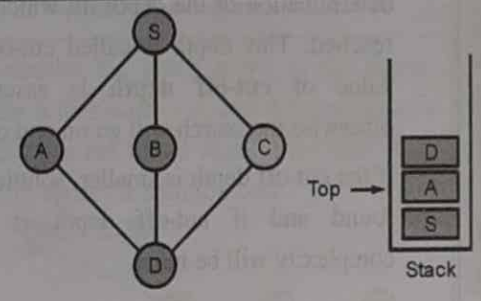
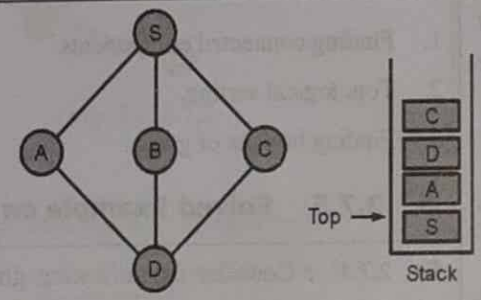


Fig. 2.7.2

Step	Traversal	Description
1.	 Fig. 2.7.3	Initialise the stack.
2.	 Fig. 2.7.4	Mark S as visited and put it onto the stack. Explore any unvisited adjacent node from S. We have three nodes and we can pick any of them. Here we take the node in alphabetical order.
3.	 Fig. 2.7.5	Mark A as visited and put it onto the stack. Explore any unvisited adjacent node from A. Bothe S and D are adjacent to A, but we want unvisited nodes only.

Step	Traversal	Description
4.	 Fig. 2.7.6	Visit D and mark it as visited and put onto the stack. Here we have B and C nodes, which are adjacent to D and both are unvisited. But again we choose in an alphabetical order.
5.	 Fig. 2.7.7	We choose B, mark it as visited and put onto the stack. Here B does not have any unvisited adjacent node. So, we pop B from the stack.
6.	 Fig. 2.7.8	We check the stack top for return to the previous node and check if it has any unvisited nodes. Here we find D to be on the top of the stack.
7.	 Fig. 2.7.9	Only unvisited adjacent node is from D is C. So we visit C, mark it and put it into the stack.

- As C does not have any unvisited adjacent node, so we keep popping the stack till we find a node that has an unvisited adjacent node. Here, there is none and the stack is empty and the program is over.

2.7.2 Performance Measures of DFS

The performance measuring factors of an algorithm are as follows :

- The two most common measures are : speed and memory usage; other measures could include transmission speed, temporary disk usage, long-term disk usage, power consumption, total cost of ownership, response time to external stimuli, etc.
- Performance measure is generally defined as regular measurement of outcomes and results which generates reliable data on the effectiveness and efficiency of programs.
- There are four ways to measure the performance of an algorithm :

  (i) Completeness : DFS is complete if the search tree is finite, it implies that for a given finite search, DFS will have a solution if it exists.

  (ii) Optimality : DFS is not optimal, it means that the number of steps in reaching the solution, or the cost spent in reaching it is high.

  (iii) Time complexity : The time complexity of DFS, if the entire tree is traversed, is O (V) where V is the number of nodes.

  For a directed graph, the sum of the sizes of the adjacency lists of all nodes is E. So, the time complexity in this case is

  $$O(V) + O(E) = O(V + E)$$

  For an undirected graph, each edge appears twice.

  (iv) Space complexity : For DFS, which goes along a single 'branch' all the way down and uses a stack implementation, the height of the tree matters. The space complexity for DFS is O (h) where h is the maximum height of the tree.

2.7.3 Advantages and Disadvantages of Depth First Search

Advantages of DFS

1. Memory requirement is linear with respect to nodes.
2. Less time and space complexity rather than BFS.
3. Solution can be found out without much more search.

Disadvantages of DFS

1. Not guarantee that it will give you solution.
2. Cut-off depth is smaller so time complexity is more.
3. Determination of depth until the search proceeds.
4. The major drawback of depth-first search is the determination of the depth till which the search is reached. This depth is called cut-off depth. The value of **cut-off depth** is essential because otherwise the search will go on and on.

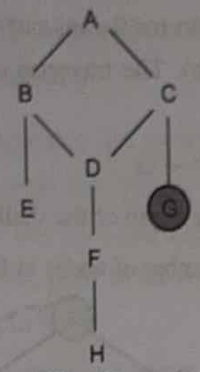
   If the cut-off depth is smaller, solution may not be found and if cut-off depth is large, time-complexity will be more.

2.7.4 Applications of DFS

1. Finding connected components.
2. Topological sorting.
3. Finding bridges of graph.

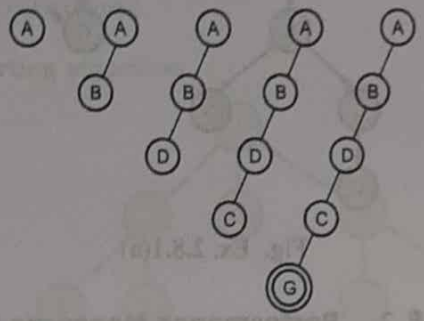
2.7.5 Solved Example on DFS

Ex. 2.7.1 : Consider the following graph shown in Fig. 2.7.1 starting from 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.



(1B13)**Fig. Ex. 2.7.1**

☑ **Soln. :**



(1B14)**Fig. Ex. 2.7.1(a)**

Ex. 2.7.2 : Consider the graph given in the figure. Assume that the initial state is A and the goal state is G. Find a path from the initial state to the goal state using DFS. Also report the solution cost.

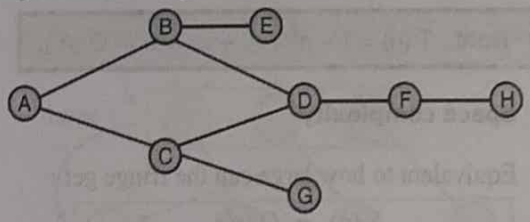


Fig. Ex. 2.7.2

☑ **Soln. :**

A is given initial state and G is the goal node.

- **Step (I) :** Place the starting node into the stack

A	

- **Step (II) :** Now the stack is not empty and A is not our goal node. Hence we move to next step.
- **Step (III) :** The neighbours of A are B and C.

B	C	A

- **Step (IV) :** Now, B is top node of the stack. Its neighbours are E and D.

E	D	B

- **Step (V) :** E is top node of the stack. We find its neighbour, but there is no neighbour of E in the graph, so

B	D	E

- **Step (VI) :** Now, D is top node; and its neighbours are F and C; we push D into the stack.

F	C	D

- **Step (VII) :** Now, G is our top node of the stack, which is our goal node.
- **Step (VIII) :** solution cost is

  $A \rightarrow B \rightarrow E \rightarrow D \rightarrow G$

2.8 BREADTH-FIRST SEARCH (BFS)

GQ. Comment upon statement that breadth-first search is a special case of uniform cost search.

GQ. Explain breadth first search with its algorithm.

(1) BFS stands for Breadth-First Search is a vertex based technique for finding a shortest path in graph. In BFS, one vertex is selected at a time when it is visited and marked then its adjacent are visited and stored in the queue. It is slower than DFS.

(2) BFS is the core of many graph analysis algorithms and it is used in many problems, such as social network, computer network analysis and data organization.

(3) BFS involves search through a tree one level at a time. We traverse through one entire level of children nodes first, before moving onto traverse through the grand children nodes.

(4) Breadth first search is an algorithm for node that satisfies a given property. It starts at the **tree root** and explores all nodes at the present depth prior to moving on to nodes at the next depth level.

(5) BFS uses **Queue-data structure** for finding the shortest path. BFS can be used to find **single source shortest path** in an un-weighted graph, because in BFS, we reach a vertex with minimum number of edges from a source vertex.

2.8.1 BFS Traversal Algorithm

- **Step 1 :** Add a node / vertex from the graph to a queue of nodes to be 'visited'.,
- ↓
- **Step 2 :** Visit the topmost node in the queue, and mark it as such.
- ↓
- **Step 3 :** If that node has any neighbours, check to see if they have been 'visited' or not.
- ↓
- **Step 4 :** Add any neighbouring nodes that still need to be 'visited' to the queue.

Illustrative Example

Ex. 2.8.1 : Which solution would BFS find to move from node S to node G if run on the graph below ?

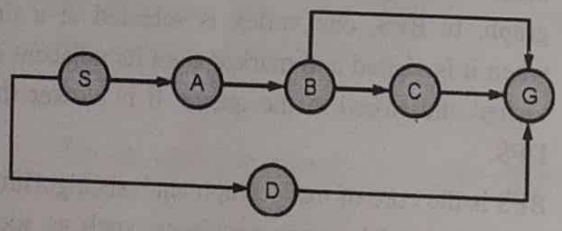


Fig. Ex. 2.8.1

Soln. :

The equivalent search tree for the above graph is as follows :

As BFS traverses the tree "shallowest node first", it would always pick the shallower branch until it reaches the solution (or it runs out of nodes, and goes to the next branch). The traversal is shown by dotted line.

Path : S → D → G

= the depth of the shallowest solution.

= number of nodes in level.

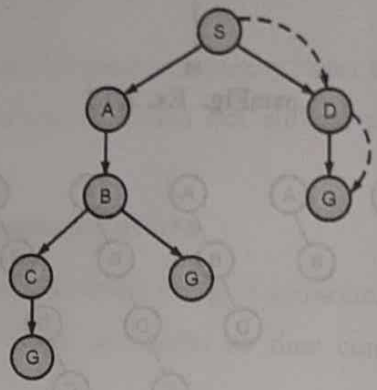


Fig. Ex. 2.8.1(a)

2.8.2 Performance Measures for BFS

Time complexity

Equivalent to the number of nodes traversed in BFS until the shallowest solution.

Note : $T(n) = 1 + n^2 + n^3 + ... + n^s = O(n^s)$.

Space complexity

Equivalent to how large can the fringe get :

$$S(n) = O(n^s)$$

Completeness

BFS is complete, meaning for a given search tree, BFS will come up with a solution if it exists.

Optimality

BFS is optimal as long as the costs of all edges are equal.

2.8.3 BFS Algorithm for Extra Memory

- BFS is an algorithm for searching a tree data structure for a node that satisfies a given property.
- It starts at the tree root and explores all nodes at the present depth prior to moving on to the nodes at the next depth level.
- Extra memory, usually a queue, is needed to keep track of the child nodes that were encountered but not yet explored.

Sorting algorithm

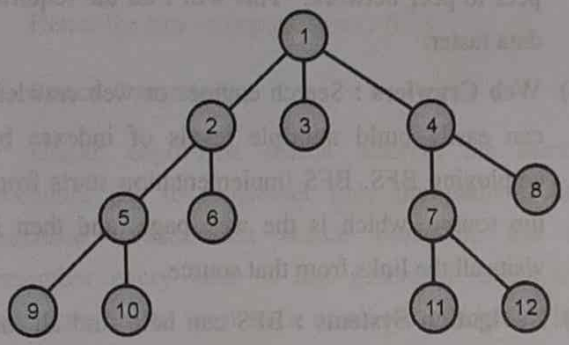


Fig. 2.8.1 : Order in which nodes are expanded

Concept diagram

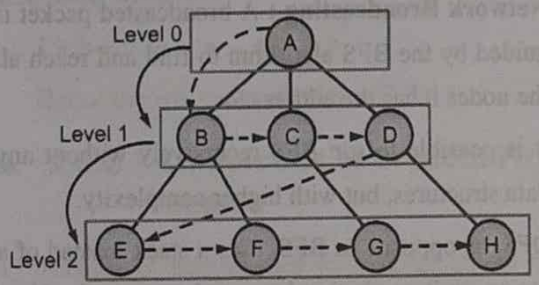


Fig. 2.8.2

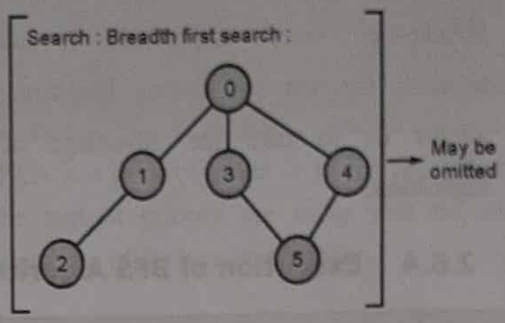
Algorithm

(1) Mark any node as starter or initial

(2) Explore and traverse un-visited nodes adjacent to starting node.

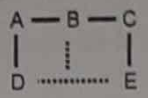
(3) Mark node as completed and move to next adjacent and un-visited nodes.

Search : Breadth first search



BFS will always find the shortest path in an unweighted graph.

Consider an unweighted graph like this :



And my goal is to get from A to E.

I begin at A as it my origin. I queue A, followed by immediately dequeueing A and exploring it. This yields B and D, because A is connected to B and D. I thus queue both B and D.

I cheque B and explore it, and find that it leads to A (already explored), and C, so I queue C. I then dequeue D, and find that it leads to E, and that is my goal.

I then deque C, and find that it also leads to E, my goal.

- BFS can only be used to find shortest path in a graph if
  1. There are no loops.
  2. All edges have same weight or no weight.
- To find the shortest path, all you have to do is start from the source and perform a breadth first source and stop when you find your destination node.
- Greedy best-first search algorithm always selects the path which appears best at that moment.

Unit II In Sem.

- It is the combination of depth-first search and breadth-first search algorithms. It uses the heuristic function and search. Best-first search allows us to take the advantage of both algorithms.

2.8.4 Execution of BFS Algorithm

- **Step 1 :** Start by putting any one of the graph's vertices at the back of the queue.

↓

- **Step 2 :** Take the front item of the queue and add it to the visited list.

↓

- **Step 3 :** Create a list of that vertex's adjacent nodes.

↓

- **Step 4 :** Keep continuing steps two and three till the querry is empty.

2.8.5 Advantages and Disadvantages of BFS

Advantages

(1) Solution will be definitely found out by BFS if there is some solution.

(2) BFS will never get trapped in blind valley, means unwanted nodes.

(3) If there are more than one solution then it will find solution with minimal steps.

Disadvantages

(1) Memory constraints as it stores all the nodes of present level to go for next level.

(2) If solution is far away then it consumes time.

2.8.6 Applications of BFS Algorithm

We mention some of the applications where a BFS algorithm implementation can be highly effective.

(1) **Unweighted graphs :** BFS algorithm can easily create the shortest path and a minimum spanning tree to visit all the vertices of the graph in the shortest time possible with high accuracy.

(2) **P2P Networks :** BFS can be implemented to locate all the nearest or neighbouring nodes in a peer to peer network. This will find the required data faster.

(3) **Web Crawlers :** Search engines or web crawlers can easily build multiple levels of indexes by employing BFS. BFS implementation starts from the source, which is the web page, and then it visits all the links from that source.

(4) **Navigation Systems :** BFS can help find all the neighbouring locations from the main or source location.

(5) **Network Broadcasting :** A broadcasted packet is guided by the BFS algorithm to find and reach all the nodes it has the address for.

- It is possible to run BFS recursively without any data structures, but with higher complexity.
- DFS, as opposite to BFS, uses a stack instead of a queue, so it can be implemented recursively. Note that the code used is iterative but it is trivial to make it recursive.
- In BFS, a queue data structure is used. One can mark any node in the graph as root and start traversing the data from it.
- BFS traverses all the nodes in the graph and keeps dropping them as completed.

- BFS visits an adjacent unvisited node, marks it as done, and inserts it into a queue.

2.8.7 Performance Measures of BFS

Time Complexity

Breadth-first search, being a brute search generates all the nodes for identifying the goal. The amount of time taken for generating these nodes is proportional to the depth d and branching factor b and is given by,

$$1+b+b^2+b^3+.....+b^d \simeq b^d$$

Hence the time-complexity = $O(b^d)$

Space Complexity

Unlike depth-first search wherein the search procedure has to remember only the paths it has generated, breadth-first search procedure has to remember every node it has generated. Since the procedure has to keep track of all the children it has generated, the space-complexity is also a function of the depth d and **branching factor b**. Thus space complexity becomes,

$$1+b+b^2+b^3+...+b^d \approx b^d$$

Hence the space-complexity = O (bd)

2.8.8 Limitations of Breadth First Search

(1) Amount of time needed to generate all the nodes is considerable because of the time-complexity.

(2) Memory constraint is also a major hurdle because of the space-complexity.

(3) The searching process remembers all unwanted nodes which is of no practical use for the search.

GQ. Which storage structure is preferably chosen for node representation in open list, while performing best-first search over a state space and why?

OPEN is a priority queue in which the elements with the highest priority are those with the most promising value of the heuristic function.

Unit II In Sem.

2.8.9 Example of Breadth First Search

GQ. Give an example of a problem for which breadth-first would work better than depth first search and vice-vesa.

In general, BFS is better for problems related to finding the **shortest paths** or somewhat related problems. Because here we can go from one node to all node that are adjacent to it and hence we effectively move from path length one to path length two and so on.

- While DFS on the other hand, helps more in connectivity problems and also in finding cycles in graph (cycles can be found in BFS with a bit of modification).
- Determining connectivity with DFS is trivial, if we call the explore procedure twice from the DFS procedure, then the graph is disconnected (this is for an undirected graph).
- We can see the strongly connected component algorithm for a directed graph here, which is a modification of DFS. Another application of the DFS is topological sorting.

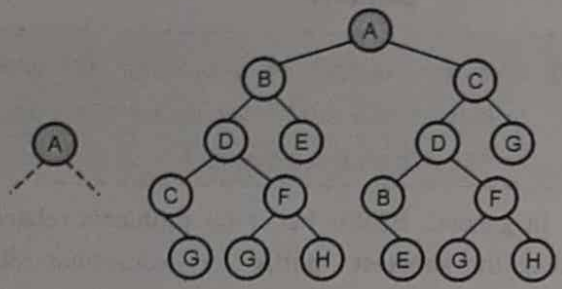
NOTES

2.8.10 Solved Example on BFS

Ex. 2.8.2 : Consider the given tree, apply breadth first search algorithm and also write the order in which 10 nodes are expanded.

Soln. :

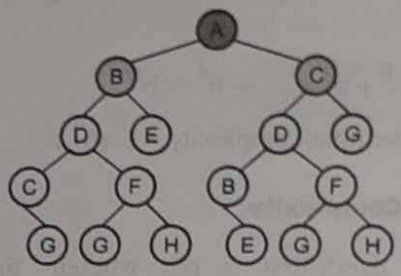
- **Step 1 :**
  Node, A can be regarded as source node.



(1B1)**Fig. Ex. 2.8.2**

FRINGE : A

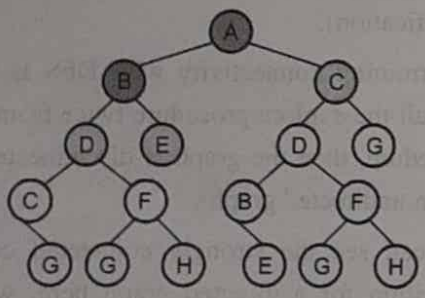
- **Step 2 :** We consider sub-nodes B and C of A. Source node A is removed.



(1B2)**Fig. Ex. 2.8.2(a)**

FRINGE : B C

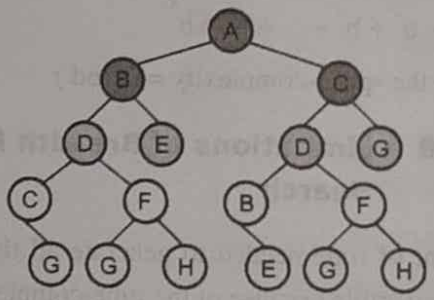
- **Step 3 :** We begin with left nodes. Consider left node B and its subnodes D and E. We remove node B..



(1B3)**Fig. Ex. 2.8.2(b)**

FRINGE : C D E

- **Step 4 :** We consider left subnode D. Its subnodes are C and F. We remove node D.



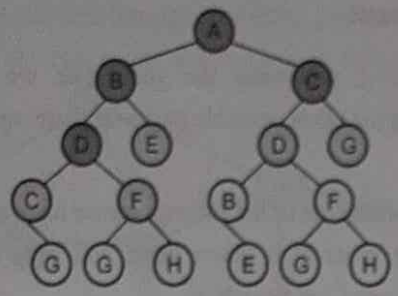
(1B4)**Fig. Ex. 2.8.2(c)**

FRINGE : D E D G

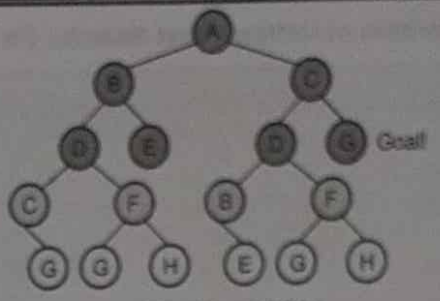
- **Step 5 :** We consider left subnode C and its subnodes are D and G. We remove subnode C from the fringe and add subnodes D and C to the fringe.

FRINGE : E D G C F

- **Step 6 :** Now, consider right subnode E of B and subnode E is removed from fringe.



(185)Fig. Ex. 2.8.2(d)



(186)Fig. Ex. 2.8.2(e)

- **Step 7 :** Now, subnode D is expanded.

∴ **Fringe : G C F B F**

- **Step 8 :** Now, we select G for expansion. It is found to be a goal node. And the algorithm terminates.

Unit II In Sem.

☞ **Performance Evaluation**

- **Time complexity :** $O(b^d)$
- **Space complexity :** O(bd)

  Where b - branching factor,

  and d - Depth of the shallowest goal node
- **Optimality :** can be reached

☞ **Advantage**

Path of minimal length to the goal can be found out.

☞ **Disadvantages**

(1) The search-space should be small in order to use Breadth First Search (BFS) algorithm.

(2) Generation and storage of a tree is required whose size is exponential (b^d), the depth of the shallowest goal node.

2.8.11 Application of BFS

(1) Finding shortest path.

(2) Checking graph with bipertiteness.

(3) Copying Cheney's algorithm.

2.9 UNIFORM COST SEARCH

- Uniform-cost-search is an uninformed search algorithm that uses the lowest cumulative cost to find a path from the source to the destination.
- Nodes are expanded, starting from the root, according to the minimum cumulative cost. The uniform-cost search is then implemented using a Priority Queue.
- Here, instead of inserting all vertices into a priority queue, we insert only source, then one by one we insert nodes when needed.

2.9.1 Algorithm of U.C.S.

- Uniform Cost Search is an algorithm used to move around a directed weighted search space to go from a start node to one of the ending nodes with a minimum cumulative cost.
- This search is an uninformed search algorithm, i.e. it does not take the state of the node or search space into consideration.
- It is used to find the path with the lowest cumulative cost in a weighted graph where nodes are expanded according to their cost of traversal from the root node. This is implemented using a priority queue where lower the cost higher is its priority.

Algorithm of Uniform Cost Search : (in AI)

- **Step 1 :** Insert Root Node into the queue.
- **Step 2 :** Repeat till queue is not empty.
- **Step 3 :** Remove the next element with the highest priority, from the queue.
- **Step 4 :** If the node is a destination node, then print the cost and the path and exit, else insert all the children of removed elements into the queue with their cumulative cost as their priorities.

Here root Node is the starting node for the path, and a priority queue is being maintained to maintain the path with the least cost to be chosen for the next traversal.

2.9.2 Execution of Algorithm of Uniform Cost Search

- Uniform-cost search is similar to Dijkstra's algorithm.

In this algorithm,

- **Step 1 :** from the starting state we will visit the adjacent states and will choose the least costly state.
- **Step 2 :** then we choose the next costly state from the all un-visited and adjacent states of the visited states,
- **Step 3 :** in this way we try to reach the goal state.

Remark

Even if we reach the goal state we continue searching for other possible paths (if there are multiple goals).

- The elements in the priority queue have almost the same costs at a given time, and thus the name Uniform Cost Search.
- It may appear that elements do not have almost the same costs, but when applied on a much larger graph it is certainly so.
- Uniform costing refers to acceptance of identical costing principles and procedures by all or many units in the same industry by mutual agreement.
- 'Uniform Cost Search (UCS)' algorithm is mainly used when the step costs are not the same but we need the optimal solution to the goal state.
- In such cases, we use Uniform Cost Search, to find the goal and the path including the cumulative cost to expand each node from the root node to the goal node.
- Uniform cost-search is optimal. This is because, at every step the path with the least cost is chosen, and paths never get shorter as nodes are added, ensuring that the search expands nodes in the order of their optimal path cost.
- To measure the time complexity, we need the help of path cost instead of depth d.

2.9.3 Example of Uniform Cost Search (Linear Displacement)

Consider the example if Fig. 2.9.1; where we need to reach any one of the destination node $\{G_1, G_2, G_3\}$ starting from node S.

Node { A, B, C, D, E and F } are the intermediate nodes. Our motive is to find the path from S to any of the destination state with the least cumulative cost. Each directed edge represents the direction of

movement allowed through that path, and its labelling represents the cost is one travels through that path.

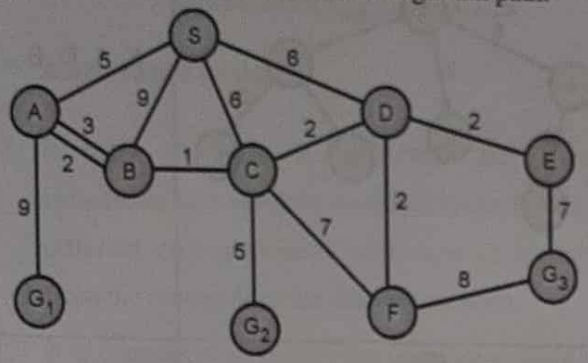


Fig. 2.9.1

Thus overall cost of the path is a sum of all the paths.

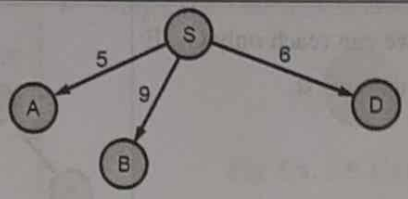
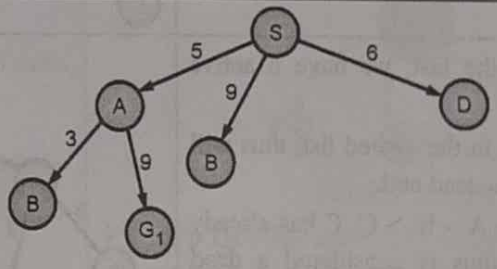
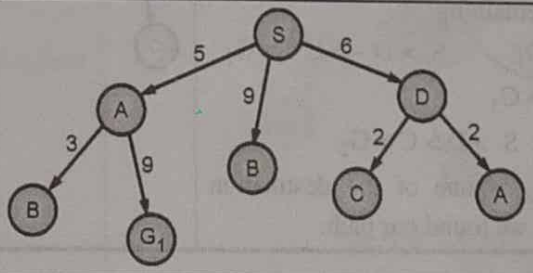
For e.g. : A path from S to G_1 –

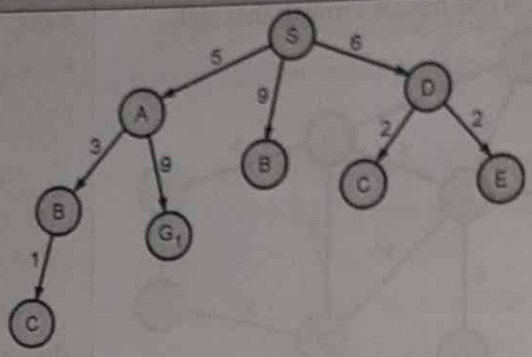
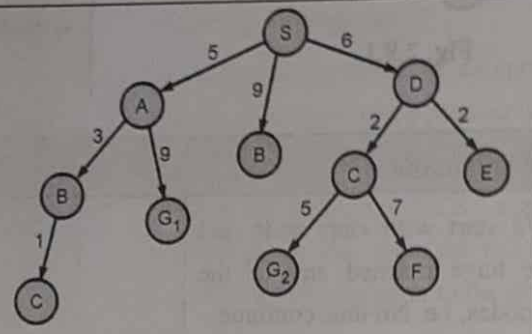
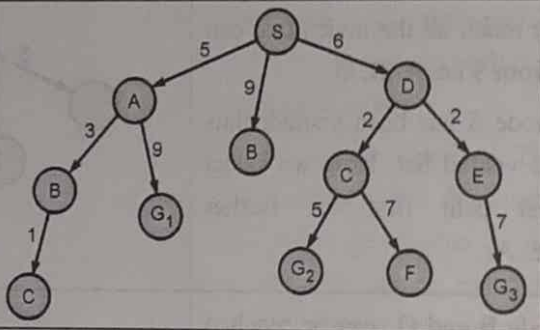
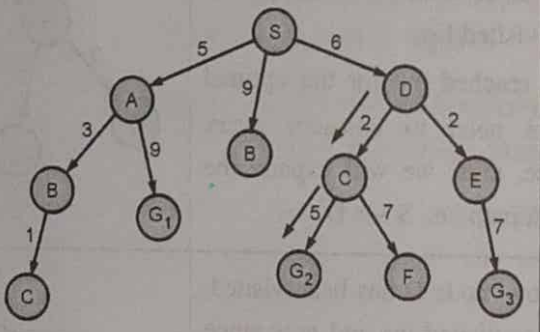
{S > A > G_1 } whose cost is SA + AG_1 = 5 + 9 = 14.

Here we maintain a priority queue the same as BFS with the cost of the path as its priority, lower the cost higher is the priority.

We use a tree to show all the paths possible and also maintain a visited list to keep track of all the visited nodes as we need not visit any node twice.

Unit II In Sem.

Explanation	Flow	Visited List
▶ **Step 1 :** We start with start node and check if we have reached any of the destination, nodes, i.e. No thus continue.		
▶ **Step 2 :** We reach all the nodes that can be reached from S i.e. A, B, D. And since node S has been visited, thus added to the visited list. Now we select the cheapest path first for further expansion i.e. A.		S
▶ **Step 3 :** Node B and G_1 can be reached from A and since node A is visited thus move to the visited list. Since G_1 is reached but for the optimal solution, we need to consider every possible case, thus, we will expand the next cheapest path i.e. S –> D		S, A
▶ **Step 4 :** Now, node D has been visited, thus it goes to visited list and now since we have three paths with the same cost, we choose alphabetically thus will expand node B.		S, A, D

Explanation	Flow	Visited List
▶ **Step 5 :** From B, we can only reach node C. Now the path with minimum weight is S –> D –> C i.e. 8 Thus expand C, and B has now visited D node.		S, A, D, B
▶ **Step 6 :** From C we can reach G_2 and F node with 5 and 7 weights respectively. Since S is present in the visited list, thus we are not considering the C –> S path. Now, C will enter the visited list. Now the next node with the minimum total path is S > D > E , i.e. 8 Thus we will expand E.		S, A, D, B, C
▶ **Step 7 :** From E we can reach only G_3. E will move to the visited first.		S, A, D, B, C, E
▶ **Step 8 :** In the last, we have 6 active paths. S –> B B is in the visited list, thus will be marked as a dead end. Same for S > A > B > C C has already been visited thus is considered a dead end. Out of the remaining S > A –> G_1 S > D –> C –> G_2 S > D > E > G_3 Minimum is S > D > C > G_2 and also G_2 is one of the destination nodes. Thus we found our path.		S, A, D, B, C, E

2.9.4 Advantages and Disadvantages of U.C.S

Advantages of U.C.S

(1) It helps to find the path with the lowest cumulative cost inside a weighted graph having a different cost associated with each of its edge from the root node to the destination node.

(2) It is considered to be an optimal solution since, at each state, the least path is considered to be followed.

Disadvantages of U.C.S

(1) The open list is required to be kept sorted as priorities in priority queue needs to be maintained.

(2) The storage required is exponentially large.

(3) The algorithm may be stuck in an infinite loop as it considers every possible path going from the root node to the destination node.

2.9.5 Performance Measures

Time and Space Complexity

- Uniform cost search is complete, when UCS finds the solution, (if there is a solution).
- Let C* be the cost of optimal solution, and ϵ be each step closer to the goal node. Then the number of steps is,

$$= \frac{C^*}{(\epsilon + 1)}$$

Here, we have taken + 1, as we start from state 0 and end to $\frac{C^*}{\epsilon}$.

2.9.6 Solved Example on Curved Displacement

Ex. 2.9.1 : Apply uniform cost search algorithm on given graph.

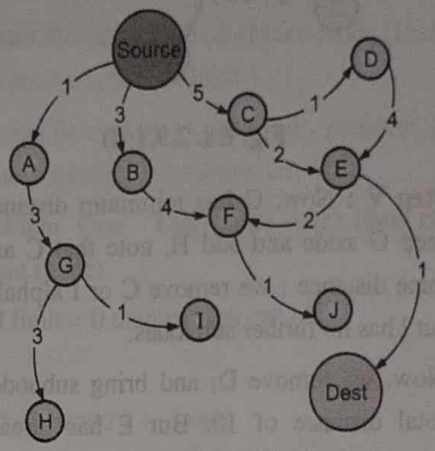


Fig. Ex. 2.9.1 : Graph

Soln. :

Step I : We mention source-node

Fig. Ex. 2.9.1(a) : Node

Step II : We add the nodes A, B, C to the source node, :

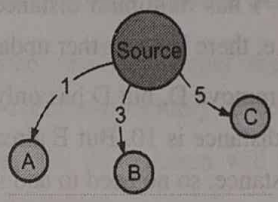


Fig. Ex. 2.9.1(b)

Step III : The node A has minimum distance 1, so we keep it aside and add the node G.

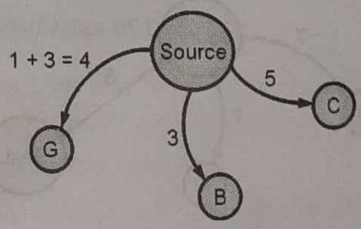


Fig. Ex. 2.9.1(c)

- **Step IV :** For nodes, B and C, B has minimum distance, so we add node F to B.

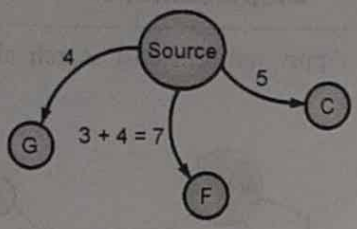


Fig. Ex. 2.9.1(d)

- **Step V :** Now, G has minimum distance, so we keep G aside and add H, note that C and I have same distance ; we remove C or I alphabetically ; but I has no further subnodes.

  Now, we remove D, and bring subnode E, with total distance of 10. But E has already lesser distance 7, so we add E with distance 7.

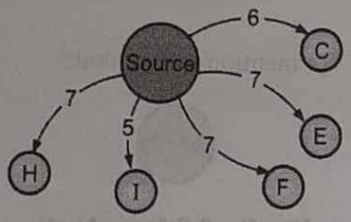


Fig. Ex. 2.9.1(e)

- **Step VI :** I has minimum distance, but I has no subnode, i.e. there is no further updating.

  Node, we remove D, but D has only one sub-node E, but its distance is 10. But E already exists with a lesser distance, so no need to add it further.

- **Step VII :** (Alphabetically) the next minimum distance is that of E, so we remove E.

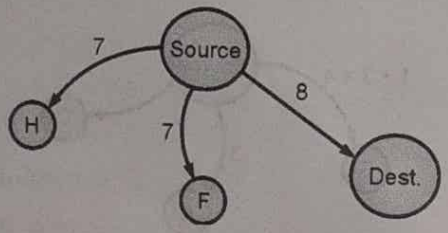


Fig. Ex. 2.9.1(f)

- **Step VIII :** Now, minimum cost is F, so it is removed (alphabetically) and subnode J is added.

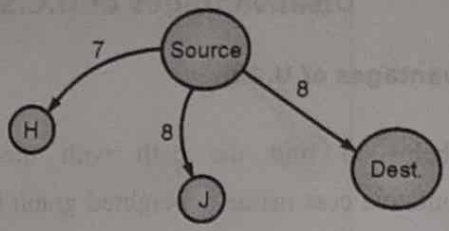


Fig. Ex. 2.9.1(g)

- **Step IX :** Now, H has minimum cost, so it is removed and H also H has no further subnodes.

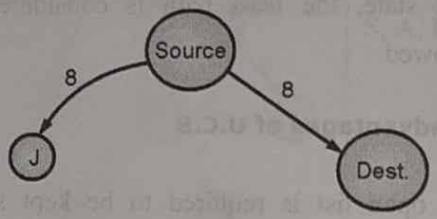


Fig. Ex. 2.9.1(h)

Here the algorithm ends.

Thus, the minimum distance between the source and destination node is **8**.

☞ **Time complexity**

The time complexity needed to run uniform cost search is : $O(b(1 + C / \varepsilon))$

Where : b - branching factor, C - optimal cost, ε - cost of each step

☞ **Optimal**

Uniform-cost search is always optimal as it only selects a path with the lowest path cost.

2.10 DEPTH LIMITED SEARCH

(I) Explanation

- In infinite state spaces, depth-first search method fails. This failure can be alleviated by supplying depth-first search with a pre -determined depth limit *l*. It means that, nodes at depth *l* are treated as if they have no successors.

This approach is called **depth-limited search.**

The depth limit can solve infinite path problem.

- Its time-complexity is O (b^l) and its space-complexity is O (bl)
- **Thus depth-first search can be viewed as a special case of depth-limited search with $l = \infty$**
- In some cases, depth limit can be based on knowledge of the problem. For example, on the map of surat there are **25 cities.**
- So, if there is a solution, it must be of length 24 at the longest, hence $l = 24$ is a possible choice. But if we look at the map carefully, we observe that any city can be reached from any other city in at most 15 steps.

  This number, known as the **diameter** of the **state space**, gives us a better depth limit. And this leads to a more efficient depth-limited search.
- Observe that **depth-limited search** can terminate with two kinds of failure :
  (i) the standard **failure value** which indicates no solution;
  (ii) the **cut-off value** which indicates no solution within the **depth-limit**

☞ Drawbacks of Depth-Limited Search

(i) The **depth limited search** is also incomplete if we choose $l < d$; that is, the shallowest goal is beyond the depth limit.

(ii) Depth-limit search will also be non-optimal if we choose $l > d$.

☞ Implementation

Depth – limited search can be implemented as a simple modification to the general tree-or graph search algorithm. Or, it can be implemented as a **simple recursive algorithm.**

(II) Algorithm

A recursive implementation of Depth-limited Search Algorithm

Function : Depth-Limited-search (Problem, limit) **returns** a solution, or failure/cut off.

Return : Recursive - DLS (Make-Node (Problem. Initial-state), problem, limit)

Function Recursive - DLS (node, problem, limit) **returns** a solution, or failure/ cut-off

If problem. Goal -Test (node-state) **then return** solution (node)

else if limit = 0 then **return cut off**

else

cut off-occurred ? ← false

for each action **in** problem. Action(node, state) **do**

child ← child – Node (problem, node, action)

result ←Recursive – DLS(child, problem, limit- 1)

If result = cutoff **then** cutoff-occurred? ← true

else if result ≠ failure **then return** result

If cutoff-occurred ? **then return** cutoff

else return failure

☞ Advantages of Depth Limited Search

(1) Depth limited search is better than DFS as it requires less time and memory space.

(2) DFS assures that the solution will be found if it exists in infinite time.

(3) DLS has applications in graph theory particularly similar to DFS.

☞ Disadvantages of DLS

(i) The goal node may not exist in the depth limit set earlier, which will push the user to iterate further adding execution time.

(ii) The goal node cannot be found out if it does not exist in the desired limit.

Unit II In Sem.

☞ **Optimality**

The DLS is a non-optimal algorithm since the depth that is chosen can be greater than d ($l > d$). Thus DLS is not optimal if $l > d$.

☞ **Time complexity**

It is similar to DFS, i.e. O (b^l), where 1 is the specified depth limit.

☞ **Space complexity**

It is similar to DFS, it is O (b^l), where 1 is the specified depth limit.

☞ **Conclusion – DLS**

(i) DLS is not the case for uninformed search strategy.

(ii) DLS algorithm is used when we know the search domain, and there exists a prior knowledge of the problem and its domain.

(iii) There is little idea of the goal nodes depth.

(iv) The problem with depth-limited search is to set the value of l optimally, so as not to leave out any solution.

Also keep the time and space complexity to a minimum.

2.11 ITERATIVE DEEPENING SEARCH TECHNIQUE (IDS OR IDDFS)

UQ. Explain Iterative Deepening search algorithms based on performance measure with justification; complete, optimal, Time and Space complexity.

(Q. 4(a), Dec. 18, 10 Marks)

(1) Iterative deepening search or more specifically iterative deepening depth-first search (IDS or IDDFS) is a **state space/graph search strategy** in which a **depth-limited version** of depth-first search is run repeatedly with increasing depth limits until the goal is found.

(2) IDDFS is equivalent to breadth-first search, but uses much less memory; on each iteration, it visits the nodes in the search tree in the same order as depth-first search, but the cumulative order in which nodes are first visited is effectively breadth-first.

(3) DDFS combines depth-first search's space-efficiency and breadth-first search's completeness (When the branching factor is finite). It is optimal when the **path cost is a non-decreasing function of the depth of the node**.

(4) The **time complexity of IDDFS is O (b^d)** and its **space complexity is O(b^d)**, where **b** is the branching factor and **d** is the depth of the shallowest goal.

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

2.11.1 IDDFS Algorithm

Function iterative-Deepening-search (problem) **returns** a solution or failure

inputs : problem, a problem

for depth ← 0 **to** ∞ **do**

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

if result ≠ cutoff **then return** result

The iterative deepening search algorithm, which repeatedly applies depth-limited search with increasing limits.

It terminates when a solution is found or if the depth- limit search returns **failure** meaning that no solution exists.

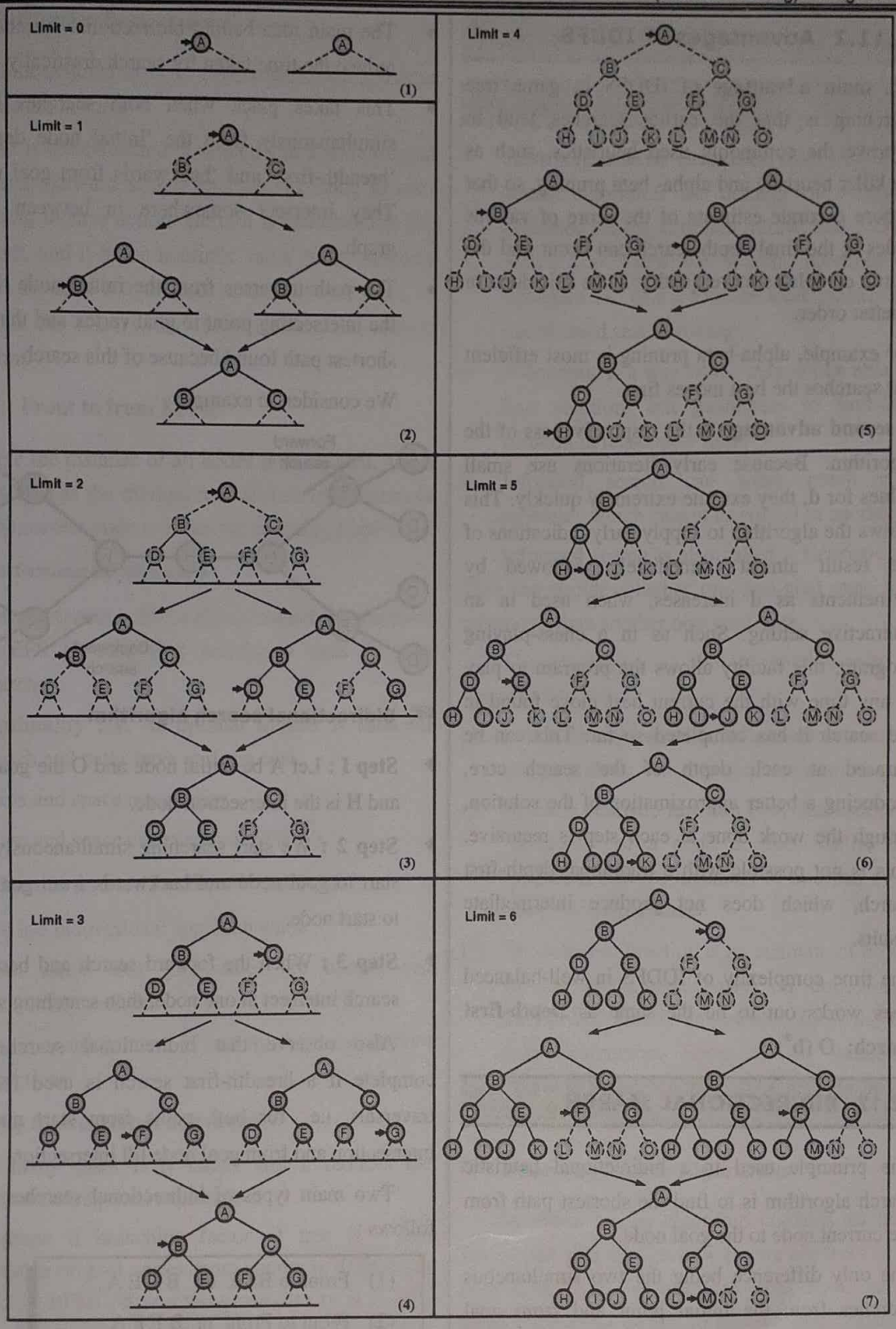


Fig. 2.11.1

2.11.2 Advantages of IDDFS

(1) The **main advantage** of IDDFS in game tree searching is that the earlier searches tend to improve the commonly used heuristics, such as the killer heuristic and alpha- beta pruning, so that a more accurate estimate of the score of various nodes at the final depth search can occur and the search completes more quickly since it is done in a better order.

For example, alpha-beta pruning is most efficient if it searches the best moves first.

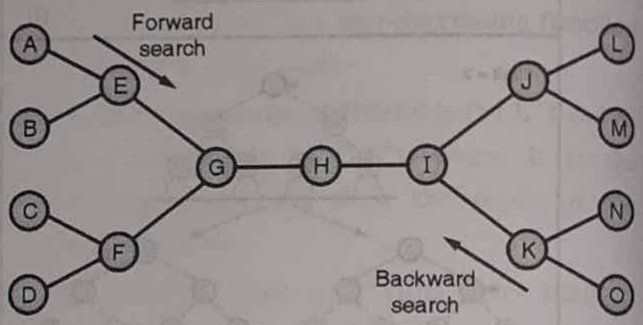
(2) A **second advantage** is the responsiveness of the algorithm. Because early iterations use small values for **d**, they execute extremely quickly. This allows the algorithm to supply early indications of the result almost immediately, followed by refinements as d increases, when used in an interactive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

(3) The time complexity of IDDFS in well-balanced trees works out to be the same as **Depth-first search: O (b^d)**

2.12 BIDIRECTIONAL SEARCH

- The principle used in a bidirectional heuristic search algorithm is to find the shortest path from the current node to the goal node.
- The only difference being the two simultaneous searches from the initial point and from goal vertex.
- The main idea behind bidirectional searches is to reduce the time taken for search drastically.
- This takes place when both searches happen simultaneously from the 'initial node depth' or 'breadth-first' and 'backwards from goal nodes'. They intersect somewhere in between of the graph.
- The path traverses from the initial node through the intersecting point to goal vertex and that is the shortest path found because of this search.

We consider an example :



Bidirectional search algorithm

- **Step I :** Let A be initial node and O the goal node and H is the intersection node.
- **Step 2 :** We start searching simultaneously from start to goal node and backwards from goal node to start node.
- **Step 3 :** When the forward search and backward search intersect at one node, then searching stops.

Also observe that bidirectional searches are complete if a breadth-first search is used for both traversals, i.e., for both paths from start node till intersection and from goal node till intersection.

Two main types of bidirectional searches are as follows :

(1) Front to Back or B F E A
(2) Front to Front or B F F A

(1) Front to Back or B F E A

In bidirectional front, to front search, two heuristic functions are needed.

First is the estimated distance from a node to goal state using forward search and second, node to start state using reverse action. Here, h is calculated in the algorithm, and it is the heuristic value of the distance between the node n to the root of the opposite tree s or t. This is the most widely used bidirectional search algorithm.

(2) Front to front BFFA

Here the distance of all nodes is calculated, and h is calculated as the minimum of all heuristic distances from the current node to nodes on opposing fronts.

☞ Performance measure

(1) **Completeness :** Bidirectional search is complete if BFS (Breadth-first search) is used in both searches.

(2) **Optimality :** It is optimal if BFS is used for search and paths have uniform cost.

(3) Time and space complexity

Time and space complexity is O ($b^{d/2}$).

☞ When to use bidirectional approach

We use bidirectional approach when :

(1) Both initial and goal states are unique and completely defined.

(2) The branching factor is exactly the same in both directions.

☞ Why bidirectional approach ?

(i) In many cases it is faster, and it reduces the amount of required exploration.

(ii) Suppose if branching factor of tree is b and distance of goal vertex from source is d, then the normal BFS/DFS searching complexity is O (b^d). But for two search complexity is O ($b^{d/2}$) which is far less than O (b^d).

2.13 INFORMED SEARCH

GQ. What is informed search ?

- **Informed search (heuristic search) :** Informed search algorithms use domain knowledge in an informed search, problem information is available which can guide the search. Informed search strategies can find a solution more quickly than an uninformed search strategy.
- A heuristic is a way which may not be able to give best solutions but guarantees to find a good solution in reasonable time.
- Informed search can solve much complex problems which cannot be solved in another way.

Informed search is also called a Heuristic search. This can decide whether one non-goal state is more promising than another non-goal state.

The advantages of the informed search comes from the fact that :

(1) It adds **domain-specific information** to select the best path along which to continue searching.

(2) Define a heuristic function **h(n)** that estimates the "goodness" of a node n. Specifically, **h(n) = estimated cost** (or distance) of minimal cost path from n **to a goal state.**

(3) The heuristic function is an estimate of how close we are to a goal, based on domain-specific information that is computable from the current state description. Some of the examples of informed search are best first search, beam search, **A*** and **AO*** algorithms etc.

☞ Informed Search Algorithms

Informed search algorithm contains an array of knowledge that tells us how far we are from the goal, path cost, how to reach to goal node etc. This knowledge helps agents to explore less to the search space and find the goal node more efficiently.

Unit II In Sem.

The informed search algorithm is more useful for large search space. Informed search uses the idea of heuristic, hence it is also called as Heuristic search.

2.13.1 Example for Informed Search

GQ. Give an example for informed search.

Beam search is an example for informed search

Beam search : This is an attractive heuristic search technique because it permits searching to be done on a **multi-processor machine**, thereby reducing computations. Reduction in computations is achieved by pursuing some paths and selecting only selected paths.

The searching process is similar to breadth-first search wherein searching proceeds level by level. At each level, heuristic functions are applied to reduce the number of paths to be explored. In fact, it is done to keep the width of the beam to be minimal. The width of the beam is fixed and whatever be the depth of the tree, the number of alternatives to be scanned is the product of the width and the depth.

2.13.2 Algorithm for Beam Search

- **Step 1 :** Let width_of_beam = w.
- **Step 2 :** Put the initial node on a list START.
- **Step 3 :** If (START is empty) or (START = GOAL), then terminate search.
- **Step 4 :** Remove the first node from START, Call this as node a.
- **Step 5 :** If (a = GOAL), then terminate search with success.
- **Step 6 :** Else if node a has successors, generate all of them and add them at the tail of START.
- **Step 7 :** Use a heuristic function to rank and sort all the elements of START.
- **Step 8 :** Determine the nodes to be expanded. The number of nodes should not be greater than w. Name these as START1.
- **Step 9 :** Replace START with START1.
- **Step 10 :** Goto Step 2.

Fig. 2.13.1 shows how beam search proceeds. This search has been used in an expert system called **ISIS for factory scheduling**.

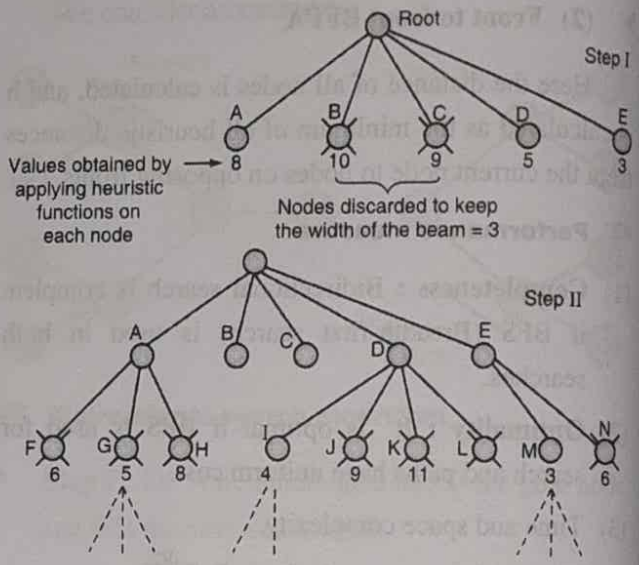


Fig. 2.13.1 : Beam search procedure

2.14 HEURISTIC FUNCTION

UQ. What is heuristic function ?
Q. 3(b), Dec. 18, 10 Marks, Q. 1(c), Dec. 15, May 18, 3 Marks, Q. 1(c), May 17, Q. 3(b), Dec. 16, 5 Marks

The process of searching can be drastically reduced by the use of heuristics. Heuristics are approximations used to minimize the searching process.

- It is a function that maps from problem state description to measures of desirability, usually represented as number.

- Which aspect of the problem state are considered, how these aspects are evaluated, and weight given to the individual aspects are chosen. Define a heuristic function h(n) that estimates the "goodness" of a node n.

  Specifically, **h(n) = estimated cost (or distance) of minimal cost path from n to a goal state.**

- The heuristic function is an estimate of how close we are to a goal, based on domain-specific information that is computable from the current state description.
- Computed in such a way that the value of the heuristic function at a given node in the search give as good estimate as possible of whether that node is on the desired path to a solution.
- Well-designed heuristic function can play an important role in efficiently guiding a search process toward a solution.
- Sometimes very simple heuristic function can provide a fairly good estimate of whether a path is any good or not. In other situation more complex function should be employed.

☞ **Generally, two categories of problems use heuristics**

(1) Problems for which no exact algorithms are known and one needs to find an approximate and satisfying solution. E.g., computer vision, speech recognition etc.

(2) Problems for which exact solutions are known, but computationally infeasible. E.g., Rubik's cube, chess etc. The heuristics which are needed for solving problems are generally represented as a heuristic function which maps the problem states into number. These numbers are then appropriately used to guide search.

2.14.1 Simple Heuristic Functions

(1) In the famous 8-tile puzzle, the hamming distance is a popular heuristic function. It is an indicator of the number of tiles in the position they are to be.

(2) In a game like chess, the material advantage one has over the opponent is an indicator. Normally, the following values are assigned to the pieces. Queen-9, etc.

Unit II In Sem.

The following algorithms make use of heuristic evaluation functions

1. Hill climbing
2. Constraint satisfaction
3. Best-first search
4. A* algorithm
5. AO* algorithm
6. Beam search

- The purpose of heuristic function is to guide the search process in the most profitable direction by suggesting which path to follow first when more than one is available.
- The more accurate the heuristic function estimate the true merit of each node in the search tree, the more direct the solution process.
- In the extreme, the heuristic would be so good that essentially no search would be required system would move directly to a solution.
- But for many problems, the cost of computing the value of such a function would outweigh the effort saved in the search process.
- In general, there is a trade-off between the cost of evaluating a heuristic function and the saving of search time that the functions provide.

2.14.2 Problem Characteristics for Heuristic Search

GQ. What are the various problem characteristics for heuristic search? **Or** Explain the problem characteristic briefly with appropriate example.

Heuristic search is a very general method applicable to a large class of problems. It encompasses a variety of specific techniques, each of which is particularly effective for a small class of problems.

Several key dimensions for Heuristic Search

(1) Is the problem decomposable into a set of (nearly) independent smaller or easier sub problems?

(2) Can solution steps be ignored or at least undone if they prove unwise?

(3) Is the problem's universe predictable?

(4) Is a good solution to the problem obvious without comparison to all other possible solutions?

(5) Is the desired solution a state of the world or a path to a state?

(6) Is a large amount of knowledge absolutely required to solve the problem, or is knowledge important only to constrain the search?

(7) Can a computer that is simply given the problem return the solution, or will the solution of the problem require interaction between the computer and a person?

2.14.3 Is the Problem Decomposable ?

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

Ex : $\int (x^2 + 3x + \sin^2 x \cos 2x)\, dx$

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

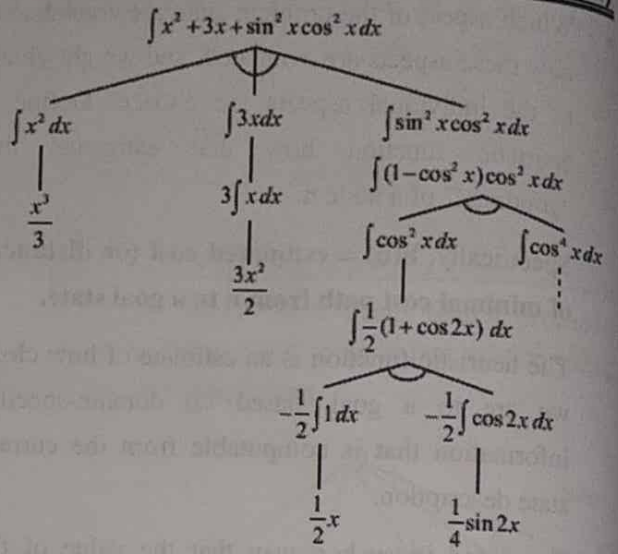


Fig. 2.14.1

2.14.4 Can Solution Steps be Ignored or Undone ?

- Problem fall under three classes ignorable, recoverable and irrecoverable. This classification is with reference to the steps of the solution to a problem.
- For example, consider proving the theorem. We may later find that it is of no help. We can still proceed further, since nothing is lost by this redundant step. This is an example of ignorable solutions steps.
- Now consider the 8 puzzle problem tray and arranged in specified order. While moving from the start state towards goal state, we may make some stupid move and consider theorem proving.
- We may proceed by first proving lemma. But we may backtrack and undo the unwanted move. This only involves additional steps and the solution steps are recoverable.
- Lastly consider the game of chess. If a wrong move is made, it can neither be ignored nor be recovered.
- The thing to do is to make the best use of current situation and proceed. This is an example of an irrecoverable solution steps.

(i) Ignorable problems

Ex : Theorem proving

In which solution steps can be ignored.

(ii) Recoverable problems Ex : 8 puzzle

In which solution steps can be undone.

(iii) Irrecoverable problems Ex : Chess

In which solution steps can't be undone.

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

2.14.5 Is the Universal Predictable ?

- Problems can be classified into those with certain outcome (eight puzzle and water jug problems) and those with uncertain outcome (playing cards) in certain – outcome problems, planning could be done to generate a sequence of operators that guarantees to a lead to a solution.
- Planning helps to avoid unwanted solution steps. For uncertain outcome problems, planning can at best generate a sequence of operators that has a good probability of leading to a solution.
- The uncertain outcome problems do not guarantee a solution and it is often very expensive since the number of solution and it is often very expensive since the number of solution paths to be explored increases exponentially with the number of points at which the outcome cannot be predicted. Thus one of the hardest types of problems to solve is the irrecoverable, uncertain-outcome problems. **Ex :** Playing cards.

2.14.6 Is Good Solution Absolute or Relative ? (Is the Solution a State or a Path?)

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

Unit II In Sem.

2.14.7 The Knowledge Base Consistent ?

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

Remark

- A Boolean expression is a logical statement that is either True or False.
- Boolean expressions can compare data of any type as long as both parts of the expression have the same basic data type.
- One can test data to see, if it is equal to, greater than, or less than other data.

Boolean data is as follows :

Boolean values :

(Yes and no, and their synonyms, on and off, and true and false).

2.14.8 What is the Role of Knowledge ?

- Though one could have unlimited computing power, the size of the knowledge base available for solving the problem does matter in arriving at a good solution.
- Take for example, the game of playing chess, just the rules for determining legal moves and some simple control mechanism is sufficient to arrive at a solution.
- But, additional knowledge about good strategy and tactics could help to constrain the search and speed up the execution of the program. The solution would then be realistic.
- Consider the case of predicting the political trend. This would require an enormous amount of knowledge even to be able to recognize a solution, leave alone the best. Ex : Playing chess, newspaper understanding.

2.14.9 Does the Task requires Interaction with the Person

The problems can again be categorized under two heads as follows :

(i) Solitary Problems
(ii) Conversational Problems

(i) Solitary Problems

In which the computer will be given a problem description and will produce an answer, with **no intermediate communication** and with the demand for an explanation of the reasoning process. Simple theorem proving falls under this category. Given the basic rules and laws, the theorem could be proved, if one exists. **Ex:** Theorem proving (give basic rules and laws to computer)

(ii) Conversational Problems

In which there will be **intermediate communication between a person and the computer**, whether to provide additional assistance to the computer or to provide additional informed information to the user, or both problems such as medical diagnosis fall under this category, where people will be unwilling to accept the verdict of the program, if they cannot follow its reasoning. Ex : Problems such as medical diagnosis.

2.14.10 Problem Classification

Actual problems are examined from the point of view; the tasks here is examining an input and decide which of a set of known classes. **Ex :** Problems such as medical diagnosis, engineering design.

2.14.11 Heuristic Function for : Travelling Salesman Problem

GQ. Explain with suitable example, heuristic function for : travelling salesman problem

UQ Give the initial state, goal test, successor function, and cost function for the travelling salesperson problem (TSP). There is a map involving N cities some of which are connected by roads. The aim is to find the shortest tour that starts from a city, visits all the cities exactly once and comes back to the starting city. **Q. 5(A), Dec. 16, 6 Marks**

UQ. Define heuristic function. Give an example heuristics functions for 8-puzzle problem. Find the heuristics value for a particular state of the Blocks World Problem. **Q. 1(b), May 17, 5 Marks**

Travelling salesman problem : A salesman has a list of cities, each of which he must visit exactly once. There are direct roads between each pair of cities on the list. Find the route the salesman should follow for the shortest possible round trip that both starts and finishes at any one of the cities.

☞ **Why informed (Heuristic) search method?**

Informed search method are more problem specific and acquires related knowledge and hence finds solution more efficiently than uniformed method.

Exploration : Exploration is one of the technique used for information gathering.

There are a few possible issues with this algorithm. One of the most important issues is completeness. It can be a typical extension to generate and test using the knowledge that directs the search. This knowledge is based on heuristic function that detects the closeness of current state to the goal state.

Example : Let us assume that we are solving travelling salesman problem with hill climbing.

(1) Begin with the initial state (that is a city salesman has visited).

(2) Move in the direction so that the city is not repeated, but additional city is visited (better state).

(3) Heuristic can be the number of cities visited.

(4) Keep moving in the state that improves the number of cities with legal action.

(5) When there is no way to improve heuristic function, stop.

Hill climbing algorithm makes incremental changes in the optimal direction based on heuristic.

There are many variants of hill climbing algorithms :

- To apply the steepest ascent hill climbing to the travelling salesman problem, we need to consider all successor states (possible moves) to choose the best one. This algorithm has higher complexity than the basic hill climbing, since it requires selecting move from a number of possible moves.
- Both basic and steepest ascent hill climbings may not able to reach the goal state, and hence, may fail to find the solution. Generally, it terminates in finding a state that does not have a better state in the vicinity or from where it is not possible to get to a better state in a single move.

Unit II In Sem.

2.14.12 Branch And Bound Technique

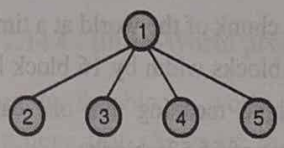
GQ. What do you mean by branch and bound technique ? Give some examples.

Branch and bound is a state space search method in which all the children of a node are generated before expanding any of its children.

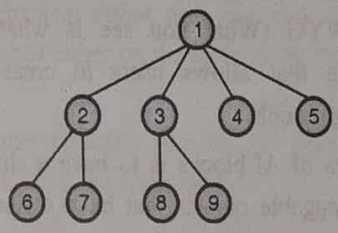
(1) Live-node : A node that has not been expanded.

(2) It is similar to backtracking technique but uses BFS-like search.

(3) Dead-node : A node that has been expanded.

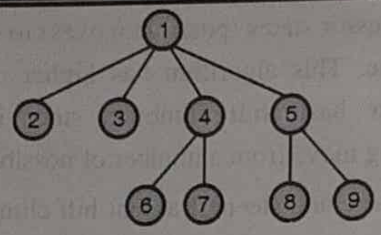


Live node : 2, 14.2 and 5



FIFO Branch and Bound Children of E-node are inserted in a queue

Fig. 2.14.2



LIFO Branch and Bound Children of E-node are inserted in a stack
Fig. 2.14.3

2.14.13 Block World Problem

UQ. Define heuristic function. Give an example heuristics function for Blocks world problem.
Q. 1(a), Dec. 15, 5 Marks

UQ. Find the heuristics value for a particular state of the Blocks World Problem.
Q. 1(b), May 17, 5 Marks

- The Block World is a 'planning domain' in artificial intelligence. The algorithm is similar to a set of wooden blocks of various shapes and colours but of same dimensions, sitting on table.
- The goal is to build one or more vertical stacks of blocks.
- The blocks in a world are infinite. Though one can see only a chunk of the world at a time. The chunk area is 16 blocks width by 16 block length by 256 blocks height, meaning that one can only see a total volume of 65,536 blocks.
- Artificial intelligence blocks is an intuitive WYSIWYG (What you see is what you get) interface that allows users to create machine learning models.
- The idea of AI-blocks is to have a simple scene with draggable objects that have scripts attached to them.
- The Blocks World is one of the most famous planning domains in Artificial Intelligence.
- A solution to the blocks world problem is a sequence of actions that allows us to start from an initial state and end up to a goal state. Therefore the optimal solution is the solution with the minimum cost (or else minimum actions required to get from the initial state to the goal state).

2.14.14 Actions

- The first member of the tuple represents the block that is about to move and second member of the tuple represents the block (or table) that will have the block (first member of the tuple) stacked on top of it.
- For example : If possible action = [(1, 2), (3, 0)] that means we have two actions :
  (i) The first action will move block 1 on top of block 2, and
  (ii) Second action will move block 3 on the table. (One cannot move a block that has other blocks stacked on top of it).
- After that we have list of all valid actions (list of tuples) and we are ready to move on and apply those actions to a state.
- From an algorithm perspective, blocks-world is an np-hard search and planning problem.
- The task is to bring the system from an initial state into a goal-state.

  [**Remark :** 'NP-complete' is non-deterministic polynomial-time complete. In computational complexity theory, a problem for which the correctness of each solution can be verified quickly and a brute-force search algorithm can actually find a solution by trying all possible solutions].

2.14.15 Motivation

- The simplicity of this toy-world (block world) / ends itself to classical symbolic AI approaches, in which the world is modelled as a set of abstract symbols, which may be reasoned about).

- AI can be researched in theory and with practical applications.
- To develop program on AI system, we invent an easy to solve domain which is called a 'toy-problem'.
- Toy problems were invented with the aim to program an AI which can solve it.
- The blocks-world domain is an example of a toy-problem. Its major advantage over more realistic AI applications is that, many algorithms and software programs are available which can handle the situation. This allows to compare different theories against each-other.
- More complicated derivations of the problem consist of cubes in different sizes, shapes and colours. From an algorithm perspective, blocks world is an np-hard search and planning problem.
- The task is to bring the system from an initial state into a goal state.

2.14.16 Symbolic Representation

(A) Four actions

(1) Unstack (A, B) : pick up **clear** block A from block B.

(2) Stack (A, B) : place block A using the arm onto **clear** block B.

(3) pickup (A) : lift clear block A with the empty arm.

(4) putdown (A) : place the held block A onto a free space on the table.

(B) Five predicates

(i) On (A, B) : block A is on block B.

(ii) On table (A) : block A is on table.

(iii) Clear (A) : block A has nothing on it.

(iv) Holding (A) : the arm holds block A.

(v) Arm empty : the arm holds nothing.

- **Remark :** We use the blocks world as an example, because :

(i) It is sufficiently simple and well-defined.

(ii) Easily understood.

(iii) Yet still provides a good sample environment to study planning.

(iv) Problems can be broken into nearly distinct subproblems.

Here, we can show how partial solutions need to be combined to form a realistic complete solution.

Unit II In Sem.

2.14.17 Sussman Anomaly

This is a problem in Artificial Intelligence described by Gerald Sussman.

In the problem, three blocks (labelled A, B and C) rest on table. The agent must stack the blocks such that A is a top B, which in turn is a top C. But it may only move one block at a time. The problem starts with B on the table, C a top A and A on table.

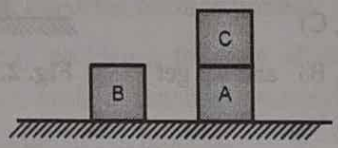


Fig. 2.14.4 : Block-World problem

A solution to the blocks world problem is a sequence of actions that allows us to start from an initial state and end up to a goal.

A mechanical **robot** arm can pick a block and place the cubes either on the table or on top of another block.

A planner typically separate the goal (stack A a top B atop C) into subgoals, such as :

(1) get A a top B ; (2) get B a top C

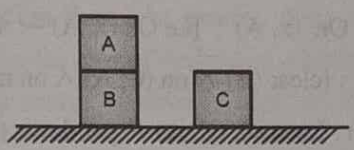


Fig. 2.14.5

Suppose the planner starts by pursuing subgoal 1, the straightforward solution is to move C out of the way, then move A a top B.

While sequence accomplishes subgoal (1), the agent cannot pursue subgoal (2) without undoing subgoal (1), since both A and B must be moved atop C.

Fig. 2.14.6

If instead the planner starts with Goal 2, i.e. to move B. But again the planner cannot pursue Goal 1 without undoing Goal 2.

Thus one of the solutions is

(i) unstack (C, A)

(ii) put down (C)

(iii) stack (B, C)

(iv) stack (A, B) and we get

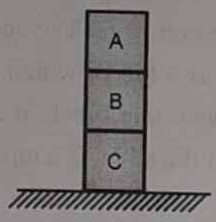


Fig. 2.14.7

☞ **Remark**

Sussman believed that intelligence requires a list of **exceptions** or **tricks**, and developed a modular planning system for 'debugging' plans.

Most modern planning systems handle this anomaly, but it is still useful for explaining why '**planning** is **non-trivial**'.

☞ **First order logic**

(i) On table (A) – block A is on the table

$$\neg \exists x \quad On(x, A) \quad or$$

$$\forall x \neg On(x, A) \quad [i.e\ On(x, A) \rightarrow x \text{ is on A}].$$

(ii) Initially : [clear (B) $\wedge$ on (C, A) $\wedge$ on table (A)]

(iii) Solution : [unstack (C, A) $\wedge$ put down (C)]

$\wedge$ clear (C) $\wedge$ stack (B, C) $\wedge$ stack (A, B)]

2.14.18 Heuristic Function : H_1

- This function gives better results. It requires less explored nodes and less execution time.
  - (i) First we check how many blocks have different positions.
  - (ii) Compare the node-state and goal-state (basically how many blocks are not where they should be in the current node-state).
  - (iii) The H-function H_1 is admissible because it never overestimates the cost of reaching the goal and is also consistent because for every node N and each successor P of N, the estimated cost of reaching the goal from N is no greater than the step cost of getting to ρ plus the estimated cost of reaching the goal from ρ.
- Thus heuristic function estimates the cost of getting from one place to another (from the current state to the goal state). Also called simply a heuristic.
- A heuristic function algorithm or simply heuristic is a shortcut of solving a problem when there are no exact solutions for it or the time to obtain the solution is too long.

☞ **Remark**

- Automated planning and scheduling problems are usually described in the planning Domain Definition Language (PDDL) notation which is an AI planning language for symbolic manipulation tasks. If something was formulated in PDDL notation, it is called a 'domain'.
- Therefore the task of stapling blocks is a block world domain, which stays in contrast to other planning problems like the 'dock worker robot' domain and the 'monkey and banana' problem.

2.14.19 Example of 'Block World Problem

(I) Local Heuristic

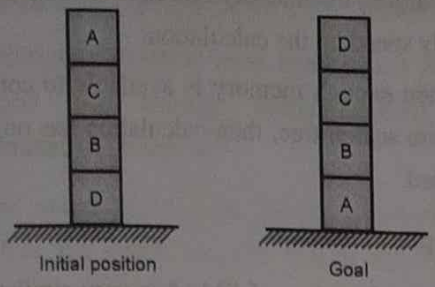


Fig. 2.14.8

(i) pickup (A) ∧ pickup (C) ∧ pickup (B) ∧ pickup (D).

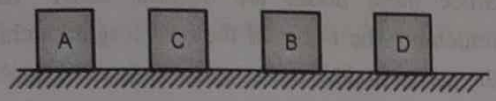


Fig. 2.14.9

(ii) Stack (B, A) ∧ stack (C, B) ∧ stack (D, C)

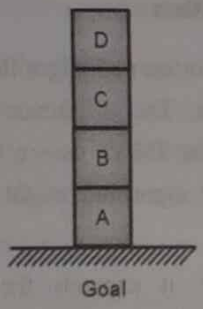


Fig. 2.14.10

(II) Global Heuristic

Here, every block has an incorrect support.

(i) pickup (A) ∧ pickup (C) ∧ pickup (B) ∧ pickup (D).

(ii) stack (B, A)

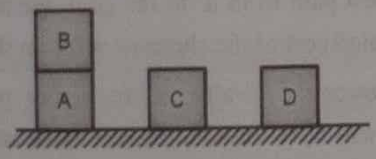


Fig. 2.14.11

(iii) stack (C, B) ∧ stack (D, C)

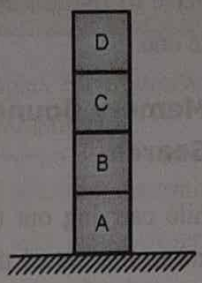


Fig. 2.14.12

2.14.20 Properties of Heuristic Functions

(1) The heuristic function for a node n is, h (n) = estimated cost of the cheapest path from the state at node n to a goal.

For example, for travelling salesman problem, the sum of the distances travelled so far can be a simple heuristic function. It is of two types : Maximise or minimise function.

(2) The more accurate the heuristic function estimate, the more direct the solution process.

(3) For many problems, the cost of computing the value of such a function outweighs the effort saved in the search process.

(4) If the heuristic function is very good, then no search is required to a solution.

(5) The purpose of heuristic function is to guide the search process in the most profitable direction by suggesting which path to follow first when more than one is available.

(6) **Optimality :** If there are many possible solution to a problem, optimal solution can be found out.

(7) **Accuracy :** Heuristic function can yield approximately good and precise solution.

(8) **Execution time :** In general, there is a trade-off between the cost of evaluating a heuristic function and the saving of execution time for the function.

Unit II In Sem.

(9) **Completeness :** If there are many solutions to a problem, we have to find all of them and choose the appropriate one.

2.14.21 Memory Bounded Heurestic Search

- Generally, while carrying out the search out the search, we tend to run out of the memory before time. This happens when the entire search paths are stored.
- To overcome this we have to remember partial solutions. But even then space requirements, memory bounded heuristic searches are used.
- Recursive best first search (RBFS), iterative deepening a* (IDA*) and memory bounded A* (MA*) are the algorithms that fall under this search.
- But there is drawback at RBFS and IDA*. They generate the same states again and again.
- The cost of memory-utilisation is same for all the above methods.
- Hence, if we are having more memory, we use memory-bound variants MA* or simplified memory bounded A* algorithms, (SMA*).
- SMA* or simplified memory bounded A* is a shortest path algorithm based on the A* algorithm.
- The main advantage of SMA* is that it uses a bounded memory, while the A* algorithm might need exponential memory. All other characteristics of SMA* are inherited from A*

Properties of SMA*

(1) It works with heuristic, just as A*

(2) It is complete if the allowed memory is high enough to store the shallowest optimal solution.

(3) It is optimal if the allowed memory is high enough to store the shallowest optimal solution.

(4) It avoids repeated states as long as the memory bound allows it.

(5) It uses all memory available.

(6) Enlarging the memory bound of the algorithm will only speed up the calculation.

(7) When enough memory is available to contain the entire search tree, then calculation has on optimal speed.

Implementation

- The implantation of SMA* is very similar to that of A*; the only difference is that nodes with the highest f-cost are deleted from the queue when there is not any space left.
- Since these nodes are deleted, SMA* has to remember the f-cost of the best forgotten child of the parent node.
- When it appears that all explored paths are worse than such a forgotten path, the path is regenerated.

SMA* algorithm

- SMA* is a shortest path algorithm that is based on A* algorithm. The difference between SMA*a and A* is that SMA* uses a bounded memory, while the A* algorithm might need exponential memory.
- Like the A*, it expands the most promising branches according to the heuristic.
- SMA* just like A* evaluates nodes by combining g(n)m, the cost to reach the node, and h(n), the cost to get from the node to the goal : $f(n) = g(n) + h(n)$.
- Since g(n) is the path cost from the start node to node n; and h(n) is the estimated cost of the cheapest path from n to the goal, we have f(n) = estimated cost of the cheapest solution through n.
- The lower the f value is, the higher priority the node will have. In addition, the node stores the f-value of the best forgotten successor.

2.15 LOCAL SEARCH ALGORITHMS

UQ. Write short note on : Local search Algorithms.
Q. 6(b), May 18, Q. 6(c), May 19, 6 Marks

- **In computer science local search** is a heuristic method for solving computationally hard **optimisation** problems.
- Local search starts from an initial solution and evolves the single solution that is mostly better solution.
- At each solution in this path it evaluates a number of moves on the solution and applies the most suitable move to take the step to the next solution. It continues this process for a high number of iterations until it is terminated.
- Local search uses a single search path and moves facts around to find a good feasible solution. Hence it is natural to implement.
- Local search algorithms are widely applied to numerous hard computational problems, including problems from **artificial intelligence, mathematics, operations research,** engineering and bioinformatics.

Some problems where local search is applied are :

(1) The **vertex cover problem,** in which a solution is a **vertex cover** of a **graph,** and the target is to find a solution with a minimal number of nodes.

(2) The **travelling salesman problem,** in which a solution is a **cycle containing** all nodes of the graph and the target is to minimise the total length of the cycle.

(3) The Hopfield Neural Networks problem for which finding stable configuration in Hop-field network.

2.16 SEARCH IN COMPLEX ENVIRONMENT

Search in complex Environment is more **complex** than single-state problem. It is :

(a) Searching with non-deterministic actions;
(b) Searching with partial observations.
(c) Online search and unknown environment.

Unit II In Sem.

(a) Searching with non-deterministic actions

- When the environment is fully deterministic and observable, then the agent knows what the effects of each action are. The agent can calculate which state results from any sequence of actions and always knows which state it is in.
- But when the environment is non-deterministic or partially observable (or both), percepts become useful.
- When the environment is nondeterministic, percepts tell the agent which of the possible outcomes of its actions has already occurred. Hence, the future percepts cannot be determined in advance and the agents future actions will depend on those future percepts.
- Hence the solution to a problem is not a sequence but a contingency plan or strategy that says what to do depending on what percepts are received.

Example : Consider an example of vacuum world. (We have already discussed it). Recall that the state space has eight states. And there are three actions – Left right and suck – and the goal is to clean up all the dirt (States 7 and 8).

If the environment is deterministic, then the problem is trivially solvable and the solution is an action sequence.

If the initial state is 1, then the action sequence [Suck, Right, suck] will reach a goal state 8.

Now, we introduce nondeterministic, in the form of vacuum cleaner.

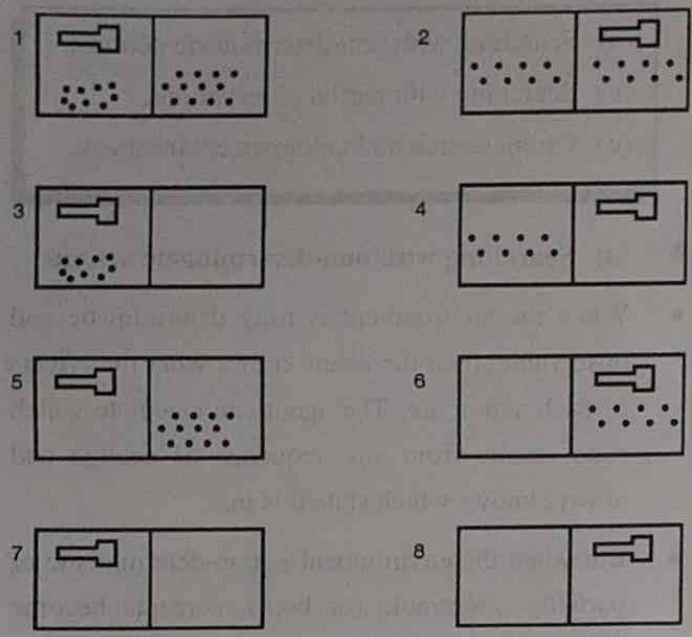


Fig. 2.16.1 : The eight possible states of the vacuum world, states 7 and 8 are goal states

- In the vacuum world, the suck action works as follows :
  - (i) When applied to a dirty square, the action cleans the square; and
  - (ii) When applied to a clean square, the action sometimes deposits dirt on the carpet.
- Here, we have to use a RESULTS function that returns a set of possible outcome states.

  For example, in the vacuum world, the suck action in state 1 leads to a state in the set [5, 7].
- We generalise the notion of a **solution** to the problem.
- For example, if we start in state 1, there is no single **sequence** of actions that solves the problem.

Instead, we need a contingency plan as :

[Suck, **if** state = 5 **then** {Right, suck}else []].

- Thus, solutions for nondeterministic problems can contain **if-then-else** statements :
- This means that they **are trees** rather than **sequences.**
- This allows the selection of actions based on contingencies arising during execution.

(b) Searching with Partial Observations

We now turn to the problem of searching with partial observations. The key concept required for solving partially observable problems is the **belief state,** It represents the agents's current belief about the possible physical states it might be in.

☞ Searching with no observation

- When the agent's percept's provide no **information at all**, we call it as sensorless problem.
- To solve sensorless problems, we search in the space of belief states. The solution is always a sequence of actions, because the percept's received after each action are completely predictable.

☞ Partially observable problems

- Since we supplied a belief-state problem to the AND-OR search algorithm, it returned a conditional plan that tests the belief rather than the actual state.
- In a partially observable environment the agent won't be able to execute a solution that requires testing the actual state.
- The design of a problem-solving agent for partially observable environments is quite similar to the simple problem-solving agent.

(i) The agent formulates a problem, and is called as search algorithm (such as AND-OR-GRAPH-SEARCH) to solve it, and executes the solution.

(c) Online search Agents and unknown Environment

An online search problem must be solved by an agent executing actions, rather than by pure computation. We assume a deterministic and fully observable environment. But we mention that the agent known only the following

(i) Actions (s), Which returns a list of action allowed in state (s);

(ii) The step-cost function c (s, a, s') – this cannot be used until the agent that s' is the outcome and

(iii) Goal-test (s).

Remark

The agent cannot determine RESULT (s, a) except by actually being in S and doing a.

Chapter Ends...

❑❑❑

UNIT III

CHAPTER 3

Adversarial Search and Games

Syllabus

Game Theory, Optimal Decisions in Games, Heuristic Alpha – Beta Tree Search, Monte Carlo Tree Search, Stochastic Games, Partially Observable Games, Limitations of Game Search Algorithms, Constraint Satisfaction Problems (CSP), Constraint Propagation: Inference in CSPs, Backtracking Search for CSPs.

3.1 GAME THEORY

GQ. Explain game tree.

- A **game tree** is a type of **recursive search function** that examines all possible moves of a strategy game, and their results, in an attempt to ascertain the optimal move.
- They are very useful for artificial intelligence in scenarios that do not require real-time decision making and have a relatively low number of possible choices per play.
- The most commonly-cited example is chess, but they are applicable to many situations. Game trees are generally used in board games to determine the best possible move. For the purpose of this article,
  Tic-Tac-Toe will be used as an example.
- The idea is to start at the current board position, and check all the possible moves the computer can make. Then, from each of those possible moves, to look at what moves the opponent may make.
- Then to look back at the computers. Ideally, the computer will flip back and forth, making moves for itself and its opponent, until the game's completion. It will do this for every possible outcome, effectively playing thousands (often more) of games.
- From the winners and losers of these games, it tries to determine the outcome that gives it the best chance of success.

☞ Limitations of Game Trees

(1) As mentioned above, game trees are rarely used in real-time scenarios (when the computer isn't given very much time to think).

(2) The method requires a lot of processing by the computer, and that takes time.

(3) For the above reason (and others) they work best in turn-based games.

(4) They require complete knowledge of how to move.

(5) Games with uncertainty generally do not mix well with game trees.

(6) They are ineffective at accurately ascertaining the best choices in scenarios with many possible choices.

GQ. How AI technique is used to solve tic-tac-toe problem?

☞ Heuristics function for Tic-Tac-Toe problem

The board used to play the Tic-Tac-Toe game consists of 9 cells laid out in the form of a 3× 3 matrix.

The game is played by 2 players and either of them can start. Each of the two players is assigned a unique symbol (generally 0 and X). Each player alternately gets a turn to make a move. Making a move is compulsory and cannot be deferred. In each move a player places the symbol assigned to him/her in a blank cell.

Seven classes of moves have been designed using the available heuristics. Each class of moves represents a set of functionally cohesive moves in order to achieve a certain objective during a game. These classes of moves are defined and their roles in playing the game are discussed below :

(1) **Prioritized selection (PS) :** The PS class of moves selects the blank cell with the maximum priority. If there exist more than one cell with the maximum priority, then any one of them can be selected. The PS class of moves makes sure that the player has control over the most important cells on the board.

(2) **Motion (M) :** The M class of moves finds all tracks with only one cell filled with the symbol assigned to the player and other two cells blank. Then one of these tracks with the highest priority is chosen. After that the blank cell with higher priority in the chosen track is selected. The M class of moves makes sure that the player continues filling a track in which there is still a chance to win.

(3) **Definitive offense (DO) :** This class of moves finds a track with exactly two cells filled with the symbol assigned to the player and the third cell blank. This blank cell is selected. This moves is meant to provide an immediate win to the player.

(4) **Definitive defense (DD) :** This class of moves finds a track with exactly two cells filled with the symbol assigned to the opponent player and the third cell blank. This blank cell is selected. It is meant to prevent the player from an immediate loss. If the moves are not used then the opponent definitely gets a chance to win in the subsequent move.

(5) **Tentative offense (TO) :** This class of moves finds all pairs of intersecting tracks in which both the tracks have exactly one cell filled with the symbol assigned to the player and the other two cells including the common one blank. All such common cells of the intersecting tracks are identified and the one with the maximum priority is selected. This move tries to keep the foundations of victory simultaneously on two tracks. If the TO class of moves can be applied then the player can win in the next move.

(6) **Tentative defense (TD) :** This class of moves finds all pairs of intersecting tracks in which both the tracks have exactly one cell filled with the symbol assigned to the opponent player and the other two cells including the common one blank. All such common cells of the intersecting tracks are identified and the one with the maximum priority is selected. It tries to undo the effect of the tentative offense class of moves applied by the opponent. If the move is not applied then the player can lose in the next move.

(7) **Diagonal correction (DC) :** The above six classes of moves may be insufficient to prevent a loss for the player moving second if either of the two diagonal tracks are filled in the first three moves. This may happen even if the losing player controls the more important cells. It is used to save the player from losing in such conditions.

In Tic-Tac-Toe, player alternate putting marks in a 3×3 array, one mark (X) and other mark (0).

Let the evaluation function, e(P) of a position P be given simply by, If P is not a wining position for either player.

e(P) = (number of complete row, column, or diagonal that are still open for X) – (number of complete row, column, or diagonal that are still open for 0)

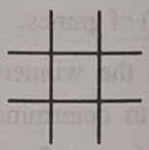
If P is a win for X,

$e(P) = \infty$ (a very large value)

If P is a win for 0,

$e(P) = -\infty$

Thus if P is



NOTES

We have, e(P) = 6.4 = 2

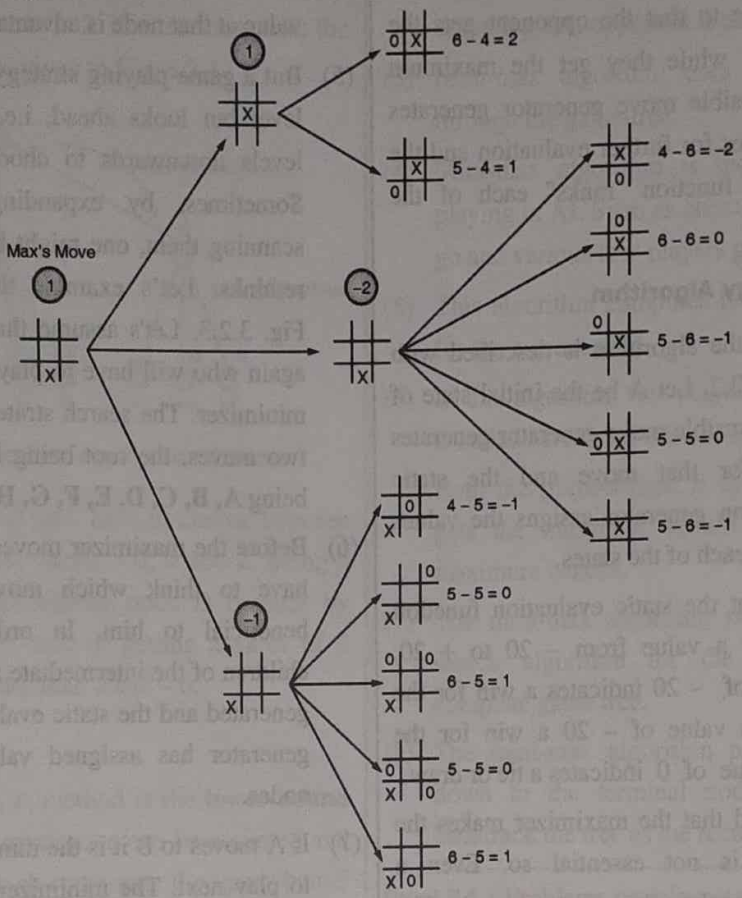


Fig. 3.1.1 : First state of search in Tic-Tac-Toe

3.2 OPTIMAL DECISIONS IN GAMES

Minmax Procedure

UQ. Explain Min max and Alpha beta pruning algorithms for adversarial search with example. (Q. 5(b), May 17, 10 Marks)

GQ. Explain Minmax procedure with suitable example. **OR** Consider the 2-ply search as shown below : (i) If the first player is a maximizing player, what move should be chosen under the mini-max strategy. (ii) What nodes should not be needed to be examined using α–β pruning technique?

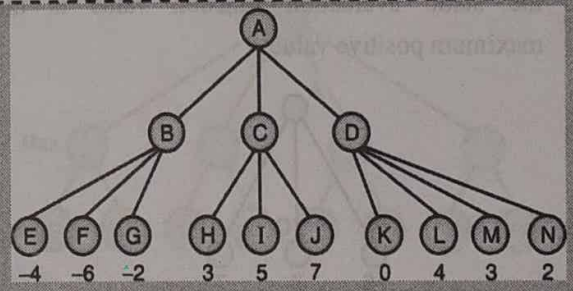


Fig. 3.2.1 : Minmax strategy

Ans. :

Minimax strategy

Minimax strategy is a simple look ahead strategy for two-person game-playing. Here one player is called a maximizer and the other is called a minimizer.

Both the adversaries, maximizer and minimizer fight it out to see to that the opponent gets the minimum benefit while they get the maximum benefit. The plausible move generator generates the necessary states for further evaluation and the static evaluation function "ranks" each of the positions.

☞ Minmax Strategy Algorithm

(1) The working of the algorithm is described with the aid of Fig. 3.2.2. Let A be the initial state of the game. The plausible move generator generates three children for that move and the static evaluation function generator assigns the values given along with each of the states.

(2) It is assumed that the static evaluation function generator returns a value from – 20 to +. 20, wherein a value of – 20 indicates a win for the maximizer and a value of – 20 a win for the minimizer. A value of 0 indicates a tie or draw.

(3) It is also assumed that the maximizer makes the first move. (It is not essential so. Even a minimizer can make the first move.) The maximizer, always tries to go to a position where the static evaluation function value is the maximum positive value.

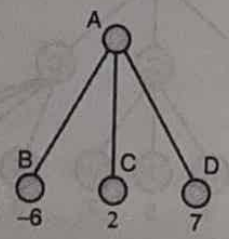


Fig. 3.2.2 : Initial state of the game

(4) The maximize being the player to make the first move; will move to node D because the static evaluation function value for that node is the maximum (figure). The same Fig. 3.2.2 that if the minimizer has to make the first move, he will go to node B because the static evaluation function value at that node is advantageous to him.

(5) But a game-playing strategy never stops with one level but looks ahead, i.e., moves a couple of levels downwards to choose the optimal path. Sometimes, by expanding these nodes and scanning them, one might be forced to retract or rethinks. Let's examine this with the help of Fig. 3.2.3. Let's assume that it is the maximizer again who will have to play first followed by the minimizer. The search strategy here tries for only two moves, the root being M and the leaf nodes being **A, B, C, D, E, F, G, H, I, J, L, M, and N.**

(6) Before the maximizer moves to B, C or D it will have to think which move would be highly beneficial to him. In order to evaluate, the children of the intermediate nodes B, C and D are generated and the static evaluation function value generator has assigned values for all the leaf nodes.

(7) If A moves to B it is the minimizer who will have to play next. The minimizer always tries to give the minimum benefit to the other and hence he will move to G (static evaluation function value – 6). This value is backed up at N.

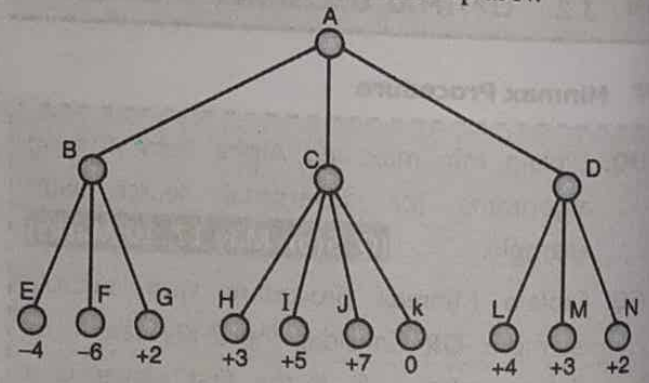


Fig. 3.2.3 : A game tree expanded by two levels and their associated static evaluation function value

(8) If A moves to C, then the minimizer will move to K (static evaluation function value = 0) which is the minimum of 3, + 5, 7 and 0. So the value of

0 is backed up at C. On similar lines, the value that is backed up at D is 2. The tree now with the backed up values is given in Fig. 3.2.4.

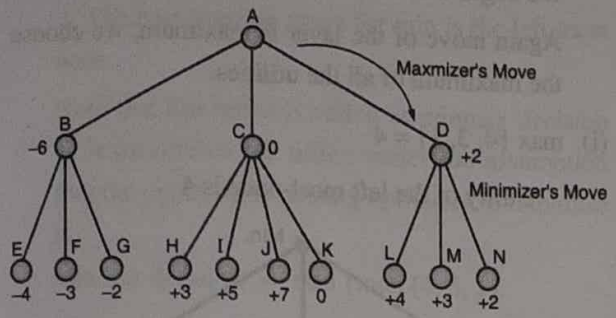


Fig. 3.2.4 : Maximizer's move for the tree

(9) The maximizer will now have to choose between B, C or D with the values – 6, 0 and 2. Being a maximizer, he will choose node D because by choosing so, he is sure of getting a value of 2 which is much better than 0 and – 6.

☞ **α – β pruning**

1. In α – β pruning, α method is the **lower bound** on the value the maximizer can be assigned and the other is β, which represents the upper bound on the value the minimizer can be assigned.
2. Here, the maximizer has to play first followed by the minimizer. Thus maximizer assigns – 6 and B which is passed back to A. This is replaced by 3, value passed by C as A has the maximizer move. Now A will not be examining D and its children. Since value of K is zero and D is having minimizer move making its value 0 or less than 0 only. Thus the tree with the root K will be pruned, which would save a lot of time in searching.

3.2.1 Properties of Min Max Algorithm

(1) Mini-max algorithm is a recursive or backtracking algorithm which is used in decision-making and game theory.

(2) It provides an optimal move the for player assuming that opponent is also playing optimally.

(3) Mini-max algorithm uses recursion to search through the game-tree.

(4) Mini-max algorithm is mostly used for game playing in AI. Such as chess, checkers, tic-tac-toe, go and various tow-players game.

(5) This algorithm computes the mini-max decisions for the current state.

(6) In this algorithm two players play the game, one is called MAX and other is called MIN.

Both the players fight it as the opponent player gets the minimum benefit while they get the maximum benefit.

(7) The mini-max algorithm performs a depth-first search algorithm for the exploration of the complete game tree.

(8) The mini-max algorithm proceeds all the way down to the terminal node of the tree, then backtrack the tree as the recursion.

Unit III End Sem.

Ex. 3.2.1 : Problems on min-max search on game tree as shown in the Fig. Ex. 3.2.1.

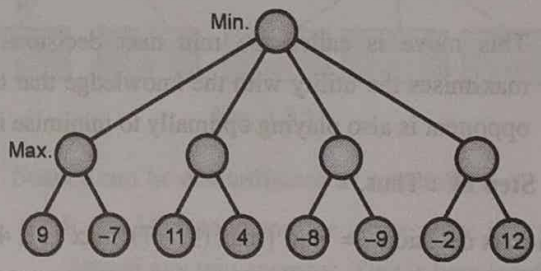


Fig. Ex. 3.2.1

☑ **Soln. :**

▶ **Step (I) :** Since the given move of the player is max; we calculate first maximum of all nodes.

We begin with left node of the layer above the terminal; to calculate the utility of the left node.

Now, max {9, – 7} = 9

∴ Utility of the left most node = 9 ...(i)

The utility of the next node of the same layer is max {11, 4} = 11 ...(ii)

Again the utility of the node (beginning from left)

= max {– 8, – 9} = – 8

and the utility of the last node of the same layer is max {– 2, 12} = 12

▶ **Step II :**

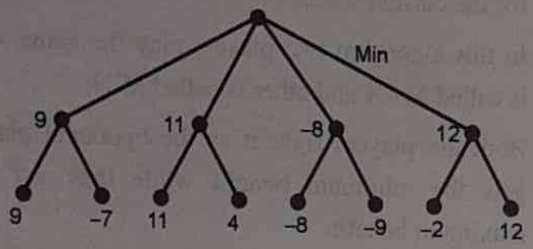


Fig. Ex. 3.2.1(a)

Now, we calculate the utility values, considering one layer at a time, till we reach the root of the tree, i.e. the top-most point.

Here we have 3 layers, so we can directly evaluate min {9, 11, –8, 12} = – 8

▶ **Step III :** Thus, the best opening move for min is the third node.

This move is called the min max decision. It maximises the utility with the knowledge that the opponent is also playing optimally to minimise it.

▶ **Step IV :** Thus,

Min-max decision = min {max (9, – 7), max {11, 4},
max {–8, – 9}, max {–2, 12}}
= min {9, 11, – 8, 12} = – 8

Ex. 3.2.2 : Apply min-max search on game tree given in the Fig. Ex. 3.2.2.

☑ **Soln. :**

▶ **Step I :** Here the given move of the player is max; we calculate first maximum of all nodes in the last layer, to determine the utilities of the terminal nodes.

We begin with the left node of the layer.

Again move of the layer is maximum, we choose the maximum of all the utilities.

(i) max {4, 3, 1} = 4

∴ Utility of the left most-node is 4

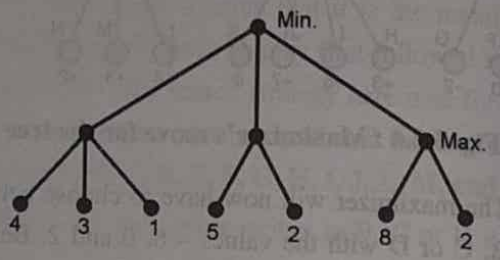


Fig. Ex. 3.2.2

▶ **Step II :** Again we evaluate maximum of the middle node in the same layer, i.e.

max {5, 2} = 5

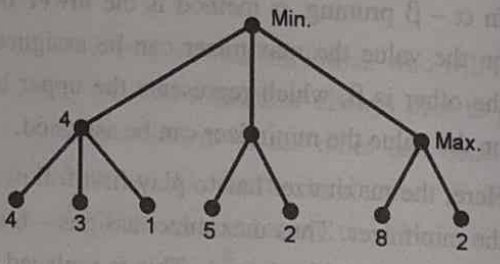


Fig. Ex. 3.2.2(a)

▶ **Step III :** Similarly, for the right –most of the same layer, we evaluate max {8, 2} = 8

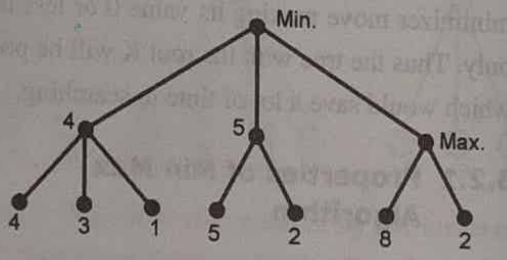


Fig. Ex. 3.2.2(b)

▶ **Step IV :** Here there are only 3 layers, so we immediately reach to the root. At the root, i.e. the

topmost point, min has to choose the minimum value.

So, we evaluate min {4, 5, 8} = 4

∴ The best opening move for min is the left most node

Note that this move is called as minmax decision as it maximises the utility under the assumption that the opponent is playing optimally to minimise it.

∴ Minmax decision = min {max (4, 3, 1), max. {5, 2}, max. {8, 2}}
= min {4, 5, 8} = 4

3.3 HEURISTIC ALPHA-BETA TRESS SEARCH

GQ. Explain Alpha-beta pruning.

UQ. Explain Min max and Alpha beta pruning algorithms for adversarial search with example. (Q. 5(b), May 17, 10 Marks)

UQ. Define Alpha and Beta value in game tree? (Q. 1(c), May 16, 4 Marks)

Alpha-beta search reduces the number of nodes explored in minimax strategy. It reduces the time required for the search.

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

Alpha values

1. At a Max node we will store an alpha value
2. A lower bound on the exact minimax score
3. The true value might be $\geq \alpha$
4. If we know Min can choose moves with **score** $< \alpha$
5. Then Min will never choose to let Max go to a node where the score will be α or more.

Beta values

- At a min node
  1. The beta value is an upper bound on the exact minimax score.
  2. The true value might be $\leq \beta$.
  3. If we know Max can choose movers with **score** $> \beta$.
- Then Max will never choose to let Min go to a node where the score will be β or less.

Alpha beta in action

Why can we cut off search?

- $\beta = 2 < \alpha = 3$ where the a value is at an ancestor node.
- At the ancestor node max had a choice to a score of at least 3.
- Max is not go going to move right to let min guarantee a score of 2.

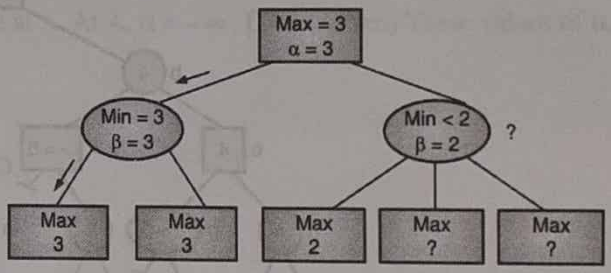


Fig. 3.3.1

- **Search can be discontinued at a node if :**
  1. It is a Max node and the alpha value is ≥ the beta of any min ancestor. This is beta cutoff
  2. Or it is a min node and the beta value is ≤ the alpha of any max ancestor. This is alpha cutoff.

3.3.1 Calculating Alpha-Beta Values

- Alpha-Beta calculations similar to Minimax, but the pruning rule cuts down search.

Unit III End Sem.

- Final backed up value of node.
  1. Might be the maximum value.
  2. Or might be an approximation where search cut off.
- Less than the minimax value at a Max node.
- More than the minimax value at a Mid node.
- We don't need to know the true value.

3.3.2 Calculating Alpha Values at a Max Node

- After we obtain the final backed up value of the first child.
  1. We set cx of the node to this value.
- When we get the final backed up value of the second child.
  1. We increase ix if the new value is larger.
- When we have the final child, or if β cutoff.
  1. α value becomes the final backed up value.
  2. Only then can we set the β of the parent Min node

3.3.3 Calculating Beta Values at Min

- After we obtain the final backed up value of the first child, we set β of the node to this value
- When we get the final backed up value of the second child
  1. We decrease β if the new value is smaller
- When we have the final child, or if a – cutoff
  1. β value becomes the final backed up value

3.3.4 Alpha-Beta Pruning Example

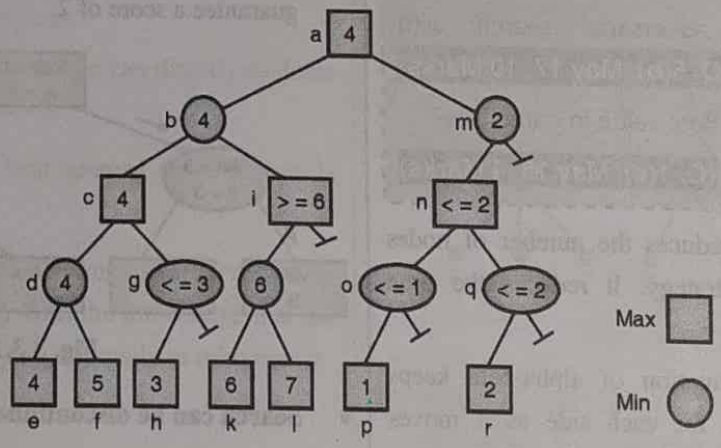


Fig. 3.3.2

- MINIMAX (n, alpha, beta)
- IF n is at the search depth, RETURN V(n)
- FOR each child m of n
- value = MAXIMIN (m, alpha, beta) IF value < beta , beta = value
- IF beta < = alpha, return alpha RETURN beta

Performance of Alpha-Beta pruning

1. Efficiency depends on the order in which the nodes are encountered at the search frontier.

2. Optimal - $b^{1/2}$ - if the largest child of a MAX node is generated first, and the smallest child of a MIN node is generated first.
3. Worst - b.
4. Average $b^{3/4}$ - random ordering.

⇒ **Example of alpha-beta pruning**

Ex. 3.3.1 : Apply alpha-beta pruning on two player search tree shown in the Fig. Ex. 3.3.1.

$\alpha = -\infty$; $\beta = \infty$

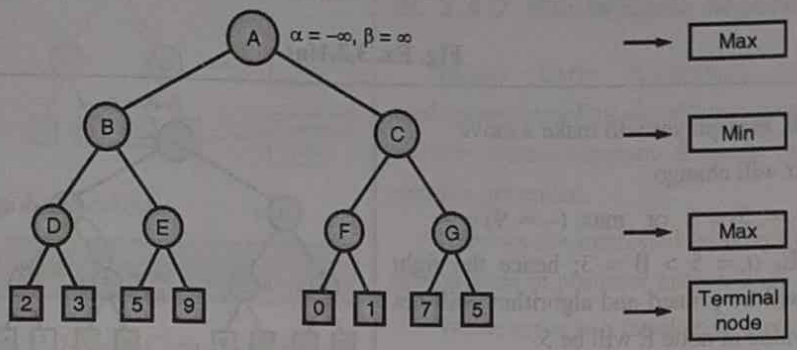


Fig. Ex. 3.3.1: Two player search tree

Unit III End Sem.

☑ **Soln. :**

- **Step I :** Initially the max. Player will begin the move at A. At A, $\alpha = -\infty$, $\beta = \infty$ (given) These values of α, β are transferred to node B.

  At B also, $\alpha = -\infty, \beta = \infty$

  Node B also passes these values of α and β to node D.

- **Step II :** At D, max player will start its turn.

  At D, $\alpha = -\infty$

  ∴ At D, max $(-\infty, 2) = 2$ and max $(-\infty, 3) = 3$

  Also, max $(2, 3) = 3$

  ∴ The value of node D is 3 and value of α is also (say) 3.

- **Step III :** Since $\alpha = 3 < \beta = \infty$; algorithm backtracks to node B. At B, min player will move and value of β $(= \infty)$ will change.

  Value of β will be compared to the nodal value of D = 3; and min $(\infty, 3) = 3$

  ∴ At B, $\alpha = -\infty$, $\beta = 3$

- **Step IV :** Since $\alpha < \beta$ at B, algorithm traverses the next successor of Node B. Which is node E, and the values of $\alpha = -\infty$, $\beta = 3$ are transferred to node E.

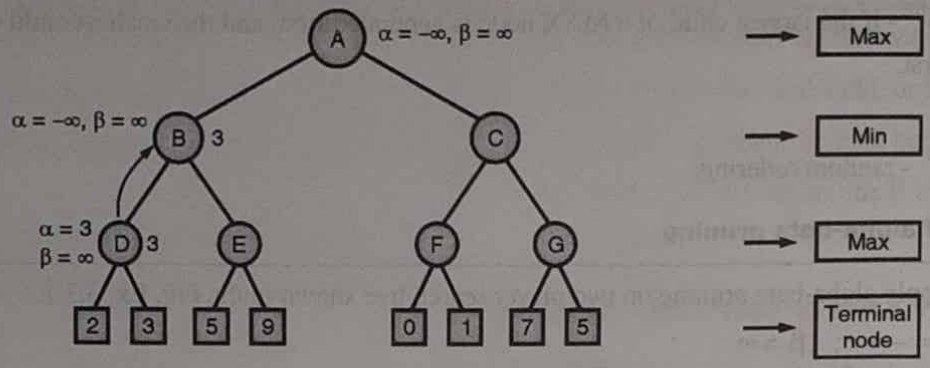


Fig. Ex. 3.3.1(a)

▶ **Step V :** At E, max player will make a move

∴ Value of α will change

Now max. $(-\infty, 5) = 5$ or max. $(-\infty, 9) = 9$

∴ at node E, $\alpha = 5 > \beta = 3$; hence the right success or of E will be pruned and algorithm will not traverse and the value at node E will be 5.

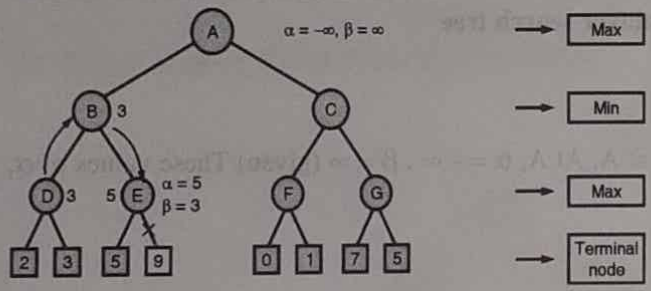


Fig. Ex. 3.3.1(b)

▶ **Step VI :** Now algorithm again backtrack the tree, from node B to node A.

At A, there is max player, and α will take the maximum value, that is 3, ∵ max $(\infty, 3) = 3$ and $\beta = \infty$

These two values i.e.; $\alpha = 3$, $\beta = \infty$ will transfer to the successor C. And at node C, again $\alpha = 3$, $\beta = \infty$, and these values again will get transferred to node F.

At F, there is max. Player

▶ **Step VII :** At node F, we compare the value of α with the left node O, and

$$\max (3, 0) = 3$$

Now node value of F = max (0, 1) = 1

$$\therefore \quad \alpha = 3 > \text{node value of F}$$

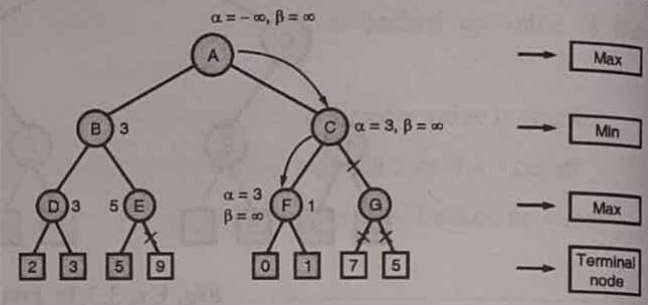


Fig. Ex. 3.3.1(c)

▶ **Step VIII :** Now the right successor is pruned. (∵ α > node value of F)

∴ node F gets back to node C and at C,

$$\alpha = 3, \ \beta = \infty$$

Here β will change, since C is min player node. At C node value is 1,

$$\therefore \quad \beta = \min (\infty, 1) = 1$$

Again $\alpha > \beta$ (∵ $\alpha = 3, \beta = 1$)

∴ Next node G will be pruned and algorithm stops computing at G

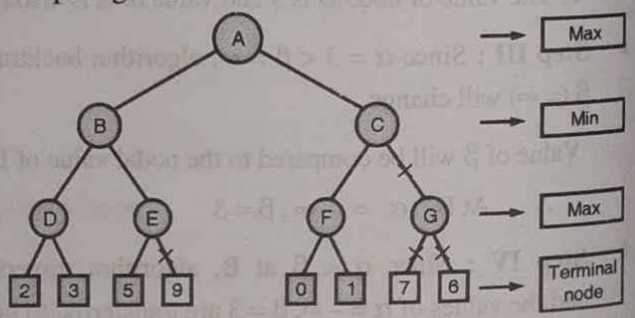


Fig. Ex. 3.3.1(d)

Step IX : At C, $\alpha = 3, \beta = 1$;

($\because \alpha > \beta$) ; C goes back to A, and for A, max.(3, 1) = 3

$\therefore$ optimiser value for this optimiser is β.

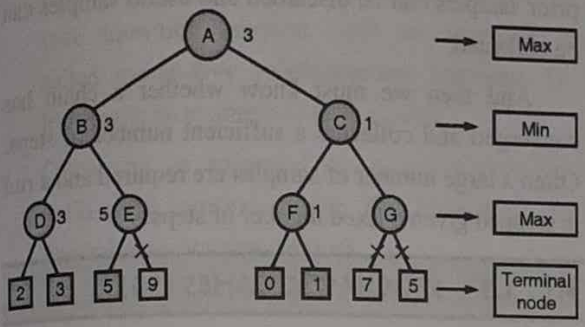


Fig. Ex. 3.3.1(e)

3.4 MONTE CARLO TREE SEARCH

In **computer science, Monte Carlo Tree Search (MCTs)** is a **heuristic search** algorithm for some kinds of decision processes, most notably those employed in **software** that plays **board games**.

MCTs is used to solve the game tree. It is also used in other board games like **chess** and games with incomplete information such as bridge and pocker.

MCTs is also used in self-driving cars, in Tesla's autopilot software.

3.4.1 Monter Carlo Method (Background)

Monte Carlo method uses random sampling for deterministic problems which are otherwise difficult to solve using other approaches.

Bruce Abramson in his Ph.D. thesis combined **minimax search** with an **expected-outcome model** based on random game players. According to him, the expected-outcome model "is shown to be precise, accurate, easily estimable, efficiently calculable, and domain independent."

He experimented in depth with **'Tic-tac-toe'** and then with machine-generated evaluation functions for chess.

Such methods were successfully applied to heuristic search in the field of **automated search.** There by improving the exponential search times of uninformed search algorithms.

3.4.2 Monte Carlo Algorithms

Monto Carlo Algorithms (also called as randomised sampling algorithms) provide approximate answers whose accuracy depends on the number of samples generated.

Here we are interested in sampling applied to the computation of posterior probabilities.

Unit III End Sem.

We describe two families of algorithms :

(1) Direct sampling method.

(2) Markov chain sampling method.

3.4.2(A) Direct Sampling Method

- Direct sampling can also refer to sampling from a particular **probability distribution** instead of a specified population.
- For example, in **Bayesian theory**, sometimes it is necessary to sample from a **posterior distribution**.
- It summarises what we know after the data has been observed. Knowing posterior distribution, we can directly sample from it. Sometimes it is not possible to sample from it because of complexity or high-dimensionality.

General definition

Definition : Direct sampling is when the sample is taken from the actual population **and not**, for example, some kind of related record like voter registration cards, or census forms etc.

In such cases, it is carried out as follows :

(i) The auditor selects each item sampled based on some judgemental decisions.

(ii) It excludes equal opportunity selections and

(iii) It depends more on intentional selection of items based on criteria, which relates more or less with sample representative of the population.

3.4.2(B) Markov Chain Sampling Method

- Probabilistic inference involves estimating an expected value using a probabilistic model. Often directly inferring values is not tractable by probabilistic models. So we use approximation method.
- Markov chain sampling method provides a class of algorithms for systematic random sampling from high dimensional probability distributions.
- Markov chain methods draw samples where the next sample is dependent on the existing sample, called a Markov chain. It is just unlike Monte Carlo sampling methods that draw independent samples from the distribution.
- This allows the algorithms to narrow the quantity that is being approximated from the distribution, even with a large number of random variables.

3.4.3 Markove Chain Monte Carlo Algorithms (MCMC)

Monte Carlo algorithms are attempts at carefully harnessing properties of the problem in order to construct the chain efficiently.

The sequence is constructed as :

(i) The first sample is generated from the prior, and

(ii) Successive samples are generated from distributions that are closer to the desired posterior.

Probabilistic Models

MCMC algorithms are sensitive to their starting point, and often require a warm-up phase to move in towards a fruitful part of the search space. And then prior samples can be discarded and useful samples can be collected.

And then we must know whether a chain has converged and collected a sufficient number of steps. Often a large number of samples are required and a run is stopped given a fixed number of steps.

3.5 STOCHASTIC GAMES (SG)

- Stochastic Games are also called as Mark games. It extends Markov Decision Process (MDP) to the case where there are multiple players in a common environment. These agents perform a joint action that defines both the reward obtained by the agents and the new state of the environment.
- A stochastic game is a dynamic game with **probabilistic transitions** played by one or more players.
- The game is played in a sequence of stages. At the beginning of each stage, the game is in a certain state. The players select actions, and each player becomes successful and that depends on the current state and the chosen actions.
- The game then moves to a new random state whose distribution depends on the previous state and the actions chosen by the players.
- The procedure is repeated at the new state, and the play continues for a finite or infinite number of stages.
- The total success to a player is taken to be the discounted sum of the stage successes or the limit inferior of the averages of the stage successes.

- A learning problem arises when the agent does not know the reward function or the state transition probabilities.
- In an agent directly learns about its optimal policy without knowing either the reward function or the state transition function, such an approach is called **model-free reinforcement learning**. Q-learning is an example of such a model.
- Q-learning is extended to a non-cooperative multi-agent context, using the framework of general-sum stochastic games.
- A learning agent maintains Q-functions over joint actions and performs updates based on assuming Nash equilibrium behavior over the current Q-values. The challenge is convergence of the learning protocol.
- Also, **poker tournaments are stochastic games**, and there are no known algorithms that are guaranteed to converge to an equilibrium in three-player stochastic games.

3.6 PARTIALLY OBSERVABLE GAMES

- Chess game, even though, described as war in miniature, it lacks one major characteristics of war, namely, **partial observability**. In the war, the existence and disposition of enemy units is often unknown until revealed by direct contact. Hence, warfare includes the use of scouts and spies to gather information. Partially observable games share these characteristics.
- In many real-world problems, there is a dynamic interaction between competitive agents.
- Partially observable stochastic games (POSGs) are among the most general formal models that capture such dynamic scenarios. The model captures stochastic events, partial information of players about the environment, and the scenario does not have a fixed horizon.
- Hence, the research has been focussed on subclasses of POSGS that have a value of the game and admit designing optimal algorithms.

We propose such a subclass for two-player zero-sum games with discounted-sum objective function-POSGS with public observations (POPOSGs) – where each player is able to reconstruct beliefs of the other player over the unobserved states.

Our results include :

(1) Theoretical analysis of POPOSGs and their value functions showing convexity (concavity) in beliefs of maximising (minimising) player,

(2) A novel algorithm for approximating the value of the game, and

(3) A practical demonstration of scalability of our algorithm.

Experimental results indicate that our algorithm can closely approximate the value of non-trivial games with hundreds of states.

3.6.1 Algorithm for Partially Observable Stochastic Games (POSGs)

- The algorithm is a synthesis of dynamic programming for partially observable Markov decision processes (POMDPs) and iterated elimination of dominated strategies in normal form games.
- When applied to finite-horizon POSGs, the algorithm iteratively eliminates very weakly dominated strategies without first forming a normal form representation of the game.
- For the special case in which agents share the same successes, the algorithm can be used to find an optimal solution.
- Partially observable decision-making in robot teams is fundamentally different from decision making in fully observable problems. Team

Unit III End Sem.

members cannot simply apply single-agent solution techniques in parallel.

- Instead we have to model the problem according to game-theoretic frameworks.
- We purpose an algorithm that approximates POSGs as a series of smaller, related Bayesian games, using heuristics such as QMDP to provide the future discounted value of action.
- This algorithm results in policies that are locally optimal with respect to the selected heuristic.

3.7 LIMITATIONS OF GAME SEARCH ALGORITHMS

In game theory and in decision-making Mini-max algorithm is used.

(i) It provides an optimal move for the player assuming that opponent is also playing optimally.

(ii) Mini-max algorithm uses recursion to search through the game-tree.

(iii) Mini-max algorithm is used for game playing in AI such as chess, tic-tac-toe, and various tow-players game. This algorithm computes the minimax decision for the current state.

(iv) In this algorithm two players play the game, one is called MAX and other is called MIN.

(v) Both players of the game are opponent of each other, where MAX will select the maximised value and MIN will select the minimised value.

(vi) The min-max algorithm performs a depth-first search algorithm for the exploration of the complete game tree.

(vii) The min-max algorithm proceeds all the way down to the terminal node of the tree, then backtrack the tree as the recursion.

(viii) A disadvantage of the minimax algorithm is that each board state has to be visited twice : one time to find its children and a second time to evaluate the heuristic value.

3.8 CONSTRAINT SATISFACTION PROBLEMS (CSP)

In artificial intelligence, constraints enumerate the possible values a set of variables may take in a given world. A possible world is a total assignment of values to variables representing a way the world (real or imaginary) could be.

Constraints in problem solving are :

(1) A problem is an issue you can resolve, while a constraint is an issue you cannot resolve. These are the simplest definitions of these two terms. One can define it in terms of one's control over the situation. A problem is an issue where you have control over while a constraint is one where you cannot have control over.

(2) In artificial intelligence and operations research, constraint satisfaction is the process of finding a solution to a set of constraints that impose conditions that the variable must satisfy.

(3) A solution is therefore a set of values for the variables that satisfies all constraints -that is, a point in the feasible region.

GQ. Define constraint satisfaction problem.

❑ **Definition :** Constraint satisfaction problems are problems whose states and goal test conform to a standard, structured and very simple representation. Search algorithm can be defined that take advantage of the structure of states and can use general-purpose rather than problem-specific heuristics to enable the solution of large problems.

- Constraint satisfaction is the search procedure that operate in a space of constraint sets.

- The initial and state contains the constraints that are originally given in the problem description.
- A goal state is any state that has been constraint "enough" whose "enough" must be defined for each problem.
- Constraint satisfaction problem has various states and goal test, a traditional problem has been converted into standard structured and very simple representation.
- The general-purpose routines can be used to access a special representation which has more benefit than problem-specific heuristics. These routines combined with special structure can find solution of large problems.
- The structure of the problem is represented in different form such as, standard representation of the goal test.
- The revealed structured is very efficient in many ways such as,
  (i) Problem decomposition.
  (ii) To understand structure of problem and the difficulty of solving it and their connection.
- A constraint satisfaction problem is defined by a set of variables X_1, X_2, ... X_n and a set of constraints
  $C_1 r C_2 / ..., C_m$.
- Each variable X_j has a nonempty domain D_1 of possible values. Each constraint $C^{\wedge}$ involves some subset of the variables and specifies the allowable combinations of values for that subset.
- A state of the problem is defined by an assignment of values to some or all of the variables.
  $[X\{ = V_i, X = V_j, ...\}$.
- An assignment that does not violate any constraints is called a consistent or legal assignment. A complete assignment is one in which every variable is maintained and a solution to a CSP is, a complete assignment that satisfies all the constraints. Some CSPs also require a solution that maximizes an objective function.

☞ **Constraint satisfaction is a 2-step process**

(1) Constraint are discovered and propagated as for as possible throughout the system. Then if this is still not a solution, search begins.

(2) A guess about something is made and added as a new constraint, prorogation can then occur with this new constraint.

Two constraints cannot contain the same value.

3.8.1 Advantages of CSP

(1) As the representation of states have standard pattern [that is a set of variables with assigned values], we can design successor function and goal test in generic way that will apply to all CSPs.

(2) We can develop effective generic heuristic that require no additional domain specific expertise.

(3) The structure of the constraint graph can be used to simplify the solution process in some cases giving exponential reduction in complexity.

3.8.2 (Examples) of Constraint Satisfaction Problems (CSPS)

- Problems are mathematical questions defined as a set of objects whose state must satisfy a number of constraints or limitations.
- CSPs represent the entities in a problem as a homogenous collection of finite constraints over variables, which is solved by constraint satisfaction methods.
- CSPs are the subject of research in both artificial intelligence and operations research, since the regularity in their formulation provides a common basis to analyse and solve problems of **many seemingly** unrelated families.

Unit III End Sem.

3.8.3 8 Puzzle Problem (N – Queens Problem)

UQ. How AI technique is used to solve 8 puzzle problem? **Q. 3(b), Dec. 18, 10 Marks, Q. 1(c), May 18, Q.1(c), May 17, Q.3(b), Dec. 16, Q. 1(c), Dec. 15, 3 Marks**

One of the most common example of the backtracking is to arrange N queens on an N × N chessboard such that no queen can strike down any other queen.

A queen can attack horizontally, vertically or diagonally. If no safe place is left, then we change the position of the previously placed queen.]

- We discuss N Queen as another example that can be solved by using Backtracking.
- The N Queen is the problem of placing N chess queens on an N × N chess board so that no two queens attack each other.
- We give below a solution for 4 Queen problem.

☞ Backtracking Algorithm

- The idea is to place queens one by one in different columns, starting from the leftmost column.
- When we place a queen in a column, we check for clashes with **already** placed queens.
- In the current column, if we find a row for which there is no clash, 'we mark this row and column as part of the solution.
- If we do not find a row due to clashes then we backtrack and return false.

(1) Start in the leftmost column :

(2) If all queens are placed, return true.

(3) Try all rows in the current column.

☞ Do following for every tried row

(a) If the queen can be placed safely in this row, then mark this [row, column] as part of the solution and check if placing queen here leads to a solution.

(b) If placing the queen in [row, column] leads to a solution then return true.

(c) If placing queen doesn't lead to a solution then unmark this [row, column] (back track) and go to step (a) to try other rows.

(d) If all rows have been tried and nothing worked, return false to trigger backtracking.

Output : The 1 Values indicate placements of queens

One Solution				Another solution			
0	0	1	0	0	1	0	0
1	0	0	0	0	0	0	1
0	0	0	1	1	0	0	0
0	1	0	0	0	0	1	0

❒ **Remark :** With the constraints mentioned above there are only 2 solutions to the 4 queens problem. As one can see from 2 solutions, no two queens share the same row, same column or diagonal.

☞ Optimisation in is – safe () function

The idea is not to check every element in right and left diagonal instead use property of diagonals :

(1) The sum of i and j is constant and unique for each right diagonal where i is the row of element and j is the column of element.

(2) The difference of i and j is constant and unique for each left diagonal where i and j are row and column of element respectively.

(1) **The eight queens puzzle :** is the problem of placing eight chess queens on an 8×8 chessboard so that no two queens threaten each other, thus, a solution requires that no two queens share the same row, column or diagonal.

The eight queens puzzle is an example of the more general n queens problem of placing n non-attacking queens on an $n \times n$ chessboard, for which solutions exist for all natural numbers n with the exception of $n = 2$, $n = 3$

(2) **Heuristic function for 8-puzzIe problem :** The 8-puzzle belongs to the family of **sliding-block puzzles**, which are often used as test problems for new search algorithms in AI.

This general class is known to be NP-complete, so one does not expect to find methods significantly better in the worst case than the search algorithms described in this chapter and the next. The 8-puzzle has 9!/2 = 181,440 reachable states and is easily solved.

The 15-puzzle (on a 4×4 board) has around 1.3 trillion states, and random instances can be puzzle (on a 5×5 board) has around 10^{25} states, and random instances are still quite difficult to solve optimally with current machines and algorithms.

(3) The goal of the 8-queens problem is to place eight queens on a chessboard such that no queen attacks any other. (A queen attacks any piece in the same row, column or diagonal.) Fig. 3.8.1 shows an attempted solution that fails: the queen in the rightmost column is attacked by the queen at the top left.

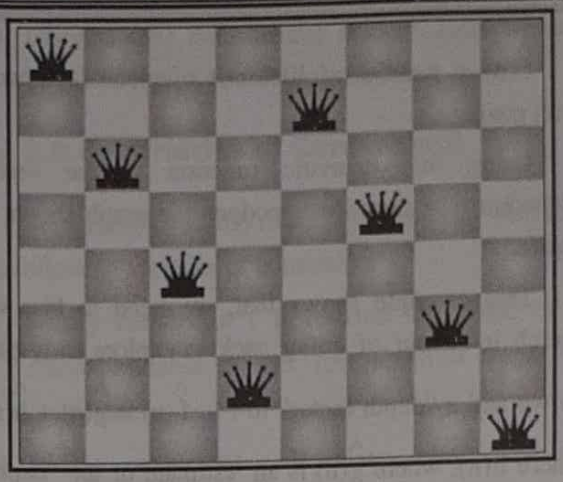


Fig. 3.8.1 : Solution to the problem

(4) Although efficient special-purpose algorithms exist for this problem and the whole n- queens family, it remains an interesting test problem for search algorithms. There are two main kinds of formulation. An incremental formulation involves operators that augment the state description, starting with an empty state; for the 8-queens problem, this means that each action adds a queen to the state. A complete-state formulation starts with all 8 queens on the board and moves them around. In either case, the path cost is of no interest because only the final state counts.

The first incremental formulation one might try is the following :

- **States :** Any arrangement of 0 to 8 queens on the board is a state.
- **Initial state :** No queens on the board.
- **Successor function :** Add a queen to any empty square.
- **Goal test :** 8 queens are on the board, none attacked.

As a heuristic function for 8-puzzle problem, we might try using the number of tiles out of place as a measure of the goodness of a state description :

Unit III End.Sem

$\hat{f}(n)$ = number of tiles out of place (compared with goal)

Using this heuristic function in the search procedure just described produce the graph shown in figure.

This example shows that, we need to bias the search in favour of going back to explore early path so we add a "depth factor" to $\hat{f}$. $\hat{f}(n) = \hat{g}(n) + \hat{h}(n)$, where $\hat{h}(n)$, where $\hat{g}(n)$ is an estimate of the "depth" of n in the graph, and $\hat{h}(n)$ is an heuristic evaluation of node n.

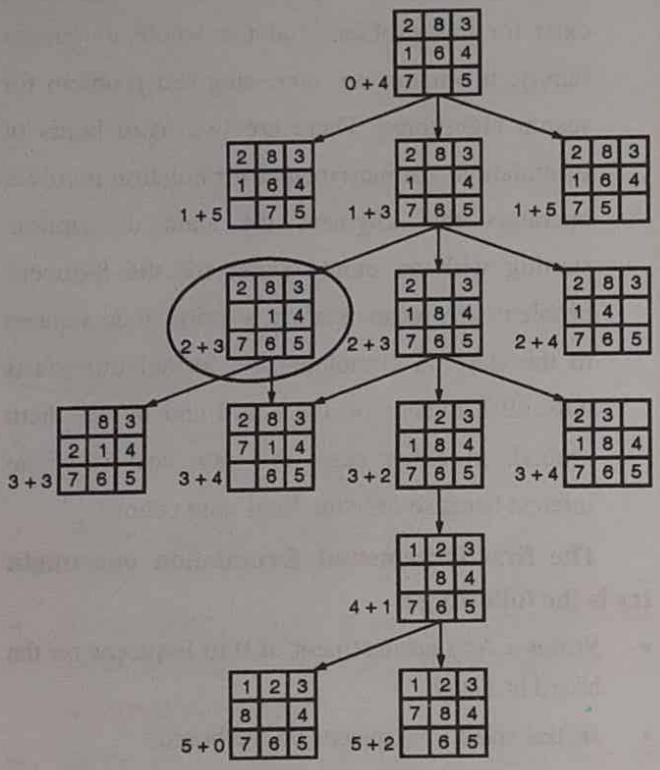


Fig. 3.8.2 : Heuristics search using

$$\hat{f}(n) = \hat{g}(n) + \hat{h}(n)$$

The only symmetrical solution to the eight queens puzzle (upto rotation and reflection)

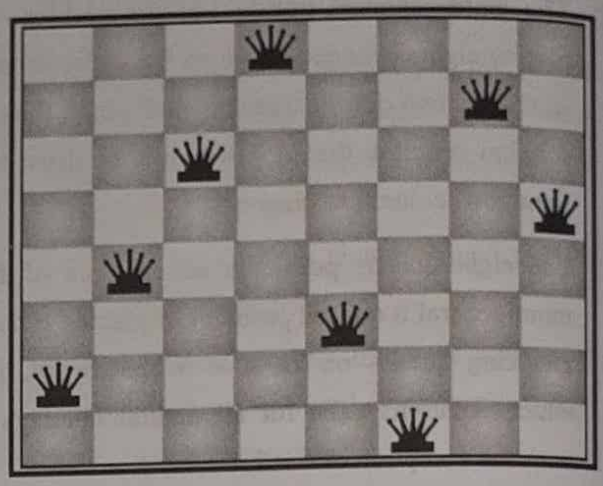


Fig. 3.8.3

Here all the queens are safe.

3.8.4 Graph Colouring

In graph theory, graph coloring is a special case of graph labelling. It is an assignment of labels traditionally called 'colors' to elements of a graph subject to certain constraints.

In its simplest form, it is a way of coloring the vertices of a graph such that no two adjacent vertices are of the same color, this is called a 'vertex coloring'. Similarly, an 'edge coloring' assigns a color to each edge so that no two adjacent edges are of the same color, and a 'face coloring' of a planar graph assigns a color to each face or region so that no two faces that share a boundary have the same color.

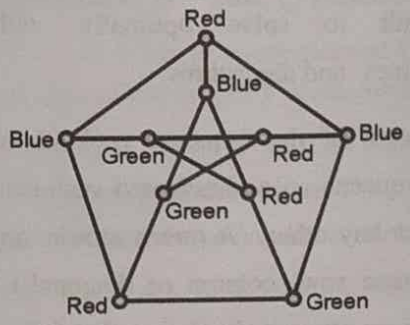


Fig. 3.8.4

A proper vertex coloring of the graph with 3 colors, the minimum number possible.

- **Hidato :** In Hidato, a grid of cells is given. It is usually square-shaped, like Sudoku or kakuno, but it can also include hexagons or any shape that forms a tessellation. It can have inner holes (like a disc), but it has to be made of only one piece.

  The goal is to fill the grid with a series of consecutive numbers adjacent to each other vertically, horizontally, or diagonally.

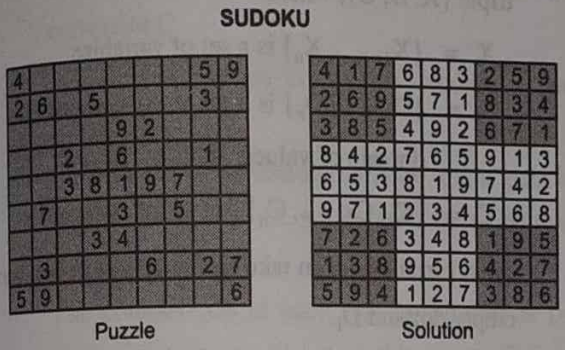


Fig. 3.8.5 : Sudoku

- **Latin square problem :** Here the task is to search the pattern which occurs several times in the game.

  Digits may get shuffled but contain the same digits.

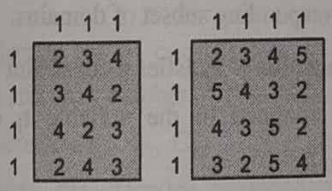


Fig. 3.8.6 : Latin square problem

3.8.5 Varieties of CSPs

Depending upon types an domain of the variables, we have different varieties of CSPs

(1) Discrete variable with finite domain e.g. Boolean CSP

(2) Continuous variables or continuous states describing one domain for one specific variable.

(3) Discrete state with infinite domain. Here we can have one state for multiple variables.

3.8.6 Varieties of Constraints

Depending upon the nature of variables, we have the following varieties of constraints :

(i) **Unary constraint :** It restricts the value of a single variable. It is the simplest type of constraint.

(ii) **Binary constraint :** It relates two variables. Generally a value lies in [0, 1].

(iii) **Global constraint :** It involves three or more variables; it is a multivariable constraint.

(iv) **Linear constraint :** Here each variable containing integer is in linear form. These type of constraints occur in linear programming.

(v) **Non-linear constraint :** Here each variable is in non-linear form. It may be in exponential or logarithmic form etc.

3.8.7 Commutativity

CSPs are all commutative. A problem is commutative if the order of application of any given set of actions has no effect on the outcome.

To solve CSPs efficiently :

(1) Function select-unassigned variable : which variable should be assigned next?

Function –order-domain values : in what order should its values be tried.

(2) Function 'inference' : what inferences should be performed at each step in the search?

(3) When the search arrives at an assignment that violates a constraint, can the search avoid repeating this failure?

3.8.8 Backtracking Efficiency

The efficiency of the backtracking algorithm depends on

(i) Reject returning true for candidates that are as close to the root as possible.

(ii) If reject always returns false, the algorithm will still find all solutions, but it will be equivalent to a brute-force search.

(iii) Which variable is to be assigned next.

(iv) To detect inevitable failure early.

(v) Backtracking is an important tool for solving constraint satisfaction problems, such as verbal arithmetic, Sudoku, crosswords and many other puzzles.

3.9 CONSTRAINT PROPAGATION

- Constraint Satisfaction Problems (CSPs) are mathematical questions.
- They are defined as a set of objects whose state must satisfy a number of **constraints** or **limitations**.
- CSPs represent entities in a problem, they are homogeneous collection of finite constraints over **variables**, which is solved by **constraint satisfaction** methods.
- The regularity in the formation of CSPs provides a common basis to analyse and solve problems of many unrelated families in both artificial intelligence and operations research.
- There is high complexity in CSPs. They require a combination of heuristics and combinatorial search methods to solve problems in a reasonable time.
- Constraint Programming (CP) specifically focuses on tackling these kinds of problems.
- Boolean Satisfiability Problem (SAT), the Satisfiability Modulo Theories (SMT), Mixed Integer Programming (MIP) and answer set programming (ASP) are all fields of research. They focus on the resolution of particular forms of the constraint satisfaction problem.

3.9.1 Mathematical Aspect of (CSP)

- A constraint satisfaction problem is defined as a triple {X, D, C}, where

  $X = \{X_1, ..., X_n\}$ is a set of variables.

  $D = \{D_1, ..., D_n\}$ is a set of their respective domain of values, and

  $C = \{C_1, C_2, ..., C_n\}$ is a set of constraints.

  Each variable X_i can take on the values in the non empty domain D_i.
- Every constraint $C_j \in C$ is in turn a pair (t_j, R_j) where $t_j \subset X$ is a subset of k variables and R_j is a k-avy relation on the corresponding subset of domain D_j.
- An evaluation of the variables is a function from a subset of variables to a particular set of values in the corresponding subset of domains.
- An evaluation V satisfies a constraint (t_j, R_j) if the values assigned to the variable t_j satisfies the relation R_j.
- An evaluation is consistent if it does not violate any of the constraints.
- An evaluation is complete if it includes all variables.
- An evaluation is a solution if it is consistent and complete; such an evaluation is said to solve the constraint satisfaction problem.

3.9.2 Inference in CSPs

- In constraint satisfaction, **constraint inference** is a relationship between constraints and their consequences.
- A set of constraints D entails a constraint C if every solution to D is also a solution to C.
- In other words, if V is a valuation of the variables in the constraints in D and all constraints in D are satisfied by V, then V also satisfies the constraint C.
- Some operations on constraints produce a new constraint.
- **Constraint composition** operates on a pair of binary constraints ((x, y), R) and ((y, z), S) with a common variable.
- The composition of such two constraints is the constraint ((x, z), Q). And this is satisfied by every equation of the two non-shared variables. And there exists a value of the shared variable y such that the evaluation of these three variables satisfies the two original constraints ((x, y), R) and ((y, z), S).

3.9.3 Extended Composition

It is similar to the composition, in principle. It assumes an arbitrary number of constraints.

Given constraints C_1, C_2, ..., C_m and a list A of their variables, the extended composition of them is the constraint (A, R) where an evaluation of A satisfies the constraint if it can be extended to the other variables so that C_1, C_2, ..., C_m are all satisfied.

3.9.4 Backtracking in CSPs

- A depth-first search that chooses values for one variable at a time and backtracks when a variable has no legal values left to assign.
- Backtracking algorithm repeatedly chooses an unassigned variable, and then tries all values in the domain of that variable in turn, trying to find a solution.
- Backtracking search, a form of depth-first search, is used for solving CSPs

3.9.5 Backtracking Search for CSPs

The name backtracking itself suggests that we are going back and coming forward. If it satisfies the condition then return success, else we go back again.

When we have multiple choices, then we make the decisions from the available choices. I the following cases, we need to use the backtracking algorithm :

(i) A piece of information required, is not available to make the best choice, so we use the backtracking strategy to try out all the possible solutions.

(ii) Each decision leads to a new set of choices, Then again, we backtrack to make new decisions. In this case, we need to use the backtracking strategy.

3.9.6 How Backtracking Works ?

Backtrackig is a systematic method of trying out various sequences of decisions until we find out that works.

Let us consider an example :

We begin with a start node. First, we move to node A. Since it is not a feasible solution. So we move to the next node, i.e. B. B is also not a feasible solution, and it is not a dead-end, so we **back-track** from node B to node A.

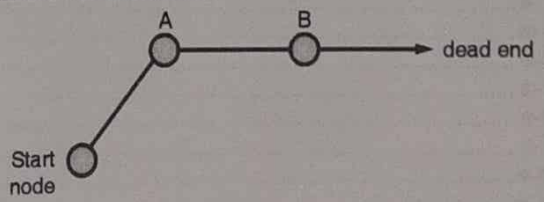


Fig. 3.9.1

Unit III End Sem

Suppose, another path exists from node A to node C. So, we move from node A to node C. It is also a dead end, so again we back-track from node C to node A.

We move from node A to the starting node.

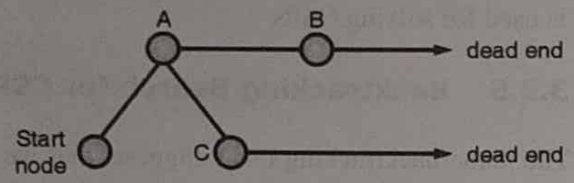


Fig. 3.9.2

Now we check any other path exists from the starting node. So we move from the start node to the node D. Since it is not a feasible solution so we move from node D to node E.

The node E is also not a feasible solution. It is a dead end so we backtrack from node E to node D.

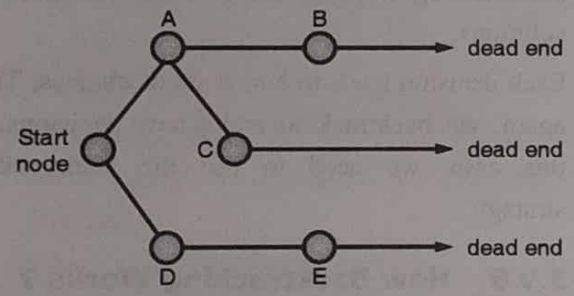


Fig. 3.9.3

Suppose another path exists from node D to node F. So we move from node D to node F. The node F is a **success node**.

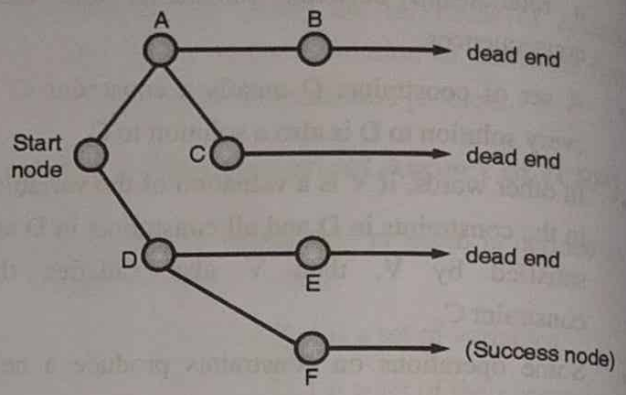


Fig. 3.9.4

Related terms are :

(i) **Live node :** The nodes that can be further generated are known as live nodes.

(ii) **E node :** The nodes whose children are being generated and become a success node.

(iii) **Success node :** The node is a success node if it provides a feasible solution.

(iv) **Dead node :** The node which cannot be further generated and also does not provide a feasible **solution is known as a dead-node.**

Chapter Ends...

❑❑❑