

R.M.K **GROUP OF** **ENGINEERING** **INSTITUTIONS**



R.M.K
GROUP OF
INSTITUTIONS

R.M.K GROUP OF INSTITUTIONS



R.M.K
GROUP OF
INSTITUTIONS



Please read this disclaimer before

This document is confidential and intended solely for the educational purpose of RMK Group of Educational Institutions. If you have received this document through email in error, please notify the system manager. This document contains proprietary information and is intended only to the respective group / learning community as intended. If you are not the addressee you should not disseminate, distribute or copy through e-mail. Please notify the sender immediately by e-mail if you have received this document by mistake and delete this document from your system. If you are not the intended recipient you are notified that disclosing, copying, distributing or taking any action in reliance on the contents of this information is strictly prohibited.

21AI401 ARTIFICIAL INTELLIGENCE

Department: C.S.E

Batch/Year:2021-2025/III

Created by: Dr . M. Arun Manicka Raja

Ms.V. Kalaipriya

Date: 15.08.2023

1.TABLE OF CONTENTS

S.NO.	CONTENTS	SLIDE NO.
1	CONTENTS	5
2	COURSE OBJECTIVES	7
3	PRE REQUISITES (COURSE NAMES WITH CODE)	8
4	SYLLABUS (WITH SUBJECT CODE, NAME, LTPC DETAILS)	9
5	COURSE OUTCOMES	10
6	CO- PO/PSO MAPPING	11
7	LECTURE PLAN –UNIT 1	13
8	ACTIVITY BASED LEARNING –UNIT 1	15
9	LECTURE NOTES – UNIT 1	18
10	ASSIGNMENT 1- UNIT 1	71
11	PART A Q & A (WITH K LEVEL AND CO) UNIT 1	72
12	PART B Q s (WITH K LEVEL AND CO) UNIT 1	80
13	SUPPORTIVE ONLINE CERTIFICATION COURSES UNIT 1	81
14	REAL TIME APPLICATIONS IN DAY TO DAY LIFE AND TO INDUSTRY UNIT 1	82

S.NO.	CONTENTS	SLIDE NO.
15	ASSESSMENT SCHEDULE	83
16	PRESCRIBED TEXT BOOKS & REFERENCE BOOKS	84
17	MINI PROJECT SUGGESTIONS	85



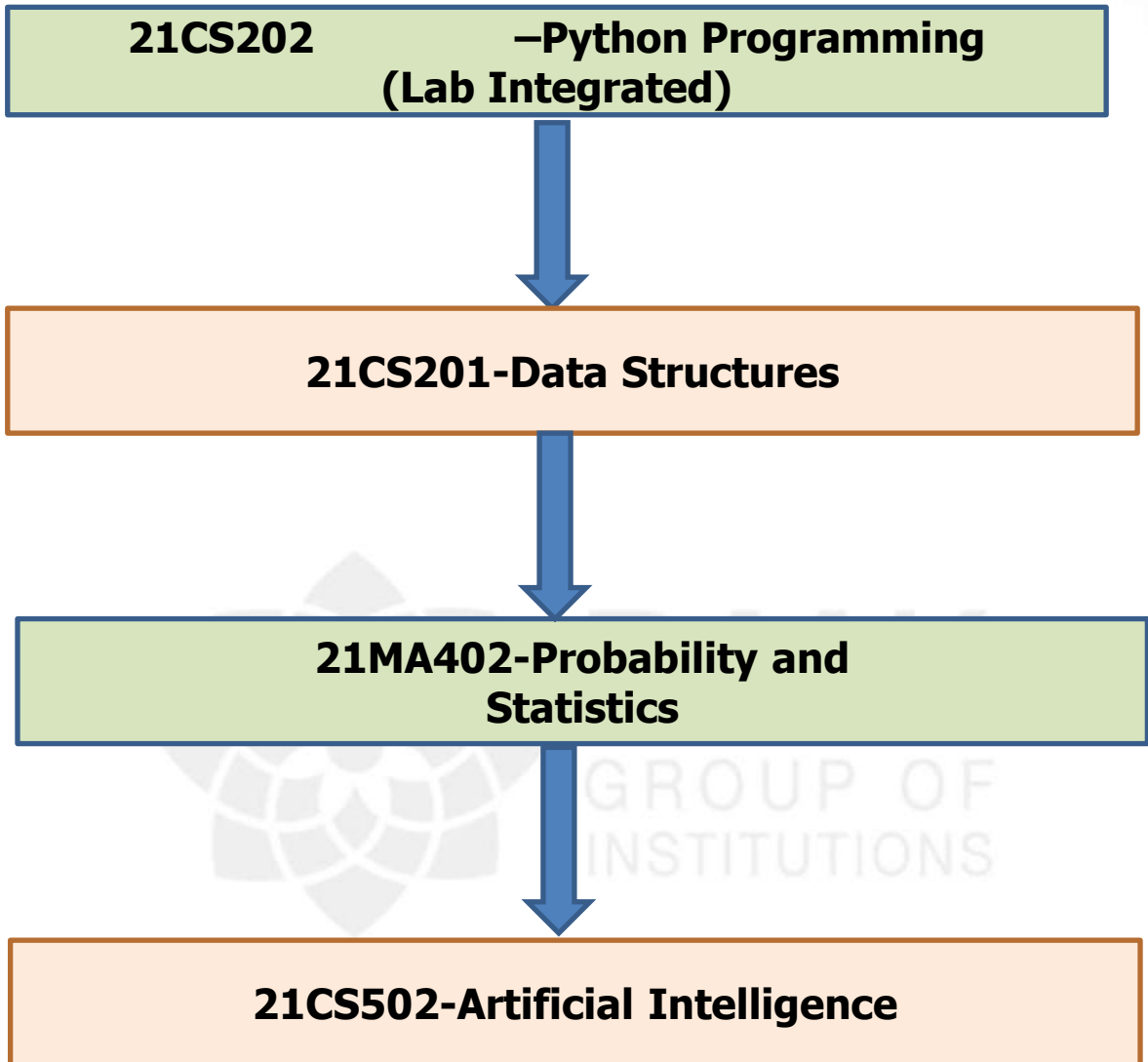
2. COURSE OBJECTIVES

- ❖ To Explain the Foundations of AI and various intelligent agents
- ❖ To discuss problem solving search strategies and game playing
- ❖ To describe logical agents and first-order logic
- ❖ To illustrate problem-solving strategies with knowledge representation mechanism for hard problems
- ❖ To Explain the basics of learning and expert systems



3. PRE REQUISITES

PRE-REQUISITE CHART



4.SYLLABUS

21AI401-ARTIFICIAL INTELLIGENCE

L T P C

3 0 0 3

Unit-I ARTIFICIAL INTELLIGENCE AND INTELLIGENT AGENTS 9

Introduction to AI–Foundations of Artificial Intelligence–Intelligent Agents–Agents and Environment–Concept of rationality – Nature of environments – Structure of agents – Problem Solving Agents–Example Problems – Search Algorithms – Uninformed Search Strategies

Unit II : PROBLEM SOLVING 9

Heuristics Search Strategies – Heuristic Functions - Game Play ing – Mini Max Algorithm- Optimal Decisions in Games – Alpha - Beta Search – Monte Carlo Search for Games - Constraint Satisfaction Problems – Constraint Propagation - Backtracking Search for CSP- Local Search for CSP-Structure of CSP

Unit III : LOGICAL AGENTS 9

Knowledge Based Agents-Logic-Propositional logic- Propositional theorem proving- Propositional model Checking- Agents based on propositional Logic-First Order Logic – Propositional Vs First Order Inference – Unification and First Order Inference – Forward Chaining- Backward Chaining – Resolution

Unit IV : KNOWLEDGE REPRESENTATION AND PLANNING 9

Ontological Engineering -Categories and Objects – Events - Mental Events and Mental Objects - Reasoning Systems for Categories - Reasoning with Default Information-Classical planning-Algorithms for Classical Planning- Heuristics for planning-Hierarchical planning - Non-Deterministic domains- Time, Schedule and resources-Analysis

Unit V : LEARNING AND EXPERT SYSTEMS 9

Forms of Learning AI applications – Developing Machine Learning Systems-Statistical Learning- Deep Learning: Simple Feed Forward network-Neural Networks-Reinforcement Learning : Learning from rewards-Passive and active Reinforcement learning . Expert Systems : Functions-Main structure – if-then rules for representing knowledge- developing the shel-Dealingg with uncertainty



5.COURSE OUTCOME

Course Code	Course Outcome Statement	Cognitive / Affective Level of the Course Outcome	Course Outcome
Course Outcome Statements in Cognitive Domain			
21AI401	Explain the foundations of AI and various Intelligent Agents	Apply K3	CO1
21AI401	Apply Search Strategies in Problem Solving and Game Playing	Apply K3	CO2
21AI401	Explain logical agents and First-order logic	Apply K3	CO3
21AI401	Apply problem-solving strategies with knowledge representation mechanism for solving hard problems	Apply K4	CO4
21AI401	Describe the basics of learning and expert systems.	Apply K4	CO5

6.CO-PO/PSO MAPPING

Course Outcomes (Cos)		Programme Outcomes (POs), Programme Specific Outcomes (PSOs)														
		PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12	PSO 1	PSO 2	PSO 3
21AI401.1	K 2	3	3	1	-	-	-	-	-	-	-	-	-	-	-	-
21AI401.2	K 3	3	2	1	-	-	-	-	-	-	-	-	-	-	-	-
21AI401.3	K 3	3	2	1	-	-	-	-	-	-	-	-	-	-	-	-
21AI401.4	K 3	3	3	2	-	-	-	-	-	-	-	-	-	-	-	-
21AI401.5	K 2	3	2	2	-	-	-	-	-	-	-	-	-	-	-	-

UNIT I ARTIFICIAL INTELLIGENCE AND INTELLIGENT AGENTS

LECTURE PLAN – UNIT I

UNIT I INTRODUCTION							
Sl. No	TOPIC	NO OF PERIODS	PROPOSED LECTURE	ACTUAL LECTURE	PERTAINING CO(s)	TAXONOMY LEVEL	MODE OF DELIVERY
			PERIOD	PERIOD			
1	Introduction to AI	1	07.08.2023		CO1	K2	MD1, MD5
2	Foundations of Artificial Intelligence	1	08.08.2023		CO1	K1	MD1, MD5
3	Intelligent Agents, agents and Environment	1	09.08.2023		CO1	K2	MD1, MD5
4	Concept of Rationality	1	10.08.2023		CO1	K2	MD1, MD5
5	Nature of Environment, Structure of Agents	1	11.08.2023		CO1	K2	MD1, MD5
6	Problem Solving Agents	1	12.08.2023		CO1	K2	MD1, MD5
7	Problem Solving Agents -Example Problems	1	14.08.2023		CO1	K5	MD1, MD5
8	Search Algorithms	1	16.08.2023		CO1	K2	MD1, MD5
9	Uninformed Search Strategies	1	17.08.2023		CO1	K5	MD1, MD5

LECTURE PLAN – UNIT I

❁ ASSESSMENT COMPONENTS

- ❁ AC 1. Unit Test
- ❁ AC 2. Assignment
- ❁ AC 3. Course Seminar
- ❁ AC 4. Course Quiz
- ❁ AC 5. Case Study
- ❁ AC 6. Record Work
- ❁ AC 7. Lab / Mini Project
- ❁ AC 8. Lab Model Exam
- ❁ AC 9. Project Review

MODE OF DELEIVERY

- MD 1. Oral presentation
- MD 2. Tutorial
- MD 3. Seminar
- MD 4 Hands On
- MD 5. Videos
- MD 6. Field Visit



R.M.K.
GROUP OF
INSTITUTIONS

8. ACTIVITY BASED LEARNING : UNIT – I

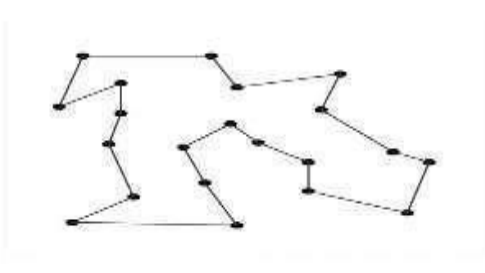
ACTIVITY 1: Travelling Sales Person

The "**Traveling Salesman Problem**" (**TSP**) is a common problem applied to **artificial intelligence**. The **TSP** presents the computer with a number of cities, and the computer must compute the optimal path between the cities. This applet uses a genetic algorithm to produce a solution to the "**Traveling Salesman Problem**".

It is the challenge of finding the shortest yet most efficient route for a person to take given a list of specific destinations. It is a well-known algorithmic problem in the fields of computer science and operations research.

There are obviously a lot of different routes to choose from, but finding the best one—the one that will require the least distance or cost—is what mathematicians and computer scientists have spent decades trying to solve for

TSP has commanded so much attention because it's so easy to describe yet so difficult to solve. In fact, TSP belongs to the class of combinatorial optimization problems known as NP-complete. This means that TSP is classified as NP-hard because it has no "quick" solution and the complexity of calculating the best route will increase when you add more destinations to the problem.



8. ACTIVITY BASED LEARNING : UNIT – I

ACTIVITY 2 : How to automatically deskew (straighten) a text image using OpenCV

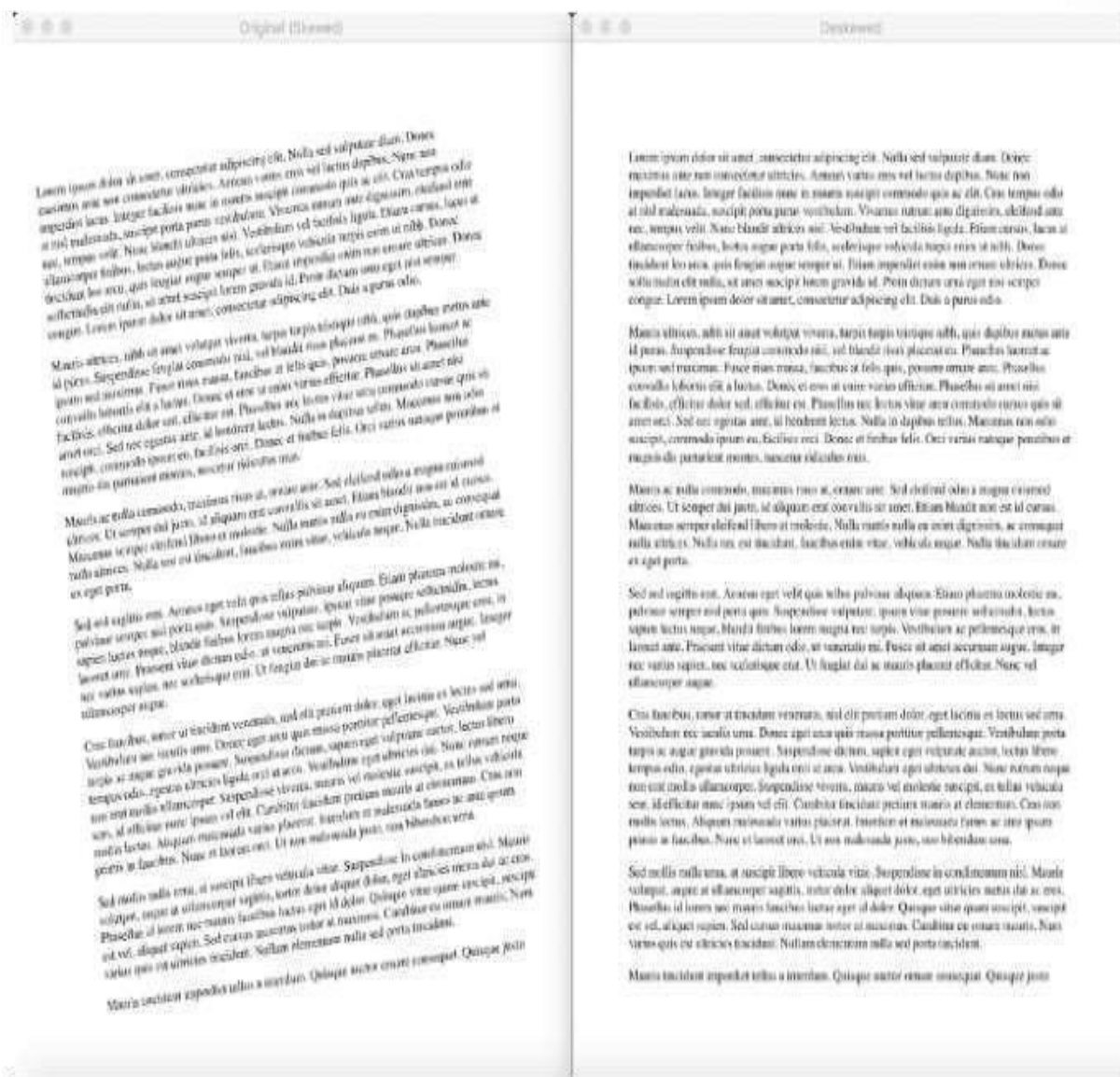
Today a simple solution to image deskewing problem (straightening a rotated image). If people working on anything that has text extraction from images — they have to deal with image deskewing in one form or another. From camera pictures to scanned documents — deskewing is a mandatory step in image pre-processing before feeding the cleaned-up image to an OCR tool.

Deskewing algorithm

- ✿ Let's start by discussing the general idea of deskewing algorithm. Our main goal will be splitting the rotated image into text blocks, and determining the angle from them. To give you a detailed break-down of the approach that I'll use:
- ✿ Per usual — convert the image to gray scale.
- ✿ Apply slight blurring to decrease noise in the image.
- ✿ Now our goal is to find areas with text, i.e. text blocks of the image. To make text block detection easier we will invert and maximize the colors of our image, that will be achieved via thresholding. So now text becomes white (exactly 255,255,255 white), and background is black (same deal 0,0,0 black).
- ✿ To find text blocks we need to merge all printed characters of the block. We achieve this via dilation (expansion of white pixels). With a larger kernel on X axis to get rid of all spaces between words, and a smaller kernel on Y axis to blend in lines of one block between each other, but keep larger spaces between text blocks intact.
- ✿ Now a simple contour detection with min area rectangle enclosing our contour will form all the text blocks that we need.

There can be various approaches to determine skew angle, but we'll stick to the simple one — take the largest text block and use its angle.

8. ACTIVITY BASED LEARNING : UNIT – I



Original, skewed image (on the left) compared to deskewed result (on the right)

9. LECTURE NOTES : UNIT – I

INTRODUCTION

Syllabus:

Introduction to AI–Foundations of Artificial Intelligence-Intelligent Agents-Agents and Environment-Concept of rationality – Nature of environments – Structure of agents – Problem Solving Agents–Example Problems – Search Algorithms – Uninformed Search Strategies

1. INTRODUCTION–DEFINITION

1. What is Artificial Intelligence?

- ✿ According to the father of Artificial Intelligence, John McCarthy, it is “The science and engineering of making intelligent machines, especially intelligent computer programs”.
- ✿ Artificial Intelligence is a way of **making a computer, a computer-controlled robot, or a software think intelligently**, in the similar manner the intelligent humans think.
- ✿ AI is accomplished by studying how human brain thinks, and how humans learn, decide, and work while trying to solve a problem, and then using the outcomes of this study as a basis of developing intelligent software and systems. The definitions are concerned with thought processes and reasoning.
- ✿ In the real world, the knowledge has some unwelcomed properties –
- ✿ Its volume is huge, next to unimaginable.
- ✿ It is not well-organized or well-formatted.
- ✿ It keeps changing constantly.

Artificial intelligence is a science and technology based on disciplines such as Computer Science, Biology, Psychology, Linguistics, Mathematics, and Engineering. A major thrust of AI is in the development of computer functions associated with human intelligence, such as reasoning, learning, and problem solving.

Programming Without AI	Programming With AI
A computer program without AI can answer the specific questions it is meant to solve.	A computer program with AI can answer the generic questions it is meant to solve.
Modification in the program leads to change in its structure.	AI programs can absorb new modifications by putting highly independent pieces of information together. Hence you can modify even a minute piece of information of program without affecting its structure.
Modification is not quick and easy. It may lead to affecting the program adversely.	Quick and Easy program modification.

1.1.2 The Foundations of Artificial Intelligence

(i) Philosophy:

Thomas Hobbes (1588–1679) proposed that reasoning was like numerical computation, that “we add and subtract in our silent thoughts.” The first known calculating machine was constructed around 1623 by the German scientist Wilhelm Schickard (1592–1635) is more famous. Pascal wrote that “the arithmetical machine produces effects which appear nearer to thought than all the actions of animals.” Thomas Hobbes suggested the idea of an “artificial animal,” arguing “For what is the heart but a spring; and the nerves, but so many strings; and the joints, but so many wheels.” It’s one thing to say that the mind operates, at least in part, according to logical rules, and to build physical systems that emulate some of those rules; it’s another to say that the mind itself is such a physical system.

(ii) Mathematics:

Philosophers staked out some of the fundamental ideas of AI, but the leap to a formal science required a level of mathematical formalization in three fundamental areas: logic, computation, and probability.

The idea of formal logic can be traced back to the philosophers of ancient Greece, but its mathematical development really began with the work of George Boole (1815–1864), who worked out the details of propositional, or Boolean, **logic** (Boole, 1847). In 1879, Gottlob Frege extended Boole's logic to include objects and a relation, creating the first-order logic that is used today.

The first nontrivial algorithm is thought to be Euclid's algorithm for computing greatest common divisors. Some fundamental results can also be interpreted as showing that some functions on the integers cannot be represented by an algorithm—that is, they cannot be computed. This motivated Alan Turing (1912–1954) to try to characterize exactly which functions are **computable**—capable of being computed. This notion is actually slightly problematic because the notion of a computation or effective procedure really cannot be given a formal definition.

Although decidability and computability are important to an understanding of computation, the notion of tractability has had an even greater impact. Besides logic and computation, the third great contribution of mathematics to AI is the theory of probability. The Italian Gerolamo Cardano (1501–1576) first framed the idea of **probability**, describing it in terms of the possible outcomes of gambling events.

(iii) Economics:

While the ancient Greeks and others had made contributions to economic thought, Smith was the first to treat it as a science, using the idea that economies can be thought of as consisting of individual agents maximizing their own economic well-being. Decision theory, which combines probability theory with utility theory, provides a formal and complete framework for decisions (economic or otherwise) made under uncertainty—that is, in cases where probabilistic descriptions appropriately capture the decision maker's environment.

Von Neumann and Morgenstern's development of **game theory** (see also Luce and Raiffa, 1957) included the surprising result that, for some games a rational agent should adopt policies that are (or least appear to be) randomized.

The field of **operations research**, which emerged in World War II from efforts in Britain to optimize radar installations, and later found civilian applications in complex management decisions. The work of Richard Bellman (1957) formalized a class of sequential decision problems called **Markov decision processes**. Work in economics and operations research has contributed much to our notion of rational agents, yet for many years AI research developed along entirely separate paths.

(iv)Neuroscience:

Neuroscience is the study of the nervous system, particularly the brain.

It has also long been known that human brains are somehow different; in about 335 B.C. Aristotle wrote, "Of all the animals, man has the largest brain in proportion to his size." Paul Broca's (1824–1880) study of aphasia (speech deficit) in brain-damaged patients in 1861 demonstrated the existence of localized areas of the brain responsible for specific cognitive functions. By that time, it was known that NEURON the brain consisted of nerve cells, or neurons, but it was not until 1873 that Camilo Golgi (1843–1926) developed a staining technique allowing the observation of individual neurons in the brain.

Nicolas Rashevsky (1936,1938) was the first to apply **mathematical models** to the study of the nervous system. The measurement of intact brain activity began in 1929 with the invention by Hans Berger of the electroencephalograph (**EEG**). The recent development of functional magnetic resonance imaging (**fMRI**) (Ogawa et al., 1990; Cabeza and Nyberg, 2001) is giving neuroscientists unprecedentedly detailed images of brain activity, enabling measurements that correspond in interesting ways to ongoing cognitive processes.

	Supercomputer	Personal Computer	Human Brain
Computational units	10^4 CPUs, 10^{12} transistors	4 CPUs, 10^9 transistors	10^{11} neurons
Storage units	10^{14} bits RAM 10^{15} bits disk	10^{11} bits RAM 10^{13} bits disk	10^{11} neurons 10^{14} synapses
Cycle time	10^{-9} sec	10^{-9} sec	10^{-3} sec
Operations/sec	10^{15}	10^{10}	10^{17}
Memory updates/sec	10^{14}	10^{10}	10^{14}

Figure 1 A crude comparison of the raw computational resources available to the IBM BLUE GENE supercomputer

Brains and digital computers have somewhat different properties. Figure 1 shows that computers have a cycle time that is a million times faster than a brain. Truly amazing conclusion is that a collection of simple cells can lead to thought, action, and consciousness or, in the pithy words of John Searle (1992), brains cause minds. The only real alternative theory is mysticism: that minds operate in some mystical realm that is beyond physical science.

(v) Psychology (How do humans and animals think and act?):

The origins of scientific psychology are usually traced to the work of the German physicist Hermann von Helmholtz (1821–1894) and his student Wilhelm Wundt (1832–1920). In 1879, Wundt opened the first laboratory of experimental psychology, at the University of Leipzig. Wundt insisted on carefully controlled experiments in which his workers would perform a perceptual or associative task while introspecting on their thought processes. Biologists studying animal behavior, on the other hand, lacked introspective data and developed an objective methodology, as described by H. S. Jennings (1906) in his influential work *Behavior of the Lower Organisms*.

Applying this viewpoint to humans, the **behaviorism** movement, led by John Watson (1878–1958), rejected any theory involving mental processes on the grounds that introspection could not provide reliable evidence. Behaviorists insisted on studying only objective measures of the percepts (or stimulus) given to an animal and its resulting actions (or response).

Cognitive psychology, which views COGNITIVE the brain as an information-processing device, can be traced back at least to the works of William James (1842–1910). Craik specified the three key steps of a knowledge-based agent: (1) the stimulus must be translated into an internal representation, (2) the representation is manipulated by cognitive processes to derive new internal representations, and (3) these are in turn retranslated back into action.

(vi) Computer engineering:

For artificial intelligence to succeed, we need two things: intelligence and an artifact. The computer has been the artifact of choice. The modern digital electronic computer was invented independently and almost simultaneously by scientists in three countries embattled in World War II.

The first operational computer was the electromechanical Heath Robinson,⁸ built in 1940 by Alan Turing's team for a single purpose: deciphering German messages. The first operational programmable computer was the Z-3, the invention of Konrad Zuse in Germany in 1941. The first electronic computer, the ABC, was assembled by John Atanasoff and his student Clifford Berry between 1940 and 1942 at Iowa State University. The first programmable machine was a loom, devised in 1805 by Joseph Marie Jacquard (1752–1834), that used punched cards to store instructions for the pattern to be woven.

AI also owes a debt to the software side of computer science, which has supplied the operating systems, programming languages, and tools needed to write modern programs (and papers about them).

(vii) Control theory and cybernetics (How can artifacts operate under their own control?):

Ktesibios of Alexandria (c. 250 B.C.) built the first self-controlling machine: a water clock with a regulator that maintained a constant flow rate. The mathematical theory of stable feedback systems was developed in the 19th century. Wiener's book **Cybernetics** (1948) became a bestseller and awoke the public to the possibility of artificially intelligent machines.

The central figure in the creation of what is now called **control theory** was Norbert Wiener (1894–1964). Wiener was a brilliant mathematician, who worked with Bertrand Russel,

among others, before developing an interest in biological and mechanical control systems and their connection to cognition.

Modern control theory, especially the branch known as stochastic optimal control, has as its goal the design of systems that maximize an **objective function** over time. This roughly matches our view of AI: designing systems that behave optimally. Calculus and matrix algebra, the tools of control theory, lend themselves to systems that are describable by fixed sets of continuous variables, whereas AI was founded in part as a way to escape from these perceived limitations.

(viii) Linguistics (How does language relate to thought?):

Modern linguistics and AI, then, were “born” at about the same time, and grew up together, intersecting in a hybrid field called **computational linguistics** or **natural language processing**.

The problem of understanding language soon turned out to be considerably more complex than it seemed in 1957. Understanding language requires an understanding of the subject matter and context, not just an understanding of the structure of sentences. This might seem obvious, but it was not widely appreciated until the 1960s. Much of the early work in **knowledge representation** (the study of how to put knowledge into a form that a computer can reason with) was tied to language and informed by research in linguistics, which was connected in turn to decades of work on the philosophical analysis of language.

1.3 What is AI Technique?

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.

<p>Thinking Humanly</p> <p>“The exciting new effort to make computers think ... <i>machines with minds</i>, in the full and literal sense.” (Haugeland, 1985)</p> <p>“[The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning ...” (Bellman, 1978)</p>	<p>Thinking Rationally</p> <p>“The study of mental faculties through the use of computational models.” (Charniak and McDermott, 1985)</p> <p>“The study of the computations that make it possible to perceive, reason, and act.” (Winston, 1992)</p>
<p>Acting Humanly</p> <p>“The art of creating machines that perform functions that require intelligence when performed by people.” (Kurzweil, 1990)</p> <p>“The study of how to make computers do things at which, at the moment, people are better.” (Rich and Knight, 1991)</p>	<p>Acting Rationally</p> <p>“Computational Intelligence is the study of the design of intelligent agents.” (Poole <i>et al.</i>, 1998)</p> <p>“AI ... is concerned with intelligent behavior in artifacts.” (Nilsson, 1998)</p>

Figure 2 Some definitions of artificial intelligence, organized into four categories

These definitions are concerned with behavior. The definitions on the left measure success in terms of fidelity to human performance, whereas RATIONALITY the ones on the right measure against an ideal performance measure, called rationality. A system is rational if it does the “right thing,” given what it knows. The four approaches in more detail are as follows:

(a) Acting humanly: The Turing Test approach

The Turing Test, proposed by Alan Turing (1950), was designed to provide a satisfactory operational definition of intelligence. A computer passes the test if a human interrogator, after posing some written questions, cannot tell whether the written responses come from a person or from a computer.

The computer would need to possess the following capabilities:

- ✿ **Natural language processing** to enable it to communicate successfully in English.
- ✿ **Knowledge representation** to store what it knows or hears;
- ✿ **Automated Reasoning** to use the stored information to answer questions and to draw new conclusions;
- ✿ **Machine Learning** to adapt to new circumstances and to detect and extrapolate patterns.
- ✿ **Computer Vision** to perceive objects
- ✿ **Robotics** to manipulate objects and move about.

Total Turing Test includes a video signal so that the interrogator can test the subject's perceptual abilities, as well as the opportunity for the interrogator to pass physical objects "through the hatch." To pass the total Turing Test, the computer will need These six disciplines compose most of AI, and Turing deserves credit for designing a test that remains relevant 60 years later. Yet AI researchers have devoted little effort to passing the Turing Test, believing that it is more important to study the underlying principles of intelligence than to duplicate an exemplar.

(b) Thinking humanly: The cognitive modeling approach

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

Through introspection—trying to catch our own thoughts as they go by;

Through psychological experiments—observing a person in action;

Through brain imaging—observing the brain in action.

Allen Newell and Herbert Simon, who developed GPS, the “General Problem Solver” tried to trace the reasoning steps to traces of human subjects solving the same problems. The interdisciplinary field of cognitive science brings together computer models from AI and experimental techniques from psychology to construct precise and testable theories of the human mind. Cognitive science is a fascinating field in itself, worthy of several textbooks and at least one encyclopedia (Wilson and Keil, 1999). Real cognitive science, however, is necessarily based on experimental investigation of actual humans or animals.

Both AI and cognitive science are developing more rapidly. The two fields continue to fertilize each other, most notably in computer vision, which incorporates neurophysiologic evidence into computational models.

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

The Greek philosopher Aristotle was one of the first to attempt to codify “right thinking” that is irrefutable reasoning processes. His syllogism provided patterns for argument structures that always yielded correct conclusions when given correct premises—for example, “Socrates is a man; all men are mortal; therefore Socrates is mortal”. These laws of thought were supposed to govern the operation of the mind; their study initiated a field called logic.

There are two main obstacles to this approach. First, it is not easy to take informal knowledge and state it in the formal terms required by logical notation, particularly when the knowledge is less than 100% certain. Second, there is a big difference between solving a problem “in principle” and solving it in practice.

(d) Acting rationally: The rational agent approach

An agent is something that acts. Computer agents are not mere programs, but they are expected to have the following attributes also: (a) operating under autonomous control, (b) perceiving their environment, (c) persisting over a prolonged time period, (d) adapting to change (e) create (f) pursue goals. A rational agent is one that acts so as to achieve the best outcome or, when there is uncertainty, the best expected outcome.

The rational-agent approach has two advantages over the other approaches. First, it is more general than the “laws of thought” approach because correct inference is just one of several possible mechanisms for achieving rationality. Second, it is more amenable to scientific development than are approaches based on human behavior or human thought.

In the “laws of thought” approach to AI, the emphasis was on correct inferences. Making correct inferences is sometimes part of being a rational agent, because one way to act rationally is to reason logically to the conclusion that a given action will achieve one’s goals and then to act on that conclusion.

2. Intelligent Agents

1. Agents and Environments

An **agent** is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators. Figure 3 shows Agents interact with environments through sensors and actuators.

A **human agent** has eyes, ears, and other organs for sensors and hands, legs, vocal tract, and so on for actuators. A **robotic agent** might have cameras and infrared range finders for sensors and various motors for actuators. A **software agent** receives keystrokes, file contents, and network packets as sensory inputs and acts on the environment by displaying on the screen, writing files, and sending network packets.

We use the term percept to refer to the agent’s perceptual inputs at any given instant. An agent’s percept sequence is the complete history of everything the agent has ever perceived.

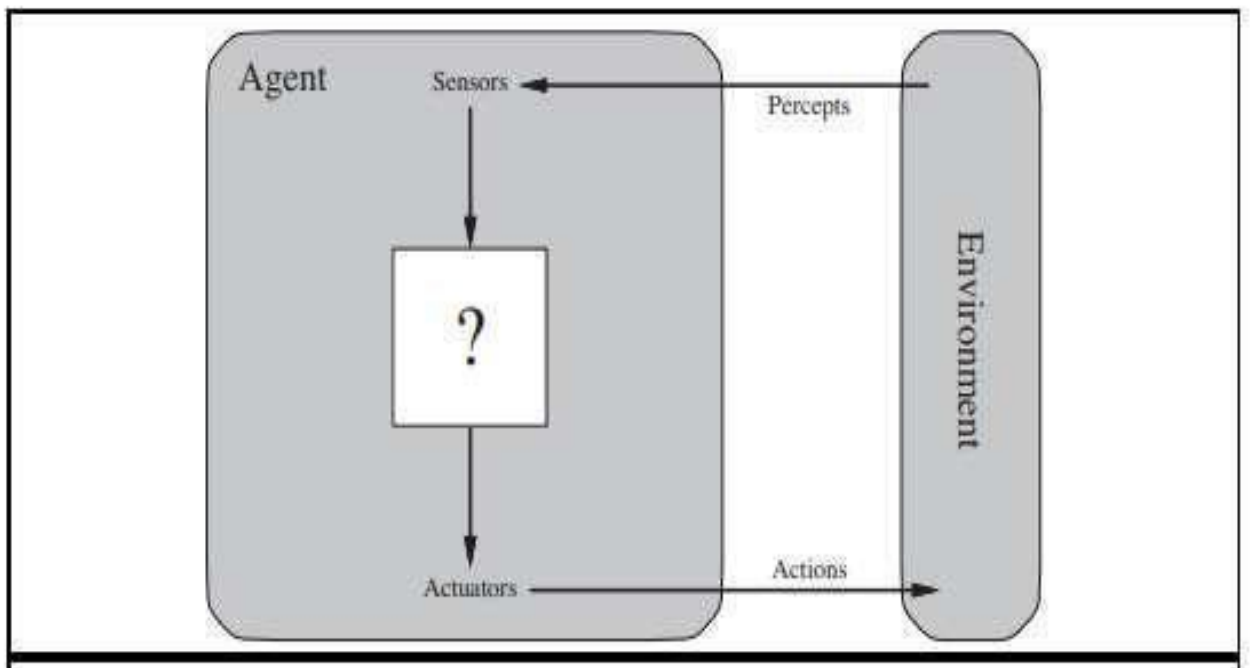


Figure 3. Agents interact with environments through sensors and actuators.

To illustrate these ideas, we use a very simple example—the vacuum-cleaner world shown in Figure 4. This world is so simple that we can describe everything that happens; it's also a made-up world, so we can invent many variations. This particular 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 choose to move left, move right, suck up the dirt, or do nothing. One very simple agent function is the following: if the current square is dirty, then suck; otherwise, move to the other square.

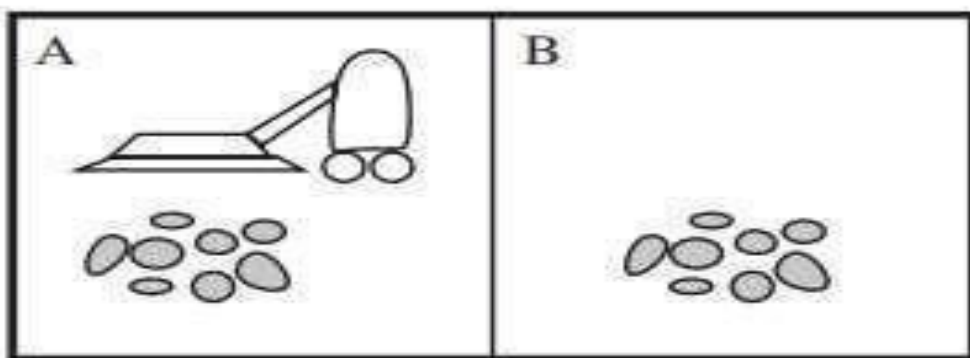


Figure 4.A vacuum-cleaner world with just two locations

We should emphasize that the notion of an agent is meant to be a tool for analyzing systems, not an absolute characterization that divides the world into agents and non-agents. One could view a hand-held calculator as an agent that chooses the action of displaying “4” when given the percept sequence “2 + 2 =,” but such an analysis would hardly aid our understanding of the calculator.

1.2.2 Good Behavior: The Concept of Rationality

A **rational agent** is one that does the right thing—conceptually speaking, every entry in the table for the agent function is filled out correctly. When an agent is plunked down in an environment, it generates a sequence of actions according to the percepts it receives. This sequence of actions causes the environment to go through a sequence of states. If the sequence is desirable, then the agent has performed well. This notion of desirability is captured by a performance measure that evaluates any given sequence of environment states.

Rationality:

The rational at any given time depends on four things:

- The performance measure that defines the criterion of success.
- The agent’s prior knowledge of the environment.
- The actions that the agent can perform.
- The agent’s percept sequence to date.

The definition of a rational agent is for each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has.

Omniscience, learning, and autonomy:

An omniscient agent knows the actual outcome of its actions and can act accordingly; but omniscience is impossible in reality. Rationality maximizes expected performance, while perfection maximizes actual performance. Doing actions in order to modify future percepts—sometimes called information gathering—is an important part of rationality.

A second example of information gathering is provided by the exploration that must be undertaken by a vacuum-cleaning agent in an initially unknown environment. The agent's initial configuration could reflect some prior knowledge of the environment, but as the agent gains experience this may be modified and augmented. There are extreme cases in which the environment is completely known a priori. In such cases, the agent need not perceive or learn; it simply acts correctly. To the extent that an agent relies on the prior knowledge of its designer rather than on its own percepts, we say that the agent lacks autonomy. A rational agent should be autonomous—it should learn what it can to compensate for partial or incorrect prior knowledge.

1.2.3 The Nature of Environments

In designing an agent, the first step must always be to specify the task environment as fully as possible. For the acronymically minded, we call PEAS this the PEAS (Performance, Environment, Actuators, Sensors) description. Figure 5 summarizes the PEAS description for the taxi's task environment.

First, what is the performance measure to which we would like our automated driver to aspire? Desirable qualities include getting to the correct destination; minimizing fuel consumption and wear and tear; minimizing the trip time or cost; minimizing violations of traffic laws and disturbances to other drivers; maximizing safety and passenger comfort; maximizing profits. Obviously, some of these goals conflict, so tradeoffs will be required.

Next, what is the driving environment that the taxi will face? Any taxi driver must deal with a variety of roads, ranging from rural lanes and urban alleys to 12-lane freeways. The actuators for an automated taxi include those available to a human driver: control over the engine through the accelerator and control over steering and braking. In addition, it will need output to a display screen or voice synthesizer to talk back to the passengers, and perhaps some way to communicate with other vehicles, politely or otherwise.

In contrast, some software agents (or software robots or softbots) exist in rich, unlimited domains. Imagine a softbot Web site operator designed to scan Internet news sources and show the interesting items to its users, while selling advertising space to generate revenue.

To do well, that operator will need some natural language processing abilities, it will need to learn what each user and advertiser is interested in, and it will need to change its plans dynamically—for example, when the connection for one news source goes down or when a new one comes online

Agent Type	Performance Measure	Environment	Actuators	Sensors
Taxi driver	Safe, fast, legal, comfortable trip, maximize profits	Roads, other traffic, pedestrians, customers	Steering, accelerator, brake, signal, horn, display	Cameras, sonar, speedometer, GPS, odometer, accelerometer, engine sensors, keyboard

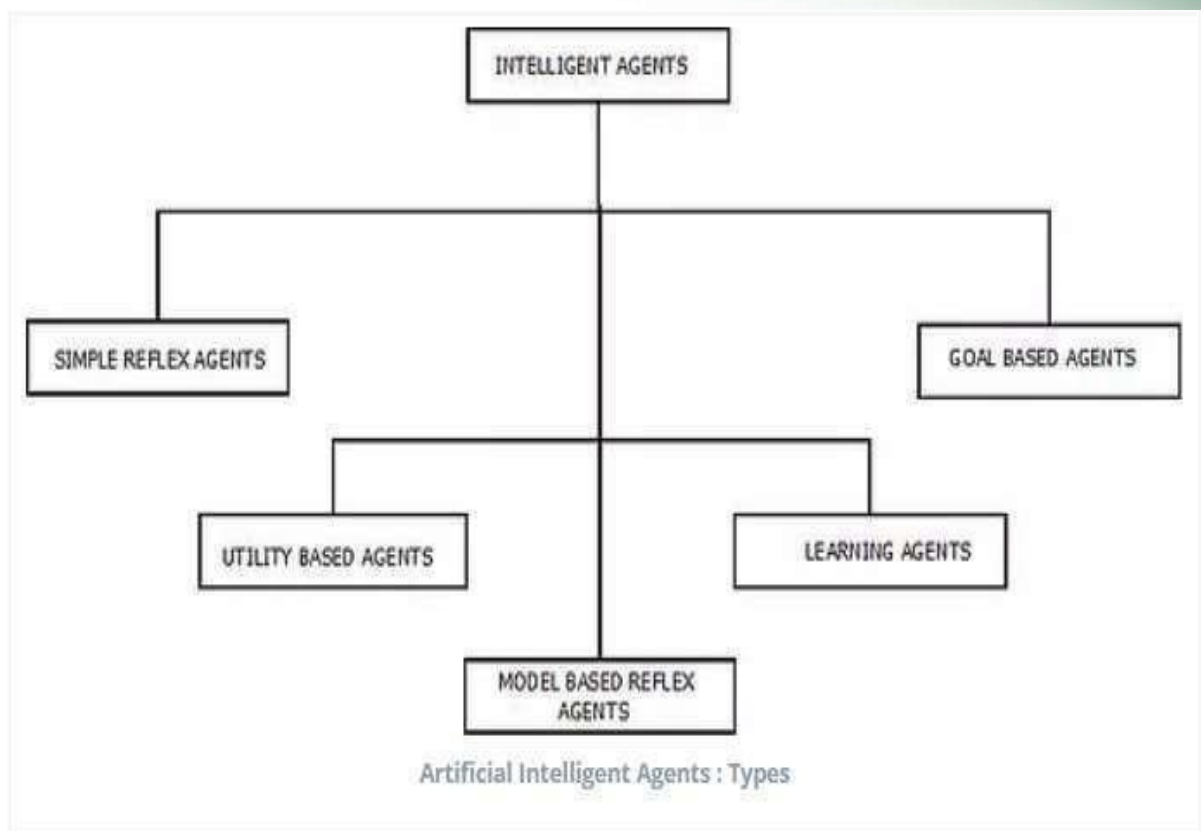
Figure 5 summarizes the PEAS description for the taxi’s task environment.

1.2.4 Structure of Intelligent Agents

Agent’s structure can be viewed as – Agent = Architecture + Agent Program

Architecture = the machinery that an agent executes on.

Agent Program = an implementation of an agent function.



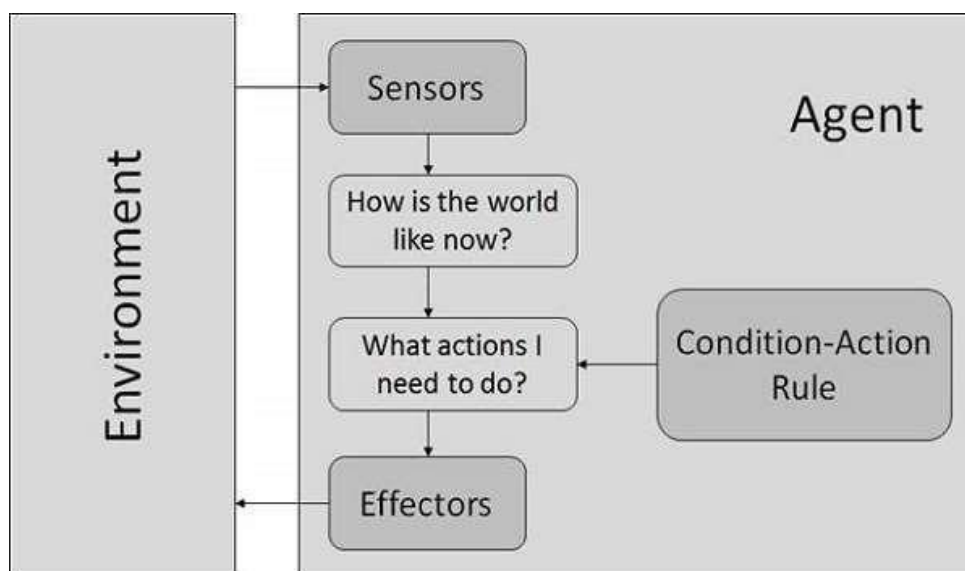
Simple Reflex Agents

They choose actions only based on the current percept.

They are rational only if a correct decision is made only on the basis of current percept.

Their environment is completely observable.

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



Model Based Reflex Agents

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

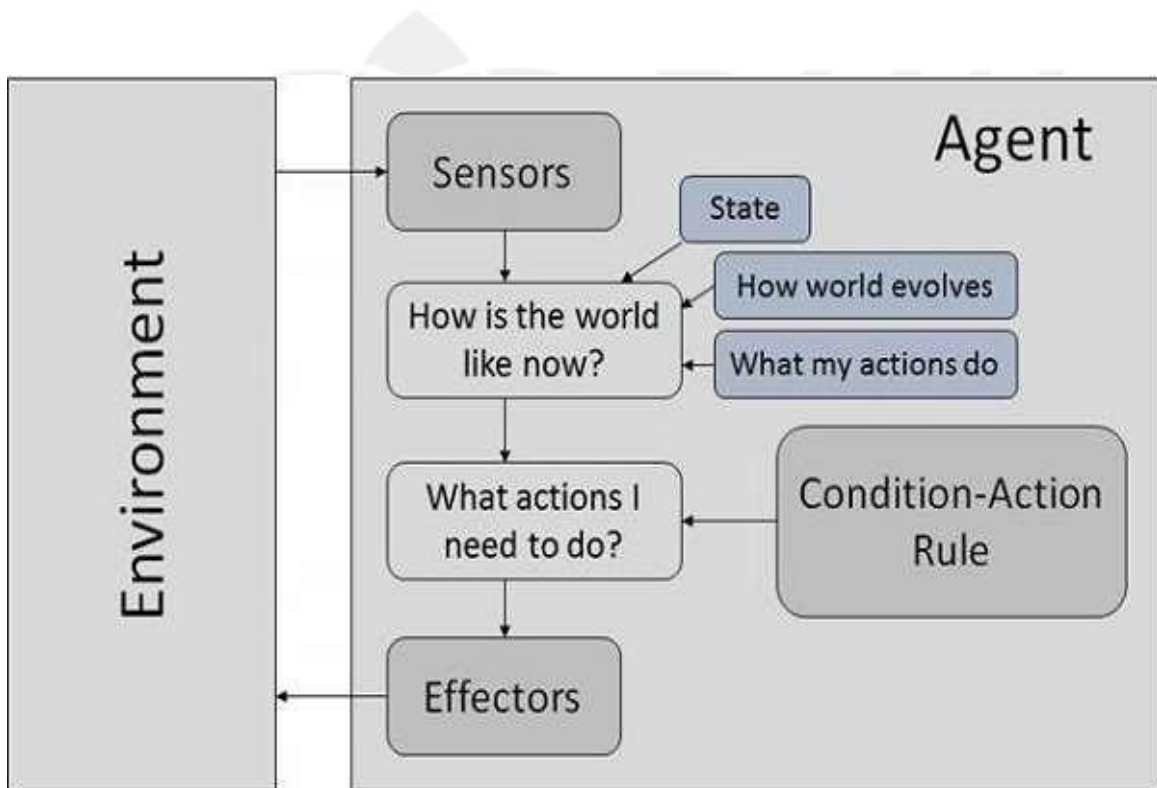
Model – knowledge about “how the things happen in the world”.

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

Updating the state requires the information about –

How the world evolves.

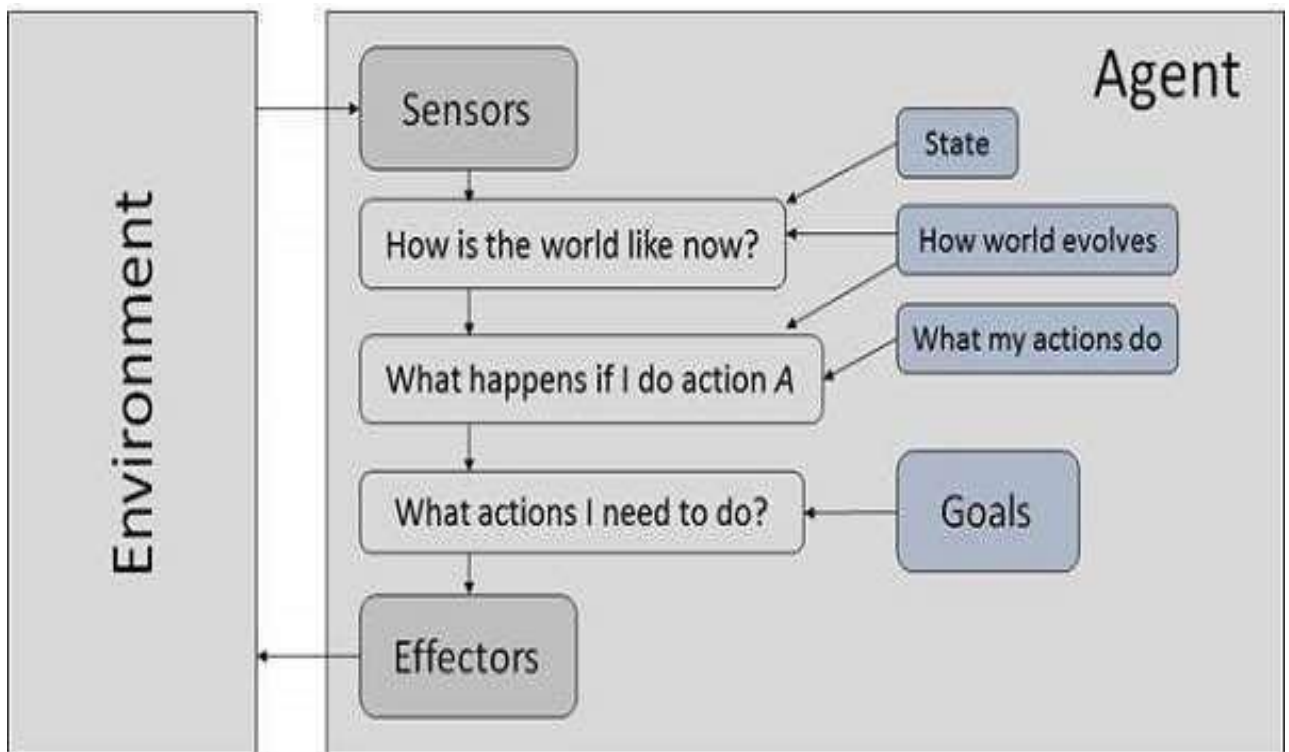
How the agent’s actions affect the world.



Goal Based Agents

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

Goal – It is the description of desirable situations.



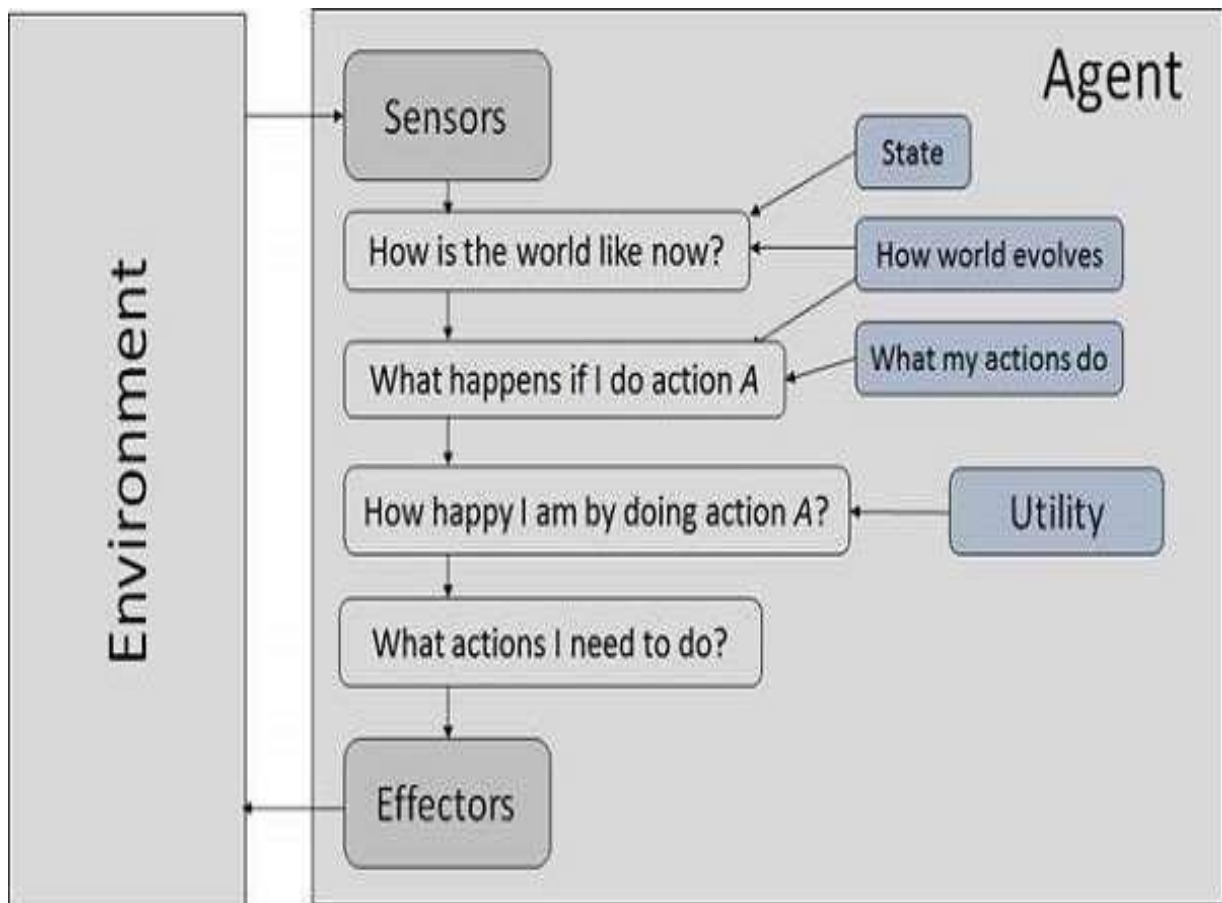
Utility Based Agents

They choose actions based on a preference (utility) for each state.

Goals are inadequate when –

There are conflicting goals, out of which only few can be achieved.

Goals have some uncertainty of being achieved and you need to weigh likelihood of success against the importance of a goal.



EXAMPLE: The Nature of Environments

Some programs operate in the entirely **artificial environment** confined to keyboard input, database, computer file systems and character output on a screen. In contrast, some software agents (software robots or softbots) exist in rich, unlimited softbots domains. The simulator has a **very detailed, complex environment**. The software agent needs to choose from a long array of actions in real time. A softbot designed to scan the online preferences of the customer and show interesting items to the customer works in the **real** as well as an **artificial** environment.

The most famous **artificial environment** is the **Turing Test environment**, in which one real and other artificial agents are tested on equal ground. This is a very challenging environment as it is highly difficult for a software agent to perform as well as a human.

Turing Test

The success of an intelligent behavior of a system can be measured with Turing Test.

Two persons and a machine to be evaluated participate in the test. Out of the two persons, one plays the role of the tester. Each of them sits in different rooms. The tester is unaware of who is machine and who is a human. He interrogates the questions by typing and sending them to both intelligences, to which he receives typed responses.

This test aims at fooling the tester. If the tester fails to determine machine's response from the human response, then the machine is said to be intelligent.

Properties of Environment

The environment has multifold properties –

- ✿ **Discrete / Continuous** – If there are a limited number of distinct, clearly defined, states of the environment, the environment is discrete (For example, chess); otherwise it is continuous (For example, driving).
- ✿ **Observable / Partially Observable** – If it is possible to determine the complete state of the environment at each time point from the percepts it is observable; otherwise it is only partially observable.
- ✿ **Static / Dynamic** – If the environment does not change while an agent is acting, then it is static; otherwise it is dynamic.
- ✿ **Single agent / Multiple agents** – The environment may contain other agents which may be of the same or different kind as that of the agent.
- ✿ **Accessible / Inaccessible** – If the agent's sensory apparatus can have access to the complete state of the environment, then the environment is accessible to that agent.
- ✿ **Deterministic / Non-deterministic** – If the next state of the environment is completely determined by the current state and the actions of the agent, then the environment is deterministic; otherwise it is non-deterministic.
- ✿ **Episodic / Non-episodic** – In an episodic environment, each episode consists of the agent perceiving and then acting. The quality of its action depends just on the episode itself. Subsequent episodes do not depend on the actions in the previous episodes. Episodic environments are much simpler because the agent does not need to think ahead.

1.3 PROBLEM SOLVING APPROACH TO TYPICAL AI PROBLEMS

Problem Solving in games such as “Sudoku” can be an example. It can be done by building an artificially intelligent system to solve that particular problem. To do this, one needs to define the problem statements first and then generating the solution by keeping the conditions in mind.

Problem Searching

In general, searching refers to as finding information one needs.

Searching is the most commonly used technique of problem solving in artificial intelligence.

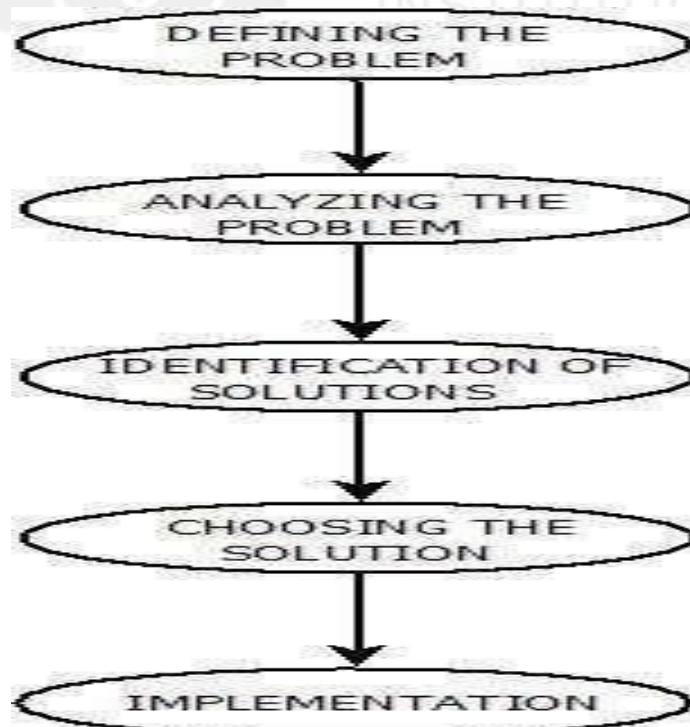
The searching algorithm helps us to search for solution of particular problem.

Problem

Problems are the issues which come across any system. A solution is needed to solve that particular problem.

Steps: Solve Problem Using Artificial Intelligence

The process of solving a problem consists of five steps. They are:



Problem Solving in Artificial Intelligence

Defining The Problem: The definition of the problem must be included precisely. It should contain the possible initial as well as final situations which should result in acceptable solution.

Analysing The Problem: Analysing the problem and its requirement must be done as few features can have immense impact on the resulting solution.

Identification of Solutions: This phase generates reasonable amount of solutions to the given problem in a particular range.

Choosing a Solution: From all the identified solutions, the best solution is chosen basis on the results produced by respective solutions.

Implementation: After choosing the best solution, its implementation is done.

The **reflex agents** are known as the simplest agents because they directly map states into actions. Unfortunately, these agents fail to operate in an environment where the mapping is too large to store and learn. **Goal-based agent**, on the other hand, considers future actions and the desired outcomes.

Here, we will discuss one type of goal-based agent known as a **problem-solving agent**, which uses atomic representation with no internal states visible to the problem-solving algorithms.

1.3.1 Problem-solving agent

The problem-solving agent performs precisely by defining problems and its several solutions.

According to psychology, "a problem-solving refers to a state where we wish to reach to a definite goal from a present state or condition."

According to computer science, a problem-solving is a part of artificial intelligence which encompasses a number of techniques such as algorithms, heuristics to solve a problem.

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

Steps performed by Problem-solving agent

Goal Formulation: It is the first and simplest step in problem-solving. It organizes the steps/sequence required to formulate one goal out of multiple goals as well as actions to achieve that goal. Goal formulation is based on the current situation and the agent's performance measure (discussed below).

Problem Formulation: It is the most important step of problem-solving which decides what actions should be taken to achieve the formulated goal. There are following five components involved in problem formulation:

Initial State: It is the starting state or initial step of the agent towards its goal.

Actions: It is the description of the possible actions available to the agent.

Transition Model: It describes what each action does.

Goal Test: It determines if the given state is a goal state.

Path cost: It assigns a numeric cost to each path that follows the goal. The problem-solving agent selects a cost function, which reflects its performance measure. Remember, **an optimal solution has the lowest path cost among all the solutions.**

Note: **Initial state, actions, and transition model** together define the **state-space** of the problem implicitly. State-space of a problem is a set of all states which can be reached from the initial state followed by any sequence of actions. The state-space forms a directed map or graph where nodes are the states, links between the nodes are actions, and the path is a sequence of states connected by the sequence of actions.

Search: It identifies all the best possible sequence of actions to reach the goal state from the current state. It takes a problem as an input and returns solution as its output.

Solution: It finds the best algorithm out of various algorithms, which may be proven as the best optimal solution.

Execution: It executes the best optimal solution from the searching algorithms to reach the goal state from the current state.

1.3.2 Example Problems

Basically, there are two types of problem approaches:

Toy Problem: It is a concise and exact description of the problem which is used by the researchers to compare the performance of algorithms.

Real-world Problem: It is real-world based problems which require solutions. Unlike a toy problem, it does not depend on descriptions, but we can have a general formulation of the problem.

Some of the most popularly used problem solving with the help of artificial intelligence is:

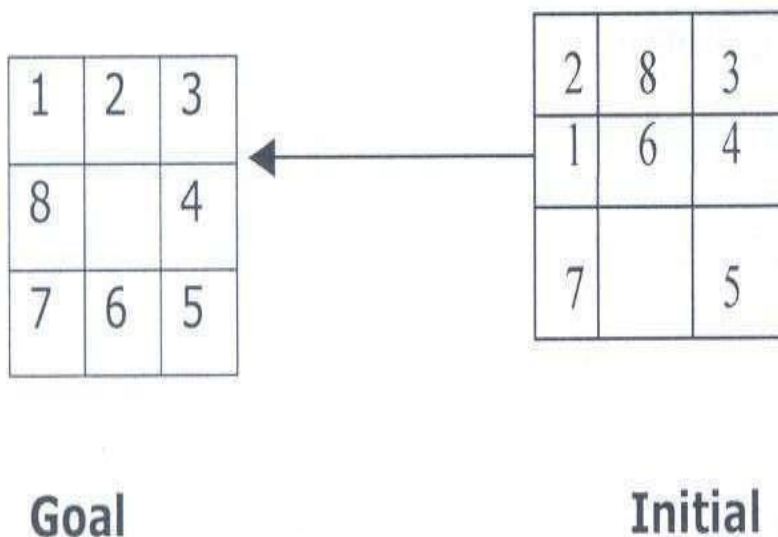
- ✿ **8 puzzle**
- ✿ **N-Queen Problem.**
- ✿ **Ball Collecting / Vacuum Cleaner Problem**
- ✿ **Tower of Hanoi Problem.**
- ✿ **Travelling Salesman Problem.**
- ✿ **Water-Jug Problem.**

SOME TOY PROBLEMS

1. 8 Puzzle Problems:

The **8-puzzle problem** is a **puzzle** invented and popularized by Noyes Palmer Chapman in the 1870s. It is played on a 3-by-3 grid with **8** square blocks labeled 1 through **8** and a blank square. Your goal is to rearrange the blocks so that they are in order.

Here, we have a 3×3 matrix with movable tiles numbered from 1 to 8 with a blank space. The tile adjacent to the blank space can slide into that space. The objective is to reach a specified goal state similar to the goal state, as shown in the below figure.



In the above figure, our task is to convert the current (Start) state into goal state by sliding digits into the blank space.

The problem formulation is as follows:

States: It describes the location of each numbered tiles and the blank tile.

Initial State: We can start from any state as the initial state.

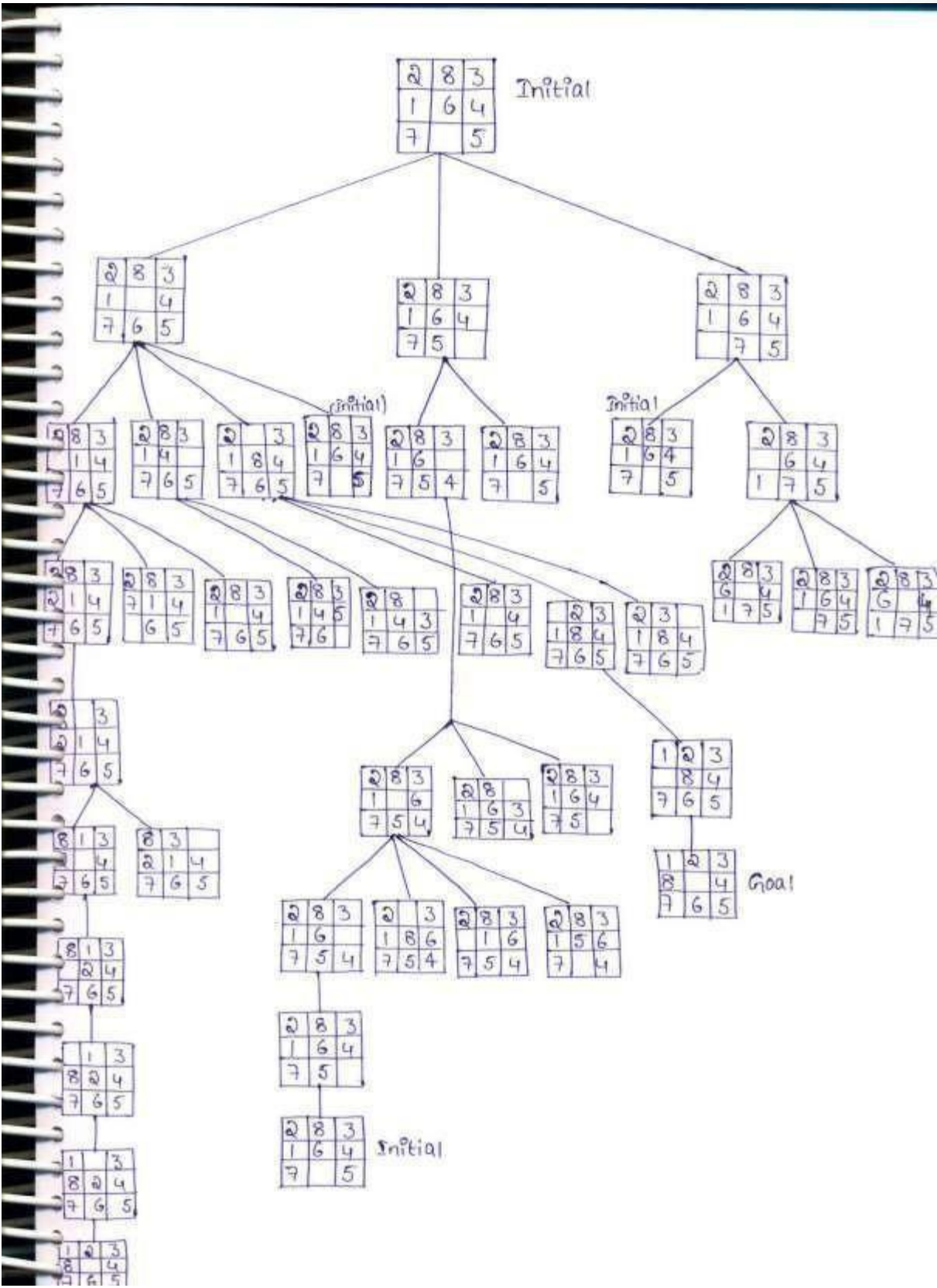
Actions: Here, actions of the blank space is defined, i.e., either **left, right, up or down**

Transition Model: It returns the resulting state as per the given state and actions.

Goal test: It identifies whether we have reached the correct goal-state.

Path cost: The path cost is the number of steps in the path where the cost of each step is 1.

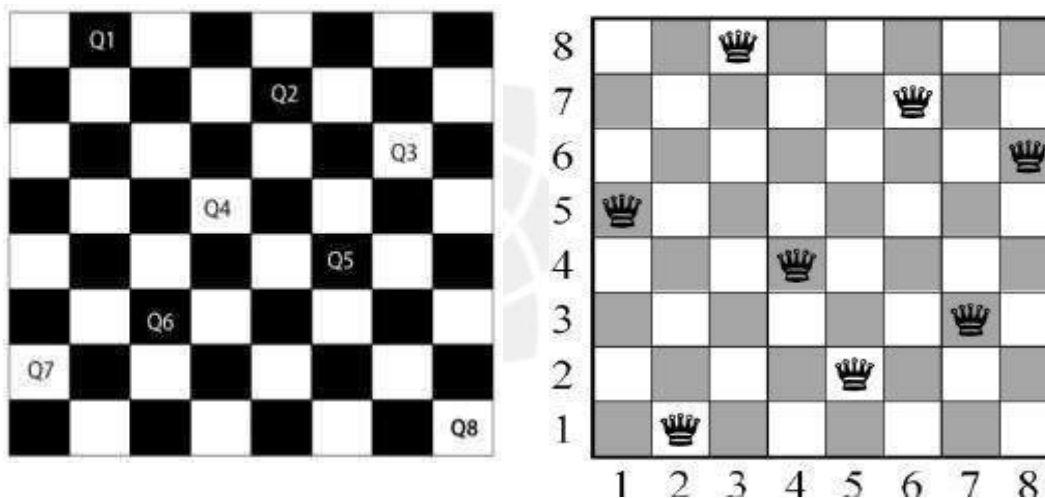
Note: The 8-puzzle problem is a type of **sliding-block problem** which is used for testing new search algorithms in artificial intelligence.



8-queens problem:

8 queen problem. The **eight queens problem** is the **problem** of placing **eight queens** on an **8×8** chessboard such that none of them attack one another (no two are in the same row, column, or diagonal). ... There are different solutions for the **problem**. The aim of this problem is to place eight queens on a chessboard in an order where no queen may attack another. A queen can attack other queens either **diagonally or in same row and column**. From the following figure, we can understand the problem as well as its correct solution.

It is noticed from the above figure that each queen is set into the chessboard in a position where no other queen is placed diagonally, in same row or column. Therefore, it is one right approach to the 8-queens problem.



Problem and Solution

For this problem, there are two main kinds of formulation:

Incremental formulation: It starts from an empty state where the operator augments a queen at each step.

Following steps are involved in this formulation:

States: Arrangement of any 0 to 8 queens on the chessboard.

Initial State: An empty chessboard

Actions: Add a queen to any empty box.

Transition model: Returns the chessboard with the queen added in a box.

Goal test: Checks whether 8-queens are placed on the chessboard without any attack.

Path cost: There is no need for path cost because only final states are counted.

In this formulation, there is approximately 1.8×10^{14} possible sequence to investigate.

Complete-state formulation: It starts with all the 8-queens on the chessboard and moves them around, saving from the attacks.

This formulation is better than the incremental formulation as it reduces the state space from 1.8×10^{14} to **2057**, and it is easy to find the solutions.

VACUUM CLEANER PROBLEM:-

Vacuum cleaner problem is a well-known search problem for an agent which works on Artificial Intelligence. In this problem, our vacuum cleaner is our agent. It is a goal based agent, and the goal of this agent, which is the vacuum cleaner, is to clean up the whole area. So, in the classical vacuum cleaner problem, we have two rooms and one vacuum cleaner. There is dirt in both the rooms and it is to be cleaned. The vacuum cleaner is present in any one of these rooms. So, we have to reach a state in which both the rooms are clean and are dust free.

SOLUTION:-

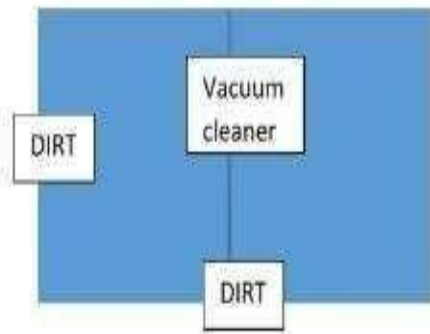
So, there are eight possible states possible in our **vacuum cleaner problem**.

These can be well illustrated with the help of the following diagrams:

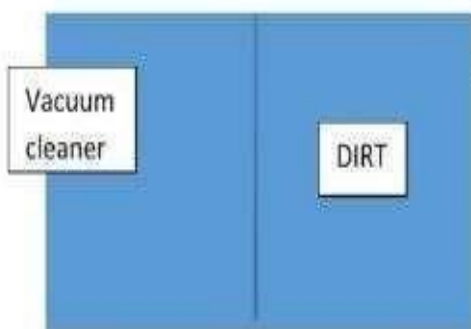
1



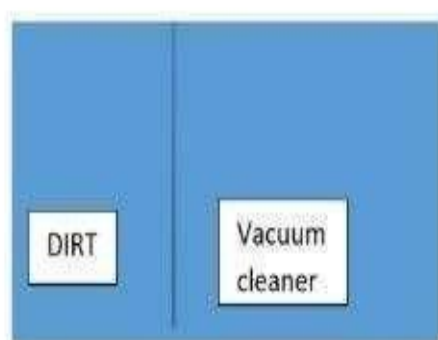
2



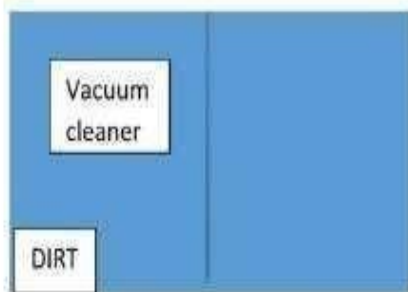
3



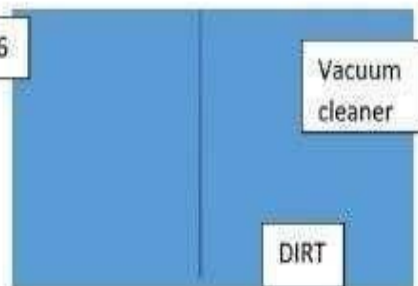
4



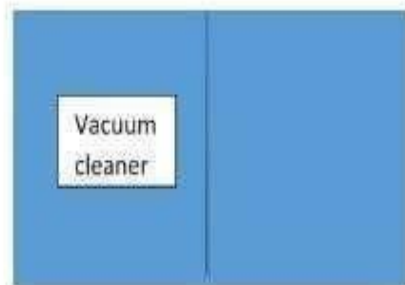
5



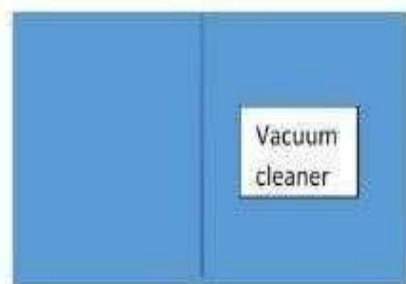
6



7



8



SOME REAL-WORLD PROBLEMS

Traveling salesperson problem (TSP): It is a **touring problem** where the salesman can visit each city only once. The objective is to find the shortest tour and sell-out the stuff in each city.

The "**Traveling Salesman Problem**" (TSP) is a common problem applied to **artificial intelligence**. The **TSP** presents the computer with a number of cities, and the computer must compute the optimal path between the cities. This applet uses a genetic algorithm to produce a solution to the "**Traveling Salesman Problem**".

The traveling salesman problem (TSP) is a problem in discrete or combinatorial optimization. It is a prominent illustration of a class of problems in computational complexity theory, which are classified as NP-hard. In the traveling-salesman problem, which is closely related to the Hamiltonian cycle problem, a salesman must visit n cities. Modeling the problem as a complete graph with n vertices, we can say that the salesman wishes to make a tour, or hamiltonian cycle, visiting each city exactly once and to finishing at the city he starts from. There is an integer cost $c(i, j)$ to travel from city i to city j , and the salesman wishes to make the tour whose total cost is minimum, where the total cost is the sum of the individual costs along the edges of the tour.

Representing the cities by vertices and the roads between them by edges. We get a graph. In this graph, with every edge there is associated a real number such a graph is called a weighted graph being the weight of edge.

In our problem, if each of the cities has a road to every other city, we have a complete weighted graph. This graph has numerous Hamiltonian circuits, and we are to pick the one that has the smallest sum of the distance. The total number of different Hamiltonian circuits in a complete graph of n vertices can be shown to be $(n - 1)!/2$.

This follows from the fact that starting from any vertex we have $n - 1$ edges to choose from the first vertex, $n - 2$ from the second, $n - 3$ from the third, and so on, these being independent, result with $(n-1)!$ choices. This number is, however, divided by 2, because each Hamiltonian circuit has been counted twice. Theoretically the problem of the traveling salesman can always be solved by enumeration of all $(n-1)!/2$ Hamiltonian circuits, calculation of the distance traveled in each, and then picking the shortest one. However for a large value of n , the labor involved is too great even for a digital computer.

The problem is to prescribe a manageable algorithm for finding the shortest route. No efficient algorithm for problems of arbitrary size has yet been found, although many attempts have been made. Since this problem has application in operations research, some specific large-scale examples have been worked out. There are also available several heuristic methods of solution that give a route very close to the shortest one, but do not guarantee the shortest.

The problem Statement

Back in the days when salesmen traveled door-to-door hawking vacuums and encyclopedias, they had to plan their routes, from house to house or city to city. The shorter the route, the better. Finding the shortest route that visits a set of locations is an exponentially difficult problem: finding the shortest path for 20 locations is much more than twice as hard as 10 locations.

An exhaustive search of all possible paths would be guaranteed to find the shortest, but is computationally intractable for all but small sets of locations. For larger problems, optimization techniques are needed to intelligently search the solution space and find near-optimal solutions.

Mathematically, traveling salesman problems can be represented as a graph, where the locations are the nodes and the edges (or arcs) represent direct travel between the locations. The weight of each edge is the distance between the nodes. The goal is to find the path with the shortest sum of weights.

Below, we see a simple four-node graph and the shortest cycle that visits every node:

In addition to finding solutions to the classical Traveling Salesman Problem, OR-Tools also provides methods for more general types of TSPs, including the following:

Asymmetric cost problems—The traditional TSP is symmetric: the distance from point A to point B equals the distance from point B to point A. However, the cost of shipping items from point A to point B might not equal the cost of shipping them from point B to point A. OR-Tools can also handle problems that have asymmetric costs.

Prize-collecting TSPs, where benefits accrue from visiting nodes

TSP with time windows

Example:

Consider the following set of cities:

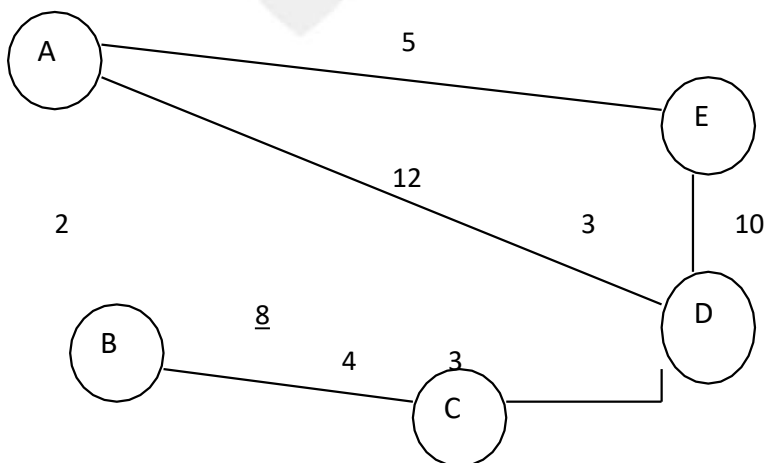


Figure 10.1 A graph with weights on its edges.

The problem lies in finding a minimal path passing from all vertices once. For example the path Path1 {A, B, C, D, E, A} and the path Path2 {A, B, C, E, D, A} pass all the vertices but Path1 has a total length of 24 and Path2 has a total length of 31.

Definition:

A Hamiltonian cycle is a cycle in a graph passing through all the vertices once.

The layout problem is split into two parts:

Cell layout: Here, the primitive components of the circuit are grouped into cells, each performing its specific function. Each cell has a fixed shape and size. The task is to place the cells on the chip without overlapping each other.

Channel routing: It finds a specific route for each wire through the gaps between the cells.

Protein Design: The objective is to find a sequence of amino acids which will fold into 3D protein having a property to cure some disease.

WATER JUG PROBLEM

Consider the following problem:

A Water Jug Problem: You are given two jugs, a 4-gallon one and a 3-gallon one, a pump which has unlimited water which you can use to fill the jug, and the ground on which water may be poured. Neither jug has any measuring markings on it. How can you get exactly 2 gallons of water in the 4-gallon jug?

State Representation and Initial State : we will represent a state of the problem as a tuple (x, y) where x represents the amount of water in the 4-gallon jug and y represents the amount of water in the 3-gallon jug. Note $0 \leq x \leq 4$, and $0 \leq y$

≤ 3 .

✿ **Our initial state:** (0,0)

✿ **Goal Predicate** state = (2,y) where $0 \leq y \leq 3$.

✿ **Operators :** we must design a set of operators that will take us from one state to another:

1. Fill 4-gal jug	$(x,y) \rightarrow (4,y)$	$x < 4$
2. Fill 3-gal jug	$(x,y) \rightarrow (x,3)$	$y < 3$
3. Empty 4-gal jug on ground	$(x,y) \rightarrow (0,y)$	$x > 0$
4. Empty 3-gal jug on ground	$(x,y) \rightarrow (x,0)$	$y > 0$
5. Pour water from 3-gal jug to 4-gal jug	$(x,y) \rightarrow (4, y - (4 - x))$	$0 < x+y \leq 4$ and $y > 0$
6. Pour water from 4-gal jug to 3-gal-jug	$(x,y) \rightarrow (x - (3-y), 3)$	$0 < x+y \leq 3$ and $x > 0$
7. Pour all of water from 3-gal jug into 4-gal jug	$(x,y) \rightarrow (x+y, 0)$	$0 < x+y \leq 4$ and $y \leq 0$
8. Pour all of water from 4-gal jug into 3-gal jug	$(x,y) \rightarrow (0, x+y)$	$0 < x+y \leq 3$ and $x \leq 0$

SOLUTION:-

Through Graph Search, the following solution is found:

Gals in 4-gal jug Gals in 3-gal jug Rule Applied

0 0

1. Fill 4

4 0

6. Pour 4 into 3 to 1

1 3

4. Empty 3

1 0

8. Pour all of 4 into 3

0 1

1. Fill 4

4 1

6. Pour into 3

2 3

1.3.3 UNINFORMED SEARCH STRATEGIES

This topic covers several search strategies that come under the heading of uninformed search (also called blind search). The term means that the strategies have no additional information about states beyond that provided in the problem definition. All they can do is generate successors and distinguish a goal state from a non-goal state. All search strategies are distinguished by the order in which nodes are expanded. Strategies that know whether one non-goal state is “more promising” than another are called informed search or heuristic search strategies.

i) Breadth-first search

Breadth-first search is a simple strategy in which the root node is expanded first, then all the successors of the root node are expanded next, then their successors, and so on. In general, all the nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded.

Breadth-first search is an instance of the general graph-search algorithm in which the shallowest unexpanded node is chosen for expansion. This is achieved very simply by using a FIFO queue for the frontier. Thus, new nodes (which are always deeper than their parents) go to the back of the queue, and old nodes, which are shallower than the new nodes, get expanded first. There is one slight tweak on the general graph-search algorithm, which is that the goal test is applied to each node when it is generated rather than when it is selected for expansion. This decision is explained below, where we discuss time complexity.

Figure 4. shows the progress of the search on a simple binary tree. BFS is complete— if the shallowest goal node is at some finite depth d , breadth-first search will eventually find it after generating all shallower nodes (provided the branching factor b is finite). Note that as soon as a goal node is generated, we know it is the shallowest goal node because all shallower nodes must have been generated already and failed the goal test. Now, the shallowest goal node is not necessarily the optimal one technically, breadth-first search is optimal if the path cost is a nondecreasing function of the depth of the node.

Algorithm for Breadth-first search on a graph:

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure  
node  $\leftarrow$  a node with STATE = problem.INITIAL-STATE, PATH-COST = 0  
if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)  
frontier  $\leftarrow$  a FIFO queue with node as the only element  
explored  $\leftarrow$  an empty set  
loop do  
  if EMPTY?(frontier) then return failure  
  node  $\leftarrow$  POP(frontier) /* chooses the shallowest node in frontier */  
  add node.STATE to explored  
  for each action in problem.ACTIONS(node.STATE) do  
    child  $\leftarrow$  CHILD-NODE(problem, node, action)  
    if child.STATE is not in explored or frontier then  
      if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)  
      frontier  $\leftarrow$  INSERT(child, frontier)
```

The most common such scenario is that all actions have the same cost. As for space complex ity: for any kind of graph search, which stores every expanded node in the explored set, the space complex ity is always within a factor of b of the time complex ity. For breadth-first graph search in particular, every node generated remains in memory. There will be $O(b^{d-1})$ nodes in the explored set and $O(b^d)$ nodes in the frontier, so the space complex ity is $O(b^d)$, i.e., it is dom inated by the size of the frontier. Switching to a tree search would not save much space, and in a state space with many redundant paths, switching could cost a great deal of time.

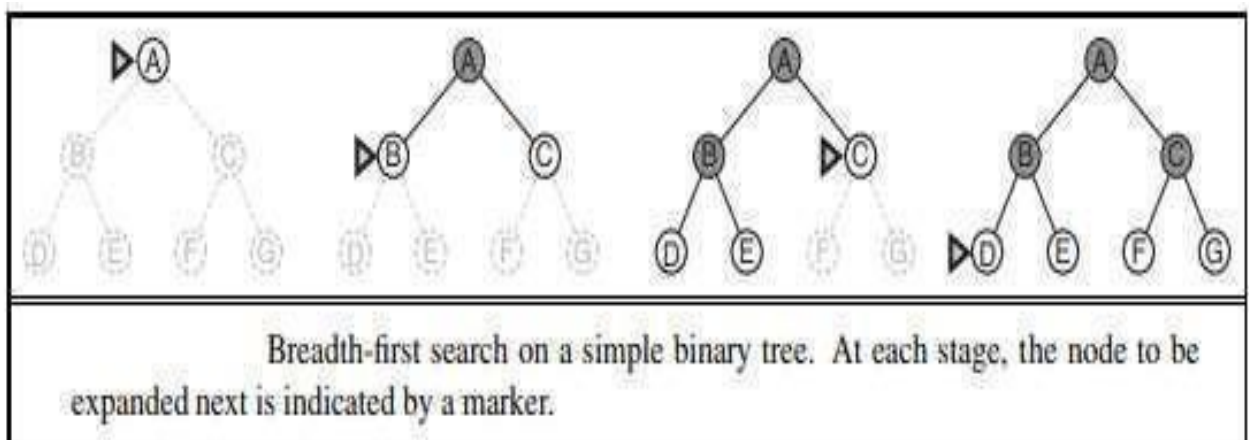
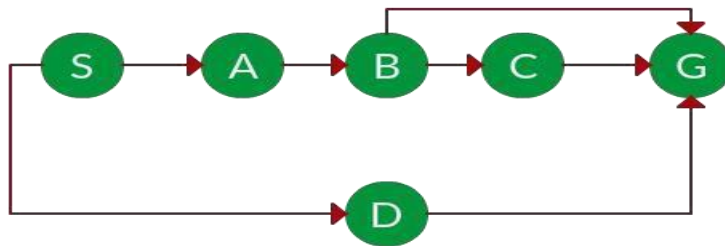


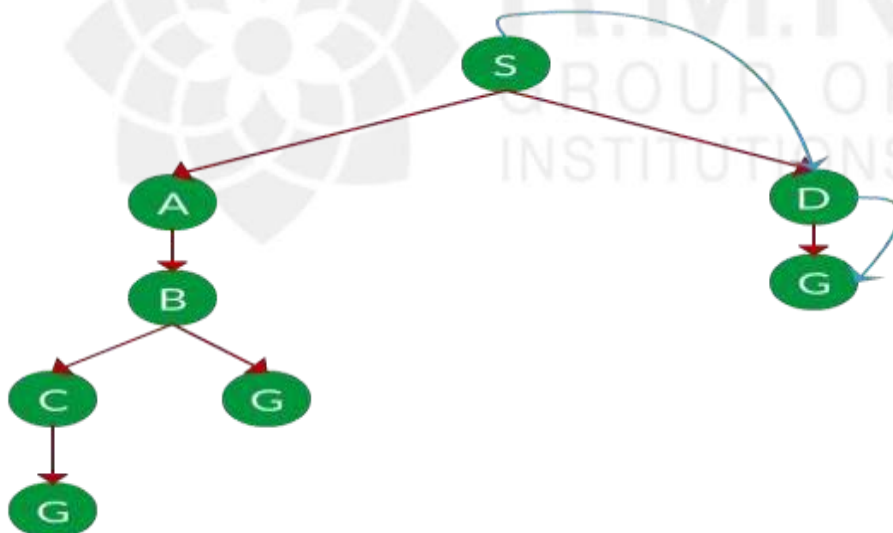
Figure 4. The progress of the search on a simple binary tree.

Example:

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



Solution. 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 in blue arrows



Path: S -> D -> G

Two lessons can be learned. First, the memory requirements are a bigger problem for breadth-first search than is the execution time. The second lesson is that time is still a major factor.

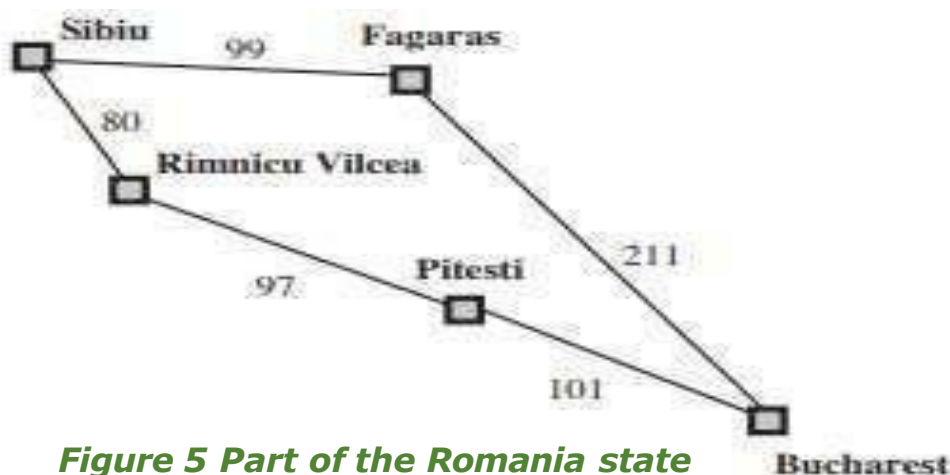
ii) Uniform-cost search

When all step costs are equal, breadth-first search is optimal because it always expands the shallowest unexpanded node. By a simple extension, we can find an algorithm that is optimal with any step-cost function. Instead of expanding the shallowest node, uniform-cost search expands the node n with the lowest path cost $g(n)$. This is done by storing the frontier as a priority queue ordered by g .

In addition to the ordering of the queue by path cost, there are two other significant differences from breadth-first search. The first is that the goal test is applied to a node when it is selected for expansion rather than when it is first generated. The reason is that the first goal node that is generated may be on a suboptimal path. The second difference is that a test is added in case a better path is found to a node currently on the frontier.

Both of these modifications come into play in the example shown in Figure 5, where the problem is to get from Sibiu to Bucharest. The successors of Sibiu are Rimnicu Vilcea and Fagaras, with costs 80 and 99, respectively. The least-cost node, Rimnicu Vilcea, is expanded next, adding Pitesti with cost $80 + 97 = 177$. The least-cost node is now Fagaras, so it is expanded, adding Bucharest with cost $99 + 211 =$

310. Now a goal node has been generated, but uniform-cost search keeps going, choosing Pitesti for expansion and adding a second path to Bucharest with cost $80 + 97 + 101 = 278$. Now the algorithm checks to see if this new path is better than the old one; it is, so the old one is discarded. Bucharest, now with g -cost 278, is selected for expansion and the solution is returned.



**Figure 5 Part of the Romania state space,
selected to illustrate uniform-cost search**

It is easy to see that uniform-cost search is optimal in general. First, we observe that whenever uniform-cost search selects a node n for expansion, the optimal path to that node has been found. Then, because step costs are nonnegative, paths never get shorter as nodes are added. These two facts together imply that uniform-cost search expands nodes in order of their optimal path cost. Hence, the first goal node selected for expansion must be the optimal solution.

Algorithm for Uniform-cost search on a graph

```

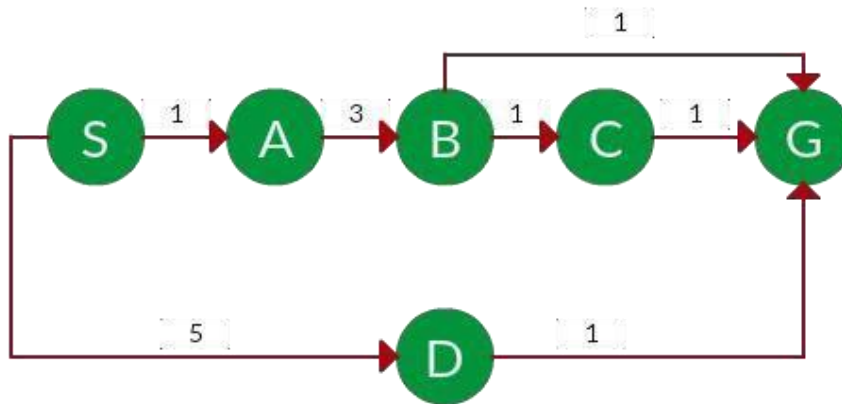
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
  node  $\leftarrow$  a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  frontier  $\leftarrow$  a priority queue ordered by PATH-COST, with node as the only element
  explored  $\leftarrow$  an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node  $\leftarrow$  POP(frontier) /* chooses the lowest-cost node in frontier */
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child  $\leftarrow$  CHILD-NODE(problem, node, action)
      if child.STATE is not in explored or frontier then
        frontier  $\leftarrow$  INSERT(child, frontier)
      else if child.STATE is in frontier with higher PATH-COST then
        replace that frontier node with child

```

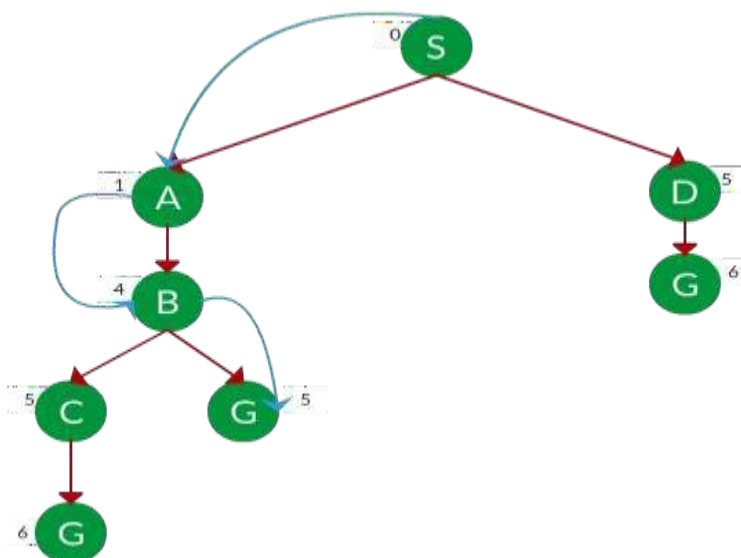
Uniform-cost search is guided by path costs rather than depths, so its complexity is not easily characterized in terms of b and d . Instead, let C^* be the cost of the optimal solution, and assume that every action costs at least ϵ . Then the algorithm's worst-case time and space complexity is $O(b^{1+\lceil C^*/\epsilon \rceil})$, which can be much greater than b^d . This is because uniform-cost search can explore large trees of small steps before exploring paths involving large and perhaps useful steps.

Example:

Question. Which solution would UCS find to move from node S to node G if run on the graph below?



Solution. The equivalent search tree for the above graph is as follows. Cost of each node is the cumulative cost of reaching that node from the root. Based on UCS strategy, the path with least cumulative cost is chosen. Note that due to the many options in the fringe, the algorithm explores most of them so long as their cost is low, and discards them when a lower cost path is found; these discarded traversals are not shown below. The actual traversal is shown in blue.



Path: S -> A -> B -> G

iii) Depth-first search

Depth-first search always expands the deepest node in the current frontier of the search tree. The search proceeds immediately to the deepest level of the search tree, where the nodes have no successors. As those nodes are expanded, they are dropped from the frontier, so then the search “backs up” to the next deepest node that still has unexplored successors.

The depth-first search algorithm is an instance of the graph-search algorithm; whereas breadth-first-search uses a FIFO queue, depth-first search uses a LIFO queue. A LIFO queue means that the most recently generated node is chosen for expansion. This must be the deepest unexpanded node because it is one deeper than its parent—which, in turn, was the deepest unexpanded node when it was selected. As an alternative to the GRAPH-SEARCH-style implementation, it is common to implement depth-first search with a recursive function that calls itself on each of its children in turn. Depth-first search is shown in figure 6. The unexplored region is shown in light gray. Explored nodes with no descendants in the frontier are removed from memory. Nodes at depth 3 have no successors and M is the only goal node.

The properties of depth-first search depend strongly on whether the graph-search or tree-search version is used. The graph-search version, which avoids repeated states and redundant paths, is complete in finite state spaces because it will eventually expand every node. Depth-first tree search can be modified at no extra memory cost so that it checks new states against those on the path from the root to the current node; this avoids infinite loops in finite state spaces but does not avoid the proliferation of redundant paths. In infinite state spaces, both versions fail if an infinite non-goal path is encountered. The time complexity of depth-first graph search is bounded by the size of the state space (which may be infinite, of course).

A depth-first tree search, on the other hand, may generate all of the $O(b^m)$ nodes in the search tree, where m is the maximum depth of any node; this can be much greater than the size of the state space. Note that m itself can be much larger than d (the depth of the shallowest solution) and is infinite if the tree is unbounded.

A variant of depth-first search called backtracking search uses still less memory. In backtracking, only one successor is generated at a time rather than all successors; each partially expanded node remembers which successor to generate next. In this way, only $O(m)$ memory is needed rather than $O(b^m)$.

Backtracking search facilitates yet another memory-saving (and time-saving) trick: the idea of generating a successor by modifying the current state description directly rather than copying it first. This reduces the memory requirements to just one state description and $O(m)$ actions.

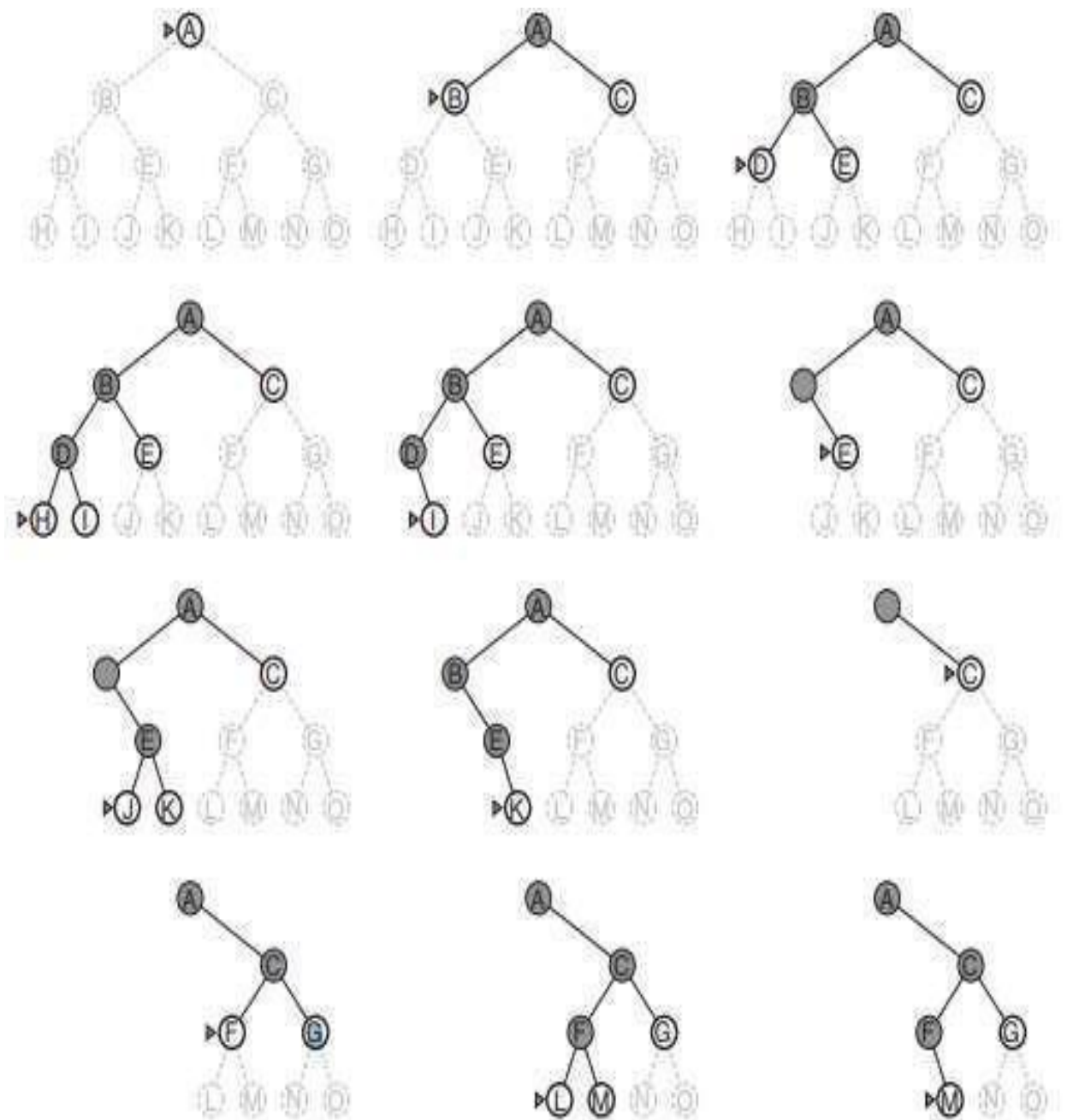
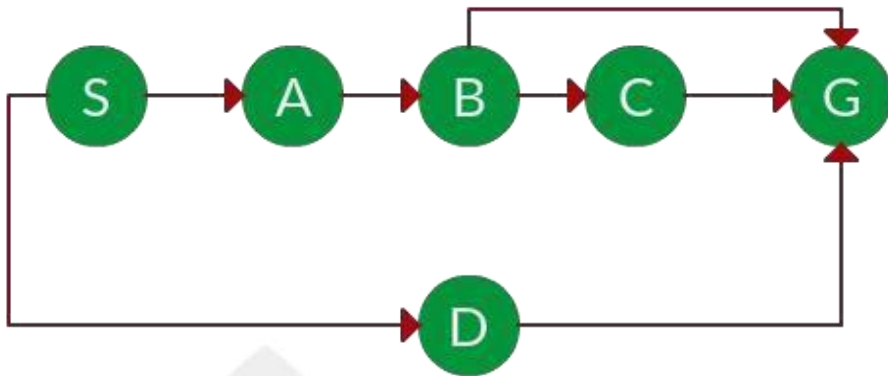


Figure 6. Depth-first search on a binary tree

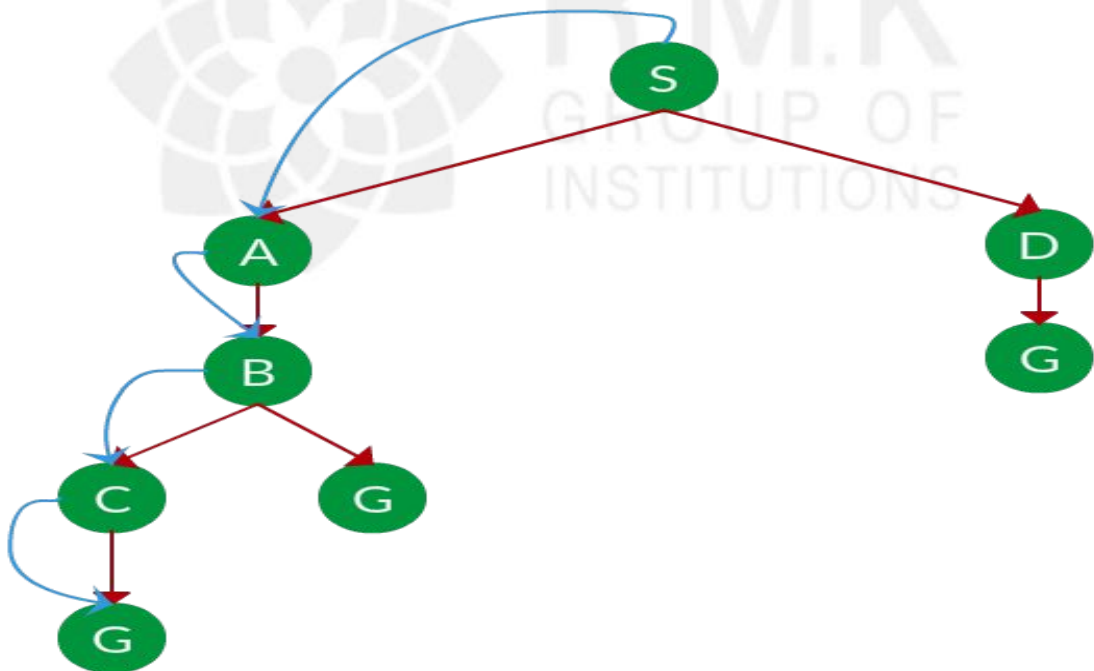
Example:

Question. Which solution would DFS find to move from node S to node G if run on the graph below?

Solution. The equivalent search tree for the above graph is as follows. As DFS traverses the tree “deepest node first”, it would always pick the deeper branch until it reaches the solution (or it runs out of nodes, and goes to the next branch). The traversal is shown in blue arrows.



Solution



Path: S -> A -> B -> C -> G

iv) Depth-limited search

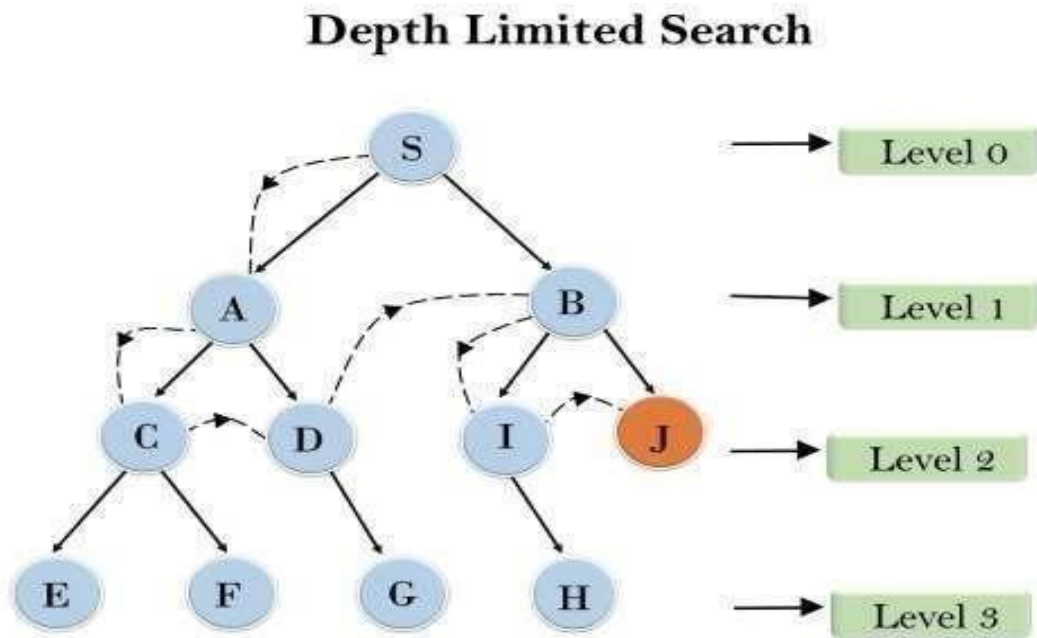
The embarrassing failure of depth-first search in infinite state spaces can be alleviated by supplying depth-first search with a predetermined depth limit. That is, nodes at depth are treated as if they have no successors. This approach is called depth-limited search. The depth limit solves the infinite-path problem. Unfortunately, it also introduces an additional source of incompleteness if we choose d . Its time complexity is $O(b^d)$ and its space complexity is $O(b)$. Depth-first search can be viewed as a special case of depth-limited search with $d = \infty$. This number, known as the diameter of the state space, gives us a better depth limit, which leads to a more efficient depth-limited search. For most problems, however, we will not know a good depth limit until we have solved the problem.

A recursive implementation of depth-limited tree search algorithm

```
function DEPTH-LIMITED-SEARCH(problem, limit) returns a solution, or failure/cutoff  
  return RECURSIVE-DLS(MAKE-NODE(problem.INITIAL-STATE), problem, limit)
```

```
function RECURSIVE-DLS(node, problem, limit) returns a solution, or failure/cutoff  
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)  
  else if limit = 0 then return cutoff  
  else  
    cutoff_occurred? ← false  
    for each action in problem.ACTIONS(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
```

Example:



Depth-limited search can be implemented as a simple modification to the general tree or graph-search algorithm. Alternatively, it can be implemented as a simple recursive algorithm. Notice that depth-limited search can terminate with two kinds of failure: the standard failure value indicates no solution; the cutoff value indicates no solution within the depth limit.

v) Iterative deepening depth-first search

Iterative deepening search (or iterative deepening depth-first search) is a general strategy, often used in combination with depth-first tree search that finds the best depth limit. It does this by gradually increasing the limit—first 0, then 1, then 2, and so on—until a goal is found. This will occur when the depth limit reaches d , the depth of the shallowest goal node. Iterative deepening combines the benefits of depth-first and breadth-first search. Like depth-first search, its memory requirements are modest: $O(bd)$ to be precise. Like breadth-first search, it is complete when the branching factor is finite and optimal when the path cost is a nondecreasing function of the depth of the node. Figure 7 shows four iterations of ITERATIVE-DEEPENING-SEARCH on a binary search tree, where the solution is found on the fourth iteration.

Iterative deepening search may seem wasteful because states are generated multiple times. It turns out this is not too costly. The reason is that in a search tree with the same (or nearly the same) branching factor at each level, most of the nodes are in the bottom level, so it does not matter much that the upper levels are generated multiple times. In an iterative deepening search, the nodes on the bottom level (depth d) are generated once, those on the next-to-bottom level are generated twice, and so on, up to the children of the root, which are generated d times.

In general, iterative deepening is the preferred uninformed search method when the search space is large and the depth of the solution is not known. Iterative deepening search is analogous to breadth-first search in that it explores a complete layer of new nodes at each iteration before going on to the next layer. It would seem worthwhile to develop an iterative analog to uniform-cost search, inheriting the latter algorithm's optimality guarantees while avoiding its memory requirements. The idea is to use increasing path-cost limits instead of increasing depth limits. The resulting algorithm is called iterative lengthening search. It turns out, unfortunately, that iterative lengthening incurs substantial overhead compared to uniform-cost search.

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-limited search returns failure, meaning that no solution exists.

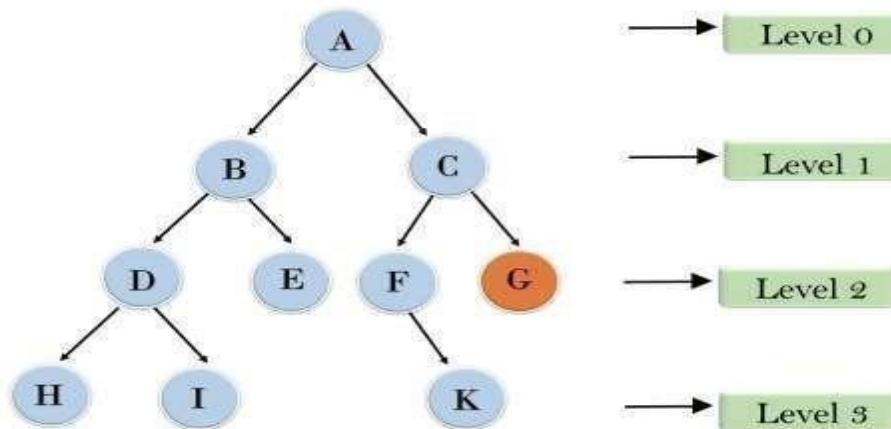
Algorithm of The iterative deepening search

```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution, or failure
  for depth = 0 to  $\infty$  do
    result  $\leftarrow$  DEPTH-LIMITED-SEARCH(problem, depth)
    if result  $\neq$  cutoff then return result
```

Example:

Following tree structure is showing the iterative deepening depth-first search. IDDFS algorithm performs various iterations until it does not find the goal node. The iteration performed by the algorithm is given as:

Iterative deepening depth first search



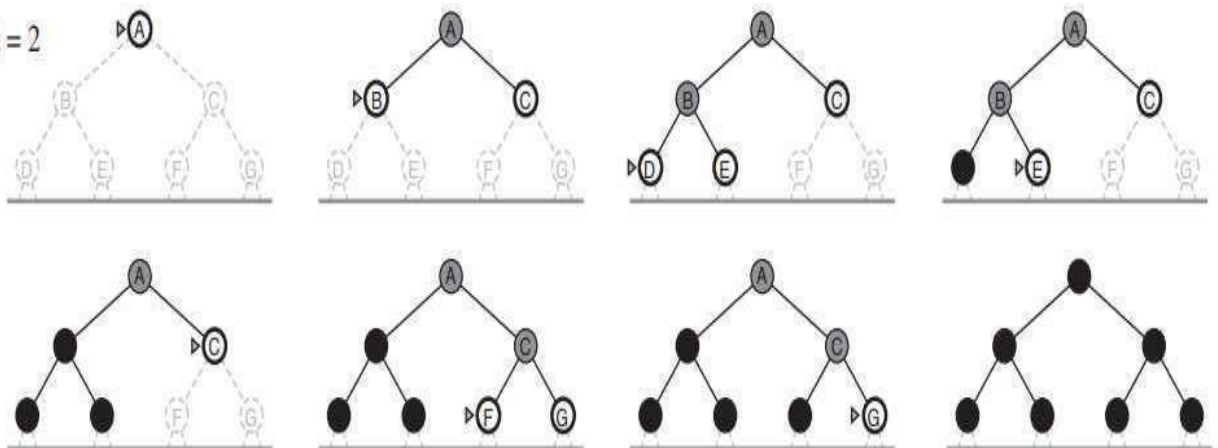
Limit = 0



Limit = 1



Limit = 2



Limit = 3

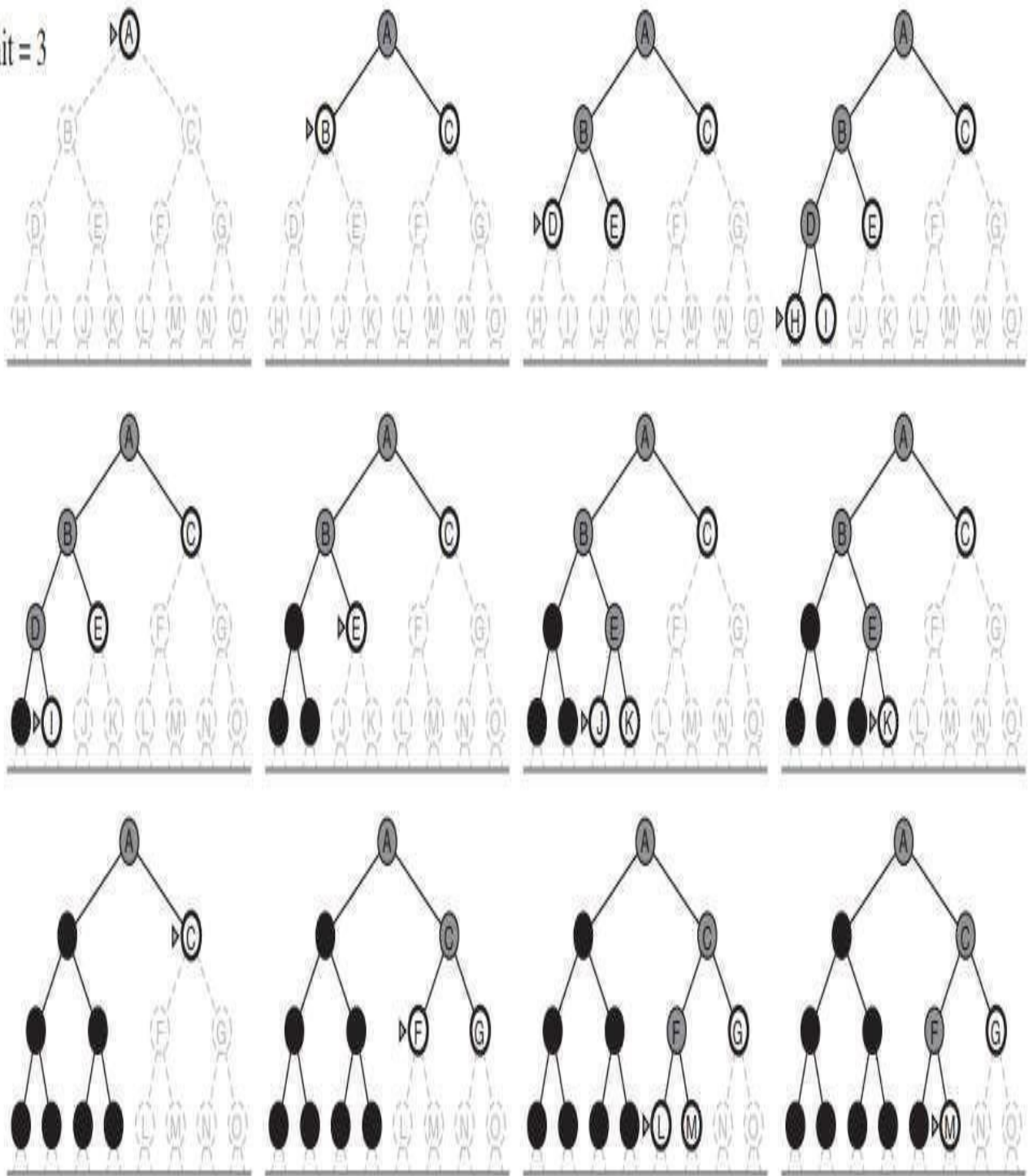


Figure 7. Four iterations of iterative deepening search on a binary tree

vi) Bidirectional search

The idea behind bidirectional search is to run two simultaneous searches—one forward from the initial state and the other backward from the goal—hoping that the two searches meet in the middle. Bidirectional search is implemented by replacing the goal test with a check to see whether the frontiers of the two searches intersect; if they do, a solution has been found. (It is important to realize that the first such solution found may not be optimal, even if the two searches are both breadth-first; some additional search is required to make sure there isn't another short-cut across the gap.) The check can be done when each node is generated or selected for expansion and, with a hash table, will take constant time. Bidirectional search requires a method for computing predecessors. When all the actions in the state space are reversible, the predecessors of x are just its successors. Other cases may require substantial ingenuity.

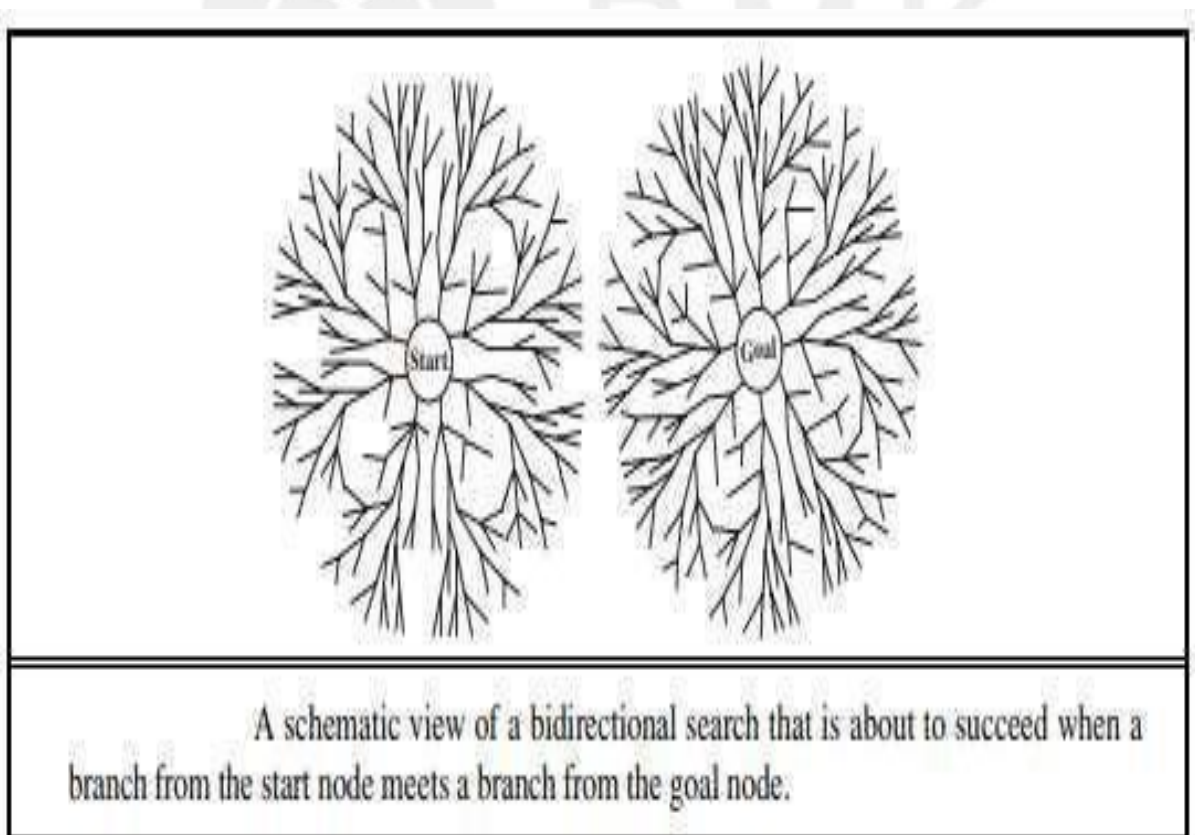


Figure 8 A Schematic view of a bidirectional search.

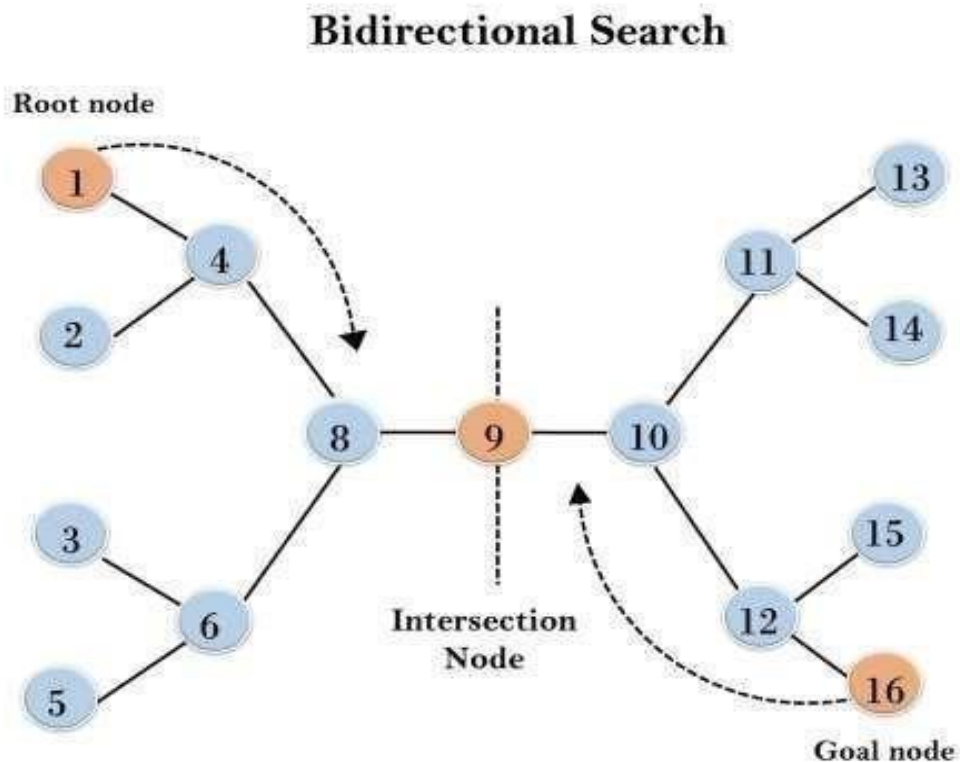
For example, if a problem has solution depth $d = 6$, and each direction runs breadth-first search one node at a time, then in the worst case the two searches meet when they have generated all of the nodes at depth 3. For $b = 10$, this means a total of 2,220 node generations, compared with 1,111,110 for a standard breadth-first search. Thus, the time complexity of bidirectional search using breadth-first searches in both directions is $O(b^{d/2})$. The space complexity is also $O(b^{d/2})$. Figure 9 shows comparison of uninformed search strategies.

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes ^a	Yes ^{a,b}	No	No	Yes ^a	Yes ^{a,d}
Time	$O(b^d)$	$O(b^{1+\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes ^c	Yes	No	No	Yes ^c	Yes ^{c,d}
Evaluation of tree-search strategies. b is the branching factor, d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: ^a complete if b is finite; ^b complete if step costs $\geq \epsilon$ for positive ϵ ; ^c optimal if step costs are all identical; ^d if both directions use breadth-first search.						

Figure 9 Comparing uninformed search strategies

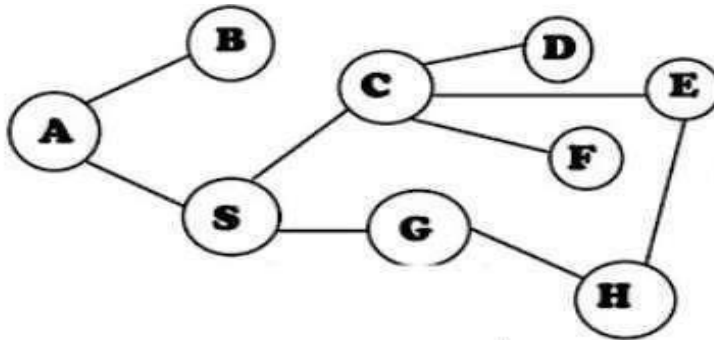
Example:

In the below search tree, bidirectional search algorithm is applied. This algorithm divides one graph/tree into two sub-graphs. It starts traversing from node 1 in the forward direction and starts from goal node 16 in the backward direction. The algorithm terminates at node 9 where two searches meet.



10.ASSIGNMENT 1- UNIT 1

1.Solve the graph using BFS, DFS, Iterative Deepening Search Initial state: C ,Goal state: H



2. List out any 2 real time agents with its PEAS Properties?



11.PART A Q & A (WITH K LEVEL AND CO) UNIT 1

1. Define Artificial Intelligence (AI). [Nov/Dec 2008] [Apr/May 2008] K1,CO1

The study of how to make computers do things at which at the moment, people are better.

- Systems that think like humans
- ✿ Systems that act like humans
- ✿ Systems that think rationally
- ✿ Systems that act rationally.
- ✿

2. Differentiate between Intelligence and Artificial Intelligence. K1,CO1

INTELLIGENCE	ARTIFICIAL INTELLIGENCE
It is a natural process.	It is programmed by humans.
It is actually hereditary.	It is not hereditary.
Knowledge is required for intelligence.	KB and electricity are required to generate output.
No human is an expert. We may get better solutions from other humans.	Expert systems are made which aggregate many person's experience and ideas.

3. Is AI a science, or is it engineering? Or neither or both? Explain. K1,CO1

Artificial Intelligence is most certainly a science. But it would be nothing with engineering. Computer Scientists need somewhere to place their programs, such as computers, servers, robots, cars, etc. But without engineers they would have no outlet to test their Artificial Intelligence on. Science and Engineering go hand in hand, they both benefit each other. While the engineers build the machines, the scientists are writing code for their AI.

4. Explain why problem formulation must follow goal formulation. K2,CO1

Well goal formulation is used to steer the agent in the right direction, thus ignoring any redundant actions. Problem formulation must follow this because it is based off of Goal Formulation. It is the process of deciding what actions and states to consider given a certain goal. A goal may be set in stone, but how you achieve it can vary. Usually the most optimal way is chosen though..

5. Define Artificial Intelligence in terms of rational acting. K2,CO1

A field of study that seeks to explain and emulate intelligent behaviors in terms of computational processes-Schalkoff. The branch of computer science that is concerned with the automation of intelligent behavior-Luger & Stubblefield.

6. Define Artificial in terms of rational thinking. K2,CO1

The study of mental faculties through the use of computational models-Charniak & McDermott. The study of the computations that make it possible to perceive, reason and act-Winston.

7. What is meant by Turing test? K2,CO1

A Turing Test is a method of inquiry in artificial intelligence (AI) for determining whether or not a computer is capable of thinking like a human being. The test is named after Alan Turing. To conduct this test we need two people and one machine. One person will be an interrogator (i.e.) questioner, will be asking questions to one person and one machine. Three of them will be in a separate room. Interrogator knows them just as A and B. so it has to identify which is the person and machine. The goal of the machine is to make Interrogator believe that it is the person's answer. If machine succeeds by fooling Interrogator, the machine acts like a human. Programming a computer to pass Turing test is very difficult.

8. What are the capabilities, computer should possess to pass Turing test? K2,CO1

- ✿ **natural language processing** -to enable it to communicate successfully in English
- ✿ **knowledge representation** - to store what it knows or hears
- ✿ **automated reasoning** - to use the stored information to answer questions and to draw new conclusions
- ✿ **machine learning** - to adapt to new circumstances and to detect and extrapolate patterns.

9. What is meant by Total Turing Test ? K2,CO1

The test which includes a video signals so that the interrogator can test the perceptual abilities of the machine is termed Total turing test.

10. What are the capabilities computers needs to pass total Turing test?

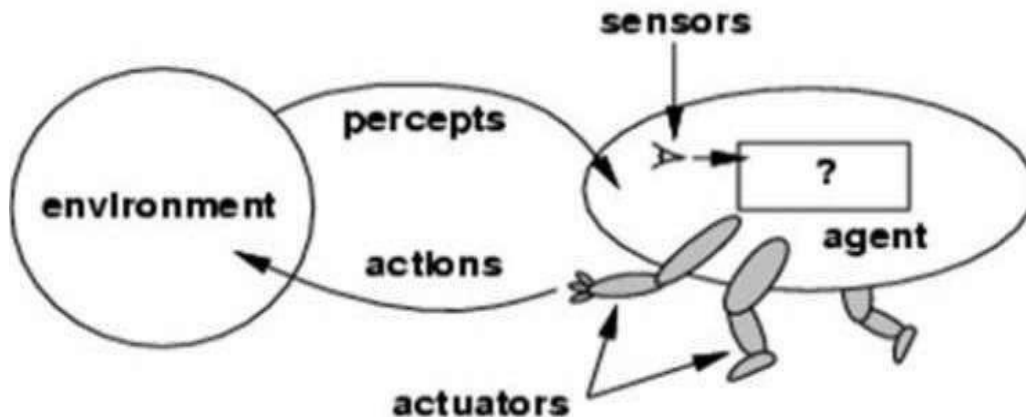
K2,CO1

Computer vision - to perceive objects.

Robotics - to manipulate objects and move about.

11. Define an agent. K1,CO1

An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators



12. What is an agent function? Differentiate an agent function and an agent program. K2,CO1

An agent's behavior is described by the agent function that maps any given percept sequence to an action.

AGENT FUNCTION - An abstract mathematical description

AGENT PROGRAM - A concrete implementation, running on the agent Architecture.

13.. What are the factors that a rational agent should depend on at any given time? K1,CO1

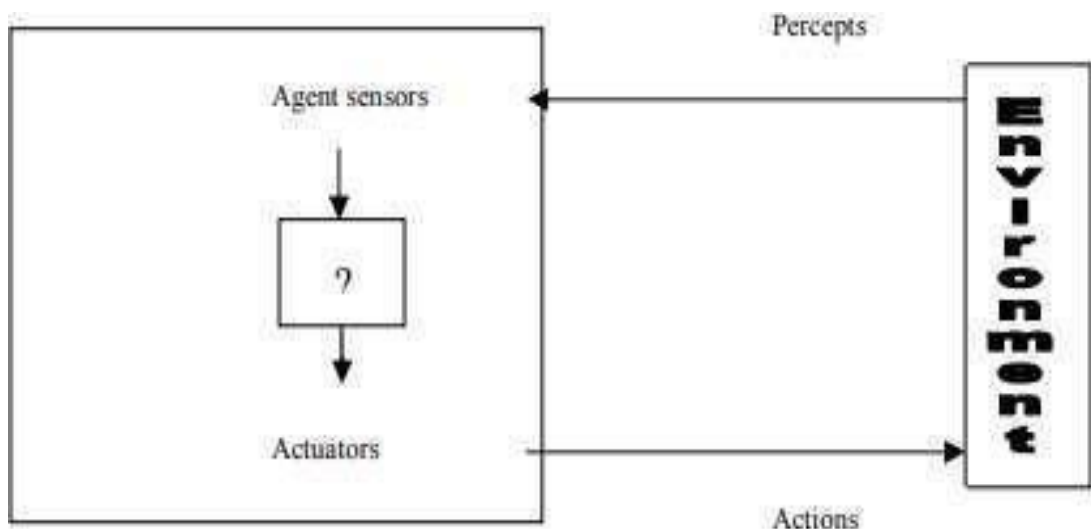
- ❖ The performance measure that defines degree of success.
- ❖ Ever thing that the agent has perceived so far We wil cal this complete perceptual history the percept sequence.
- ❖ When the agent knows about the environment.
- ❖ The action that the agent can perform.

14. Define an Omniscient agent. K1,CO1

An omniscient agent knows the actual outcome of its action and can act accordingly; but omniscience is impossible in reality.

15. Give the structure of agent in an environment K1,CO1

Agent interacts with environment through sensors and actuators. An Agent is anything that can be viewed as perceiving (i.e.) understanding its environment through sensors and acting upon that environment through actuators.



Object is a real- world entity, identifiably separate from its surroundings, has a well defined set of **attributes and a well-defined set of procedures or methods**. Object means a combination of data and logic that represents some real-world entity. (E.g.) Car is an object Color, manufacturer, cost, owner etc are attributes. Drive it, lock it, tow it, carry passengers in it are all methods.

16. Define Ideal Rational Agent. [May/June 2009][April/May 2015][Nov/Dec 2011] K2,CO1

For each possible percept sequence, an ideal rational agent should do whatever action is expected to maximize its performance measure on the basis of the evidence provided by the percept sequence & whatever built in knowledge that the agent has.

17. Why are condition-action rules important in the design of an agent?

K1,CO1

Rules are used to represent relationships. Rule-based knowledge representation employs IF condition (premise antecedent) THEN action statements. (goal consequent)

For example: IF the heating element glows AND the bread is always dark

THEN the toaster thermostat is broken

When the problem situation matches the IF part of a rule,

the action specified by the THEN part of the rule is performed.

18. List down the characteristics of intelligent agent. [APRIL/MAY 2017]

Internal characteristics are K1,CO1

- **Learning/reasoning:** an agent has the ability to learn from previous experience and to successively adapt its own behavior to the environment.
- **reactivity:** an agent must be capable of reacting appropriately to influences or information from its environment.
- **autonomy:** an agent must have both control over its actions and internal states. The degree of the agent's autonomy can be specified. There may need intervention from the user only for important decisions.
- **Goal-oriented:** an agent has well-defined goals and gradually influence its environment and so achieve its own goals.

External characteristics are

- **communication:** an agent often requires an interaction with its environment to fulfil its tasks, such as human, other agents, and arbitrary information sources.
- **cooperation:** cooperation of several agents permits faster and better solutions for complex tasks that exceed the capabilities of a single agent.
- **mobility:** an agent may navigate within electronic communication networks.
- **Character:** like human, an agent may demonstrate an external behavior with many human characters as possible.

19. What is the use of online search agents in unknown environment?

[Nov/Dec 2007] K1,CO1

Online search agents operate by interleaving computation and action: first it takes an action, and then it observes the environment and computes the next action. Online search is a good idea in dynamic or semi dynamic domains and stochastic domains. Online search is a necessary idea for an exploration problem, where the states and actions are unknown to the agent.

20. Define operationalization. K1,CO1

The process of creating a formal description of a problem using the knowledge about the problem, so as to create a program for solving a problem is called as operationalization.

21. List the major components in problem formulation in AI. K1,CO1

The four components are:

- ✿ Initial state.
- ✿ State Space
- ✿ Goal Test and Path Cost

22. Why problem formulation must follow goal formulation? [April/May 2015][Nov/Dec 2007] K1,CO1

Well goal formulation is used to steer the agent in the right direction, thus ignoring any redundant actions. Problem formulation must follow this because it is based off of Goal Formulation. It is the process of deciding what actions and states to consider given a certain goal. A goal may be set in stone, but how you achieve it can vary. Usually the most optimal way is chosen though.

23. What can AI do today? K1,CO1

Autonomous Planning and Scheduling

✿ Game Planning



✿ Autonomous Control

✿ Diagnosis, Logistics Planning, Robotics

24. What is a task environment? How it is specified? K1,CO1

- ✿ Task environments are essentially the "problems" to which rational agents are the "solutions" . A Task environment is specified using PEAS (Performance, Environment, Actuators, and Sensors) description
- ✿ **Performance measure** – evaluates the behaviour of the agent in an environment.
- ✿ **Environment** - Set of students testing Agency **Actuators** - Display exercises suggestions, corrections. **Sensors** - Keyboard entry

25. List the properties of task environments. K2,CO1

- ✿ Fully observable vs. partially observable.
- ✿ Deterministic vs. stochastic.
- ✿ Episodic vs sequential Static vs dynamic.
- ✿ Discrete vs. continuous. Single agent vs. multiagent.

26. What are the four different kinds of agent programs? K2,CO1

- ✿ Simple reflex agents;
- ✿ Model-based reflex agents;
- ✿ Goal-based agents; and
- ✿ Utility-based agents

27. What are utility based agents? K2,CO1

Goals alone are not really enough to generate high-quality behavior in most environments. For example, there are many action sequences that will get the taxi to its destination (thereby achieving the goal) but some are quicker, safer, more reliable, or cheaper than others. A utility function maps a state (or a sequence of states) onto a real number, which describes the associated degree of happiness.

28. What are the three classes of problem? K2,CO1

Ignorable, in which solution steps can be ignored.



Recoverable, in which solution steps can be undone.



✿ Irrecoverable, in which solution steps cannot be undone.

29.How can the performance of an agent are improved? K1,CO1

Performance of any agent can be improved by learning. Learning is a process that improves the knowledge of an Artificial intel igent program by making observations about its environment.

30.What are the steps involved to solve a problem in AI? K1,CO1

1. Define a state space.

2. Identify Initial States.

3. Specify goal states.

4. Specify set of rules.

31.List some of the uninformed search techniques. [APRIL/MAY 2017] K2

,C02

- Uninformed Search Techniques:
- Depth-first Search
- Breadth-first Search
- Iterative Deepening

12.PART B Q s (WITH K LEVEL AND CO) UNIT 1

1. What is an intelligent agent? Explain in detail about reflex and goal based agents. Illustrate the use of learning agents. (13) **K2,CO1**
2. Elaborate in detail about reflex and utility based agents and how learning agent works in any environment.(13) **K2,CO1**
3. Explain in detail about structure of typical intelligent agents with example.(13) **K2,CO1**
4. What is problem formulation? Formulate the water jug problem as an AI problem. Write the production rules for solving water jug problem and solve using those rules.(13) (15) **K2,CO1**
5. What are the properties of task environment? Explain in detail about the each property with examples.(8) **K2,CO1**
6. What is problem formulation? Formulate 8 puzzle problem as an AI problem. Analyze any 2 heuristics of 8 puzzle problem.(13)(15) **K2,CO1**
7. Write production rules for solving water jug problem.(13) **K2,CO1**
8. Explain problem solving approach to Typical AI problems ? Explain in detail any two AI problems? **K2,CO1**
9. Explain Breadth First Search Algorithm with the proper Example?K2,CO1
10. Elucidate Search Algorithms with Uninformed Search Strategies?K2,CO1

13. Supportive online Certification courses

1. Udacity: **Artificial Intelligence**

<https://www.udacity.com/course/ai-artificial-intelligence-nanodegree--nd898>

2. NPTEL: **Artificial Intelligence**

<https://nptel.ac.in/courses/106/102/106102220/>



14. Real time applications in day to day life and to Industry

✿ Boston Dynamic's Helping Robot

Here we see a robot dog unsure how to get past the door obstacle in front of it. Cue the entry of Boston Dynamic's helping robot. It walks in, opens the door, and steps aside to let the other robot inside.

<https://youtu.be/fUyU3IKzoio>

✿ Amazon's Warehouse Robots

In this video, take a look at how Amazon uses AI robots to pick out the orders and make the e-commerce giant even more ruthlessly efficient.

<https://youtu.be/Ox05Bks2Q3s>

15. ASSESSMENT SCHEDULE

Tentative schedule for the Assessment During 2021-2022 ODD semester

S.NO	Name of the Assessment	Start Date	End Date	Portion
1	Unit Test 1			UNIT 1
2	IAT 1	09.09.2023	15.09.2023	UNIT 1 & 2
3	Unit Test 2			UNIT 3
4	IAT 2	26.10.2023	01.11.2023	UNIT 3 & 4
5	Revision 1			UNIT 5 , 1 & 2
6	Revision 2			UNIT 3 & 4
7	Model	15.11.2023	25.11.2023	ALL 5 UNITS

16. PRESCRIBED TEXT BOOKS & REFERENCE BOOKS

TEXT BOOKS:

1. Peter Norvig and Stuart Russel, Artificial Intelligence: A Modern Approach, Pearson, Fourth Edition, 2020.
2. Bratko, Prolog: Programming for Artificial Intelligence, Fourth edition, Addison-Wesley Educational Publishers Inc., 2011.

REFERENCES:

1. Elaine Rich, Kevin Knight and B. Nair, Artificial Intelligence 3rd Edition, McGraw Hill, 2017.
2. Melanie Mitchell, Artificial Intelligence: A Guide for Thinking Humans. Series : Pelican Books, 2020
3. Ernest Friedman-Hill, Jess in action, Rule-Based Systems in Java, Manning Publications, 2003
4. Nils J. Nilsson, The Quest for Artificial Intelligence, Cambridge University Press, 2009
5. Dan W. Patterson Introduction to Artificial Intelligence and expert systems, 1st Edition by Patterson, Pearson, India, 2015

17. MINI PROJECT SUGGESTION

Tic-Tac-Toe using C Language

1. Objectives:

- Our project name is Tic-Tac-Toe game. This game is very popular and is fairly simple by itself. It is actually a two player game. In this game, there is a board with $n \times n$ squares. In our game, it is 3×3 squares. The goal of Tic-Tac-Toe is to be one of the players to get three same symbols in a row - horizontally, vertically or diagonally - on a 3×3 grid.

2. Overview:

- This game can be played in a 3×3 grid (shown in the figure 2.1). The game can be played by two players. There are two options for players:

1. Human 2. Computer

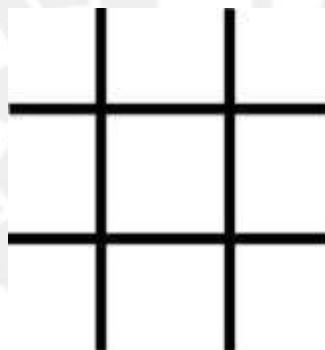


Figure: 2.1

1. Players:

- For the option human, both the players are human and for the option computer, the first player is human and the second player is computer.

2. Theory of Game:

A player can choose between two symbols with his opponent, usual games use "X" and "O". If first player choose "X" then the second player have to play with "O" and vice versa.

A player marks any of the 3x3 squares with his symbol (may be "X" or "O") and his aim is to create a straight line horizontally or vertically or diagonally with two intensions:

- a) Create a straight line before his opponent to win the game.
- b) Restrict his opponent from creating a straight line first.

In case logically no one can create a straight line with his own symbol, the game results a tie.

Hence there are only three possible results – a player wins, his opponent (human or computer) wins or it's a tie.

1	2	3
4	5	6
7	8	9

Figure: 2.2

If any player is able to draw three Xs or three Os in the following combinations then that player wins. The combinations are:

a) 1, 2, 3 b) 4, 5, 6

c) 7, 8, 9 d) 1, 4, 7

e) 2, 5, 8 f) 3, 6, 9

h) 1, 5, 9 i) 3, 5, 7

3. Core Logic - AI:

While making a Tic Tac Toe game using C language, it is important to make use of arrays. The Xs and Os are kept in different arrays, and they are passed between several functions in the code to keep track of how the game goes. With the code here you can play the game choosing either X or O against the computer.

This Tic Tac Toe C game is such that you will have to input a numerical character, from 1 to 9, to select a position for X or O into the space you want. For example: if you are playing with O and you input 2, the O will go to first row – second column. If you want to place O in third row – first column, you have to enter 7.

And, it is similar for the other positions.

1. Function Used:

I have divided this project into many functions, and below is a list of those functions.

```
void display(void);
```


```
void empty_board(void);
```

```
void input(int player);
```

```
int chkboard(void); //checks if any player has won
```

```
void counter_move(int player, int *best_i, int *best_j, int
```

```
*best_pos); void best_move(int player);
```



```
Turbo C - IDE

| |
| |
| |

Press Esc to exit
Player 1 move
Press arrow keys to move, then press <Enter> to place your mark
Press S for best move
```

```
Turbo C - IDE

X | |
| 0 |
| |

Press Esc to exit
Player 1 move
Press arrow keys to move, then press <Enter> to place your mark
Press S for best move
```

```
Turbo C - IDE

X | |
X | 0 |
0 | |

Press Esc to exit
Player 1 move
Press arrow keys to move, then press <Enter> to place your mark
Press S for best move
```



```
Turbo C++ IDE

X | O | X
-----
X | O | 
-----
O |  | 

Press Esc to exit
Player 1 move
Press arrow keys to move, then press <Enter> to place your mark
Press S for best move
```

```
Turbo C++ IDE

X | O | X
-----
X | O | O
-----
O | X | 

Press Esc to exit
Player 1 move
Press arrow keys to move, then press <Enter> to place your mark
Press S for best move
```

```
Turbo C++ IDE

X | O | X
-----
X | O | O
-----
O | X | X

Press Esc to exit
Draw
Play again (y for yes)
```

5. Conclusion

- ✿ In the conclusion of this project, I would like to say that C is a fun and easy programming language and while creating a project like this, it has not just been a good experience but it also helped in the development of my creativity and logical thinking. The program is working and I hope, it's also bug-free.

6. Future plan:

Keyboard functions will be added.

We want to design more complex boards for the game in future

Reference:

<https://www.codewithc.com/mini-project-in-c-tic-tac-toe-game/>



Thank you

Disclaimer:

This document is confidential and intended solely for the educational purpose of RMK Group of Educational Institutions. If you have received this document through email in error, please notify the system manager. This document contains proprietary information and is intended only to the respective group / learning community as intended. If you are not the addressee you should not disseminate, distribute or copy through e-mail. Please notify the sender immediately by e-mail if you have received this document by mistake and delete this document from your system. If you are not the intended recipient you are notified that disclosing, copying, distributing or taking any action in reliance on the contents of this information is strictly prohibited.