Artificial Intelligence

Chapter 1 Module I

What is Artificial Intelligence?

A broad field that means different things to different people

- ✓ Artificial intelligence (AI) is the study of how to make computers do things which, at the moment, people do better – **Rich and Knight, 1991**
- ✓ The study of the computations that make it possible to perceive, reason and act – **Patrick Henry Winston, 1992**
- ✓ AI is a branch of computer science deals with automation of intelligent behavior – **Luger & Stubblefield, 1993**
- ✓ A field of study that seeks to explain and emulate the intelligent behavior in terms of computational processes – **Schalkoff, 1990**
- ✓ The automation of activities that we associate with human thinking – **Bellman, 1978 Activities Means:** Decision making, problem solving, learning ….
- ✓ The study of techniques to make the computers to exhibit some kind of intelligence
- ✓ It is the science and engineering of making intelligent machines, especially intelligent computer programs.

It is related to the similar task of using computers to understand human intelligence
i.e., AI deals with the techniques to make computers to exhibit some kind of intelligence.

THE AI PROBLEMS

- ✓ Much of the **early work** in the field focused on formal tasks, such as game playing and theorem proving.
- ✓ **Samuel** wrote a checkers-playing program that not only played games with opponents but also used its experience at those games to improve its later performance. Chess also received a good deal of attention.
- ✓ The **Logic Theorist** was an early attempt to prove mathematical theorems. It was able to prove several theorems. **Gelernter's** theorem explored another area of **mathematics: geometry**. **Game playing** and **theorem proving** share the property that people who do them well are considered to be displaying intelligence.
- ✓ In spite of this, it appeared initially that computers could perform well at those tasks simply by being fast at exploring a large number of solution paths and then selecting the best one. It was thought that this process required very little knowledge and could therefore be programmed easily.

- ✓ Another early work in AI focused is on the sort of problems solving **called commonsense reasoning**. It includes reasoning about physical objects and their relationships to each other. To investigate this sort of reasoning, **Newell, Shaw, and Simon** built the **General Problem Solver (GPS)** - here only simple tasks were selected.
- ✓ AI research progressed and techniques for handling larger amounts of world knowledge were developed. These include **perception – vision and speech, natural language understanding, and problems solving** in specialized domains such as medical diagnosis and chemical analysis and are referred as **Mundane tasks.**
- ✓ In addition to mundane tasks, people can also perform specialized tasks in which carefully acquired expertise is necessary. Examples of such tasks include **engineering design, scientific discovery, medical diagnosis** and **financial planning. Figure below** lists some of the tasks that are targets of work in AI.
- ✓ As a result, the problem areas where AI is now flourishing most as a practical discipline are primarily the domains that require only specialized expertise without the assistance of commonsense knowledge.

Some of the Task Domains of AI

- ✓ **Mundane Tasks**
  - **Perception**
    - Vision
    - Speech
  - **Natural Languages**
    - Understanding
    - Generation
    - Translation
  - **Common sense reasoning**
  - **Robot Control**
- ✓ **Formal Tasks**
  - **Games Playing**
    - Chess
    - Backgammon
    - Checkers - Go
  - **Mathematics**
    - Geometry
    - Logic
    - Integral calculus

- Theorem Proving
- General Problem Solving

✓ **Expert Tasks (require specialized skills and training)**

- **Engineering**
  - Design
  - Fault finding
  - Manufacturing planning
- **Scientific Analysis**
- **Medical Diagnosis**
- **Financial Analysis**

Note: AI is concerned with automating both **Mundane** and **Expert tasks**

➢ **THE UNDERLYING ASSUMPTION**

The heart of research in artificial intelligence lies what Newell and Simon [1976] call the physical symbol system hypothesis. They define a Physical symbol system as follows:

- ✓ A physical symbol system consist of a **set of entities** called **symbols**, which are **physical patterns** that can occur as components of another type of entity called an **expression** or **symbol structure**. Thus, a symbol structure is composed of a number of **instances** or **tokens** of symbols related in some physical way. At any instant of time the system will contain a collection of these **symbol structures**.
- ✓ In addition to these structures, the system also contains a **collection of processes** that operate on **expressions** to produce other expressions: **processes of creation, modification, reproduction** and **destruction.** A physical symbol system is a **machine** that produces through time, an evolving collection of symbol structures.
- ✓ The physical Symbol System Hypothesis can be stated as – **A physical symbol system has the necessary and sufficient means for general intelligent action.**
- ✓ The **truth of this hypothesis** can be determined only by **experimentation**. Computers provide the perfect medium for this experimentation since they can be programmed to simulate any physical symbol system. This ability of computers to serve as arbitrary symbol manipulators was noticed very early in the history of computing by **Lady Lovelace** about **Babbage's** proposed Analytical Engine in 1842.
- ✓ The operating mechanism can even be thrown into action independently of any object to operate upon. As it has become increasingly easy to build computing machines, so it has become increasingly possible to conduct empirical investigations of the physical symbol system hypothesis. In each such investigation, a particular task that might be regarded as requiring intelligence is selected. A program to perform the task is proposed and then tested. We have not been completely successful at creating programs that perform all the selected tasks.

- ✓ Evidence in support of the physical symbol system hypothesis has come not only from areas such as game playing, but also from areas such as visual perception, where it is more tempting to suspect the influence of sub-symbolic processes. However, sub-symbolic models such as **neural networks** are beginning to challenge symbolic ones at such low-level tasks.

➢ WHAT IS AN AI TECHNIQUE?

Artificial intelligence problems span a very broad spectrum. There are techniques that are appropriate for the solution of a variety of these problems.

One of the few hard and fast results to come out of the first three decades of AI research is that **intelligence requires knowledge**. The knowledge **possesses some properties** as follows:

- ✓ It is voluminous
- ✓ It is hard to characterize accurately
- ✓ It is constantly changing
- ✓ It differs from data being organized in a way that corresponds to the ways it will be used

Here, it is concluded that an AI technique is a method that **exploits knowledge** that should be **represented** in such a way that:

- ✓ **The knowledge captures generalizations** means that, it is not necessary to represent separately each individual situation. Instead, situations that share important properties are grouped together. If **knowledge does not have this property**, excessive amounts of memory and updating will be require, then called as **data** rather than **knowledge**.
- ✓ **It can be understood by people who must provide it**. Although for many programs, the bulk of the data can be acquired automatically, for example by **taking readings from a variety of instruments**.
- ✓ It can easily be **modified** to **correct errors** and to **reflect changes** in the world and in our world view.
- ✓ It can be used in many **situations** even if it is **not totally accurate or complete**.
- ✓ It can be used to help overcome its own **complete volume** by helping to narrow the range of possibilities.

Although AI techniques must be designed in keeping with these constraints imposed by Al problems, there is some degree of independence between problems and problem-solving techniques. It is possible to solve Al problems without using AI techniques and it is possible to apply AI techniques to the solution of non-Al problems.

In order to **characterize AI techniques** as **problem-independent way as possible**, let us consider some problems.

- **Tic-Tac-Toe problem**

Here, presented a series of three programs to play tic-tac-toe. The programs in this series increase in:

- ✓ The clarity of their knowledge
- ✓ Their complexity
- ✓ Their use of generalizations
- ✓ The extensibility of their approach. Thus, they move toward being representations of what we call AI techniques.

Program 1

Data Structures:

Board A nine-element vector representing the board, where the elements of the vector correspond to the board positions as follows:

1	**2**	**3**
4	**5**	**6**
7	**8**	**9**

An element contains the **value 0** if the corresponding **square is blank**. **1** if it is filled with an X, or **2** if it is filled with an O.

Movetable A large vector of 19,683 elements (3^9), each element of which is a nine-element vector.

The contents of this-vector are chosen specifically to allow the algorithm to work.

Algorithm:

To make a move, do the following:

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

Comments:

This program is very efficient in terms of time. And, in theory, it could play an optimal game of tic-tac-toe. But it has several disadvantages:

- ✓ It takes a lot of space to store the table of each move
- ✓ Have to do a lot of work specifying all the entries in the **movetable.**
- ✓ It is very unlikely that all the required **movetable** entries can be determined and entered without any errors.

- ✓ If we want to extend the game, say to **three dimensions,** we would have to start from scratch, since 3^{27} board positions would have to be stored, occupies large memory.

Program 2

Data Structures:

Board A nine-element vector representing the board, as described for Program 1. But instead of using the numbers 0, 1 or 2 in each element, we store **2 is blank, 3 for X and 5 for O.**

Turn An integer indicating which move of the game is about to be played; 1 indicates the first move, 9 the last.

Algorithm:

The main algorithm uses three sub-procedures:

Make 2 Returns 5 if the **center square of the board is blank**, that is, if Board [5] = 2. Otherwise, this function returns any blank Noncorner Square – **2, 4, 6, or 8.**

Posswin (p) Returns 0 if player p cannot win on his next move; otherwise, it returns the **number of the square** that constitutes a winning move. This function will enable the program both to **win** and to **block the opponent's win**. Posswin operates by checking, one at a time, each of the rows, columns, and diagonals. Because of the way values are numbered it can test an entire row/column/diagonal to see if it is a possible win by multiplying the values of its squares together. If the product is **18** (3 x 3 x 2), then X can win (**means one of the squares is empty).** If the product is **50** (5 x 5 x 2), then O can win **(means one of the squares is empty)**. If we find a winning row, we determine which element is blank, and return the **number of that square**.

Go (n) Makes a move in square n. This procedure sets **Board[n] to 3 if Turn is odd,** or **5 if Turn is even.** It also increments **Turn by one.**

The algorithm has a built-in strategy for each move. It makes the odd-numbered moves if it is playing X, the even-numbered moves if it is playing O. The strategy for each turn is as follows:

Turn=1 Go(1) (upper left corner).

Turn=2 If Board [5] is blank, Go(5), else Go(1).

Turn=3 If Board [9] is blank, Go(9), else Go(3).

Turn=4 If Posswin(X) is not 0, then Go(Posswin(X)) [i.e., block opponent'swin],else Go(Make2).

Turn= 5 If Posswin(X) is not 0 then Go(Posswin(X))[i.e., win] else if Posswin(O) is not 0, then Go(Posswin(O)) [i.e., block win], else if Board[7] is blank, then Go(7), else Go(3). [Here the program is trying to make a fork.]

Turn=6 If Posswin(O) is not 0 then Go (Posswin(O)), else if Posswin(X) is not 0, then Go(Posswin(X)), else Go(Make2).

Turn=7 If Posswin(X) is not 0 then Go(Posswin(X)). else if Posswin(O) is not 0, then Go(Posswin(O)). **Else** go anywhere that is blank.

Turn=8 If Posswin(O) is not 0 then Go(Posswin(O)), else if Posswin(X) is not 0, then Go(Posswin(X)), else go anywhere that is blank.

Turn=9 Same as Turn=7.

Program 2 – A

This program is identical to program 2 except for the one change in the representation of the board and is as follows:

8	**3**	**4**
1	**5**	**9**
6	**7**	**2**

Here, numbering of the board produces a **magic square**: all the **rows, columns**, and **diagonals sum up to 15.** This means that we can simplify the process of checking for a possible win. In addition to marking the board as moves are made, we keep a list, for each player, of the squares in which he or she has played. To check for a possible win for one player, we consider **each pair of squares owned by that player** and **compute the difference between 15 and the sum of the two squares**. If this difference is **not positive** or **if it is greater than 9**, then the original two squares were **not collinear** and so can be ignored. Otherwise, if the square representing the difference is blank **(1 to 9 squares)**, a move there will **produce a win**. This shows how the choice of representation can have a major impact on the efficiency of a problem-solving program.

Program 3

Data Structures

BoardPosition: A structure containing a nine-element vector representing the board, a list of board positions that could result from the next move, and a number representing an estimate of how likely the board position is to lead to an ultimate win for the player to move.

Algorithm:

To decide on the next move, look ahead at the board positions that result from each possible move. Decide which position is best (as described below), make the move that leads to that position, and assign the rating of that best move to the current position.

To decide which of a set of board positions is best, do the following for each of them:

1. See if it is a win. If so, call it the best by giving it the highest possible rating.
2. Otherwise, consider all the moves the opponent could make next. See which of them is worst for us (by recursively calling this procedure). Assume the opponent will make that move. Whatever rating that moves has, assign it to the node we are considering.
3. The best node is then the one with the highest rating.

This algorithm will look ahead at various sequences of moves in order to find a sequence that leads to a win. It attempts to maximize the likelihood of winning, while assuming that the opponent will try to minimize that likelihood. This algorithm is called the **minimax procedure.**

- **Question Answering problem**

Here, programs that **read in English text** and then answer questions also **stated in English**, about that text. This is more difficult to state formally and precisely what our problem is and what constitutes correct solutions to it. For example, suppose that the input text were just the single sentence

Example 1:
Russia massed troops on the Czech border.

Then either of the following question-answering dialogues might occur with the POLITICS program:

Dialogue 1

Q: **Why did Russia do this?**
A: Because Russia thought that it could take political control of Czechoslovakia by sending troops.
Q: **What should the United States do?**
A: The United States should intervene (interfere) militarily.

Dialogue 2
Q: **Why did Russia do this?**
A: Because Russia wanted to increase its political influence over Czechoslovakia.
Q: **What should the United States do?**
A: The United States should denounce (condemn) the Russian action in the United Nations.

✓ In the POLITICS program, answers were constructed by considering both the **input text** and a **separate model of the beliefs** and **actions of various political entities**, including Russia. When the model is changed, the system's answers also change.

)

- ✓ The general point here is that defining what it means to produce a correct answer to a question may be very hard.

Example 2:

Mary went shopping for a new coat. She found a red one she really liked. When she got it home, she discovered that it went perfectly with her favorite dress.

Attempt to answer each of the following questions with each program:

Q1: What did Mary go shopping for?
Q2: What did Mary find that she liked?
Q3: Did Mary buy anything?

Program 1

This program is to attempt to answer questions using the literal text. It simply matches text fragments in the questions against the input text.

Data structures

Question patterns: A set of **templates** that match common question forms and produce patterns to be used to match against inputs. Templates and patterns are paired so that if a template matches successfully against an input question then its associated text patterns are used to try finding appropriate answers in the text. **For example**, if the template "**who did x y**" matches an input question, then the text patterns "**x, y, z**" is matched against the text and the value of **z** is given as the answer to the question.

Text The input text stored simply as a long character string.
Question The current question also stored as a character string.

Algorithm
To answer a question, do the following:

1. Compare each element of **Question Patterns, the question** and all those that match successfully to generate a set of text patterns.
2. Pass each of these patterns through a **substitution process** that generates alternative forms of verbs so that, for example, "**go**" in a question might match "**went**" in the text. This step generates a new, expanded set of **text patterns**.
3. Apply each of these text patterns to text, and collect all the resulting answers.
4. Reply with the set of answers just collected.

Answers:

Q1: The template "**what did x y**" matches this question and generates the text pattern "**Mary go shopping for z.**" After the pattern-substitution step, this pattern is expanded to a set of patterns including "**Mary goes shopping for z,**" and "**Mary went shopping for z.**" the latter pattern matches the input

)

text; the program, using a convention that variables match the longest possible string up to a sentence delimiter (such as a period), assigns **z** the value, **"a new coat,"** which is given as the answer.

Q2: Unless the template set is very large, allowing for the insertion of the object of **"find"** between it and the modifying phrase **"that she liked,"** the insertion of the word **"really"** in the text, and the substitution of **"she"** for **"Mary,"** this question is hot answerable. If all of these variations are accounted for and the question can be answered, then the response is **"a red one".**

Q3: Since no answer to this question is contained in the text, no answer will be found.

Program 2

This program first converts the input text into a structured internal from that attempt to capture the meaning of the sentences. It also converts questions into that form. It finds answers by matching structured forms against each other.

Data structures

EnglishKnow: A description of the words, grammar, and appropriate semantic interpretations of a large enough subset of English to account for the input texts that the system will see. This knowledge of English is used both to map input sentences into an internal, meaning-oriented form and to map from such internal forms back into English. The former process is used when English text is being read; the latter is used to generate English answers from the meaning-oriented from that constitutes the program's knowledge base.

Inputtext The input text in character form.

StructuredText A Structured representation of the content of the input text. This Structure attempts to capture the essential Knowledge contained in the text, independently of the exact way that the Knowledge was stated in English

A Structured Representation of a sentence:

Event 2

Instance:	Finding
Tense:	Past
Agent:	Mary
Object:	Thing 1

Thing 1

Instance:	Coat
Color:	Red

Event 2

Instance:	Liking
Tense:	Past
Modifier:	Much
Object:	Thing 1

InputQuestion The input question in character form.

StructQuestions A structured representation of the content of the user's question. The structure is the same as the one used to represent the content of the input text.

Algorithm:

Convert the input text into structured form using the Knowledge contained in English Know. This may require considering several different potential structures, for a variety of reasons, including the fact that English words can be ambiguous, English grammatical structures can be ambiguous, and pronouns may have several possible antecedents. Then, to answer a question, do the following:

1. Convert the question to structured form, again using the Knowledge contained in English Know. Use some special marker in the structure to indicate the part of the structure that should be returned as the answer. This marker will often correspond to the occurrence of a question word (like "who" or "what") in the sentence. The exact way in which this marking gets done depends on the form chosen for representing structured Text.
2. Match this structured form against Structured Text.
3. Return as the answer those parts of the text that match the requested segment of the question.

Answers:
Q1: This question is answered straightforwardly with, "a new coat"
Q2: This one also is answered successfully with, "a red coat".
Q3: This one, though, cannot be answered, since there is no direct response to it in the text.

Program 3

This program converts the input text into a structured form that contains the meanings of the sentences in the text, and then it combines that form with other structured forms that describe prior knowledge about the objects and situations involved in the text. It answers questions using this augmented knowledge structure.

Data structures

World Model: A Structured representation of background world Knowledge. This structure contains Knowledge about objects, actions and situations that are described in the input text. This Structure is used to construct Integrated Text from the input text. For example, **Figure below** shows an example of a Structure that represents the systems Knowledge about shopping. In the case of this text, for example, M is a coat and M' is a red coat. Branches in the figure describe alternative paths through the script.

)

The Algorithm:

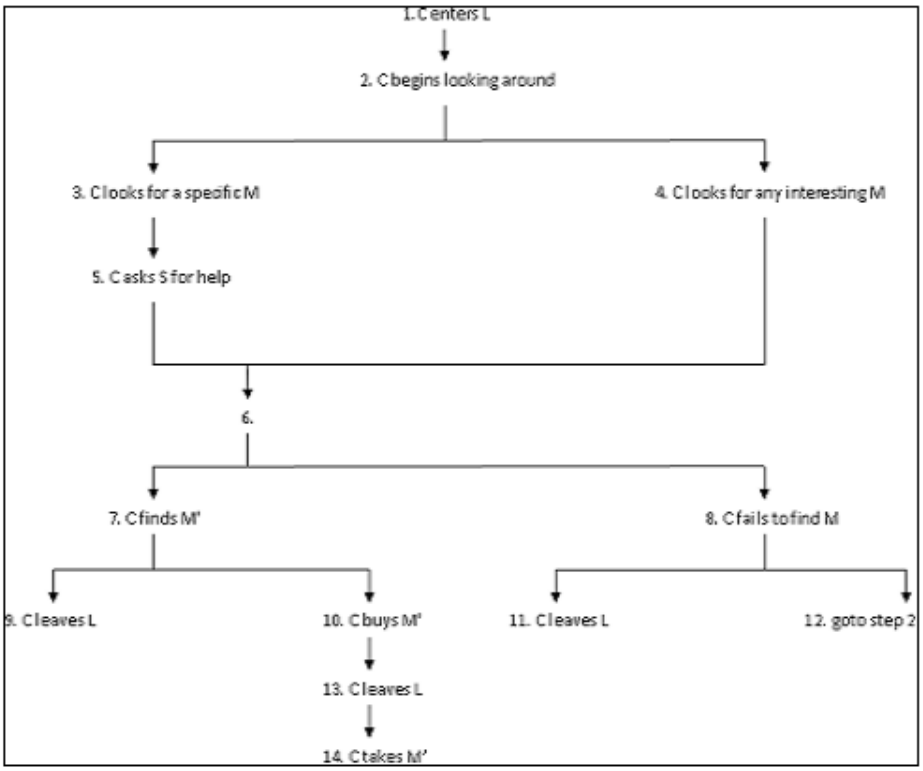
Convert the input Text into structured form using both the Knowledge contained in English Know and that contained in world model. The number possible structures will usually be greater now than it was in program 2 because so much more Knowledge is being used. Sometimes, though, it may be possible to consider fewer possibilities by using the additional knowledge to filter the alternatives.

Shopping script:
Roles: C (customer), S (salesperson)
Props: M (merchandise), D (dollars)
Location: L (a store)

To answer a question, do the following:

1. Convert the question to structured form as in program 2 but use WorldModel necessary to resolve any ambiguities that may arise.
2. Match this structured form against Integrated Text.
3. Return as the answer those parts of the text that match the requested segment of the question.

A Shopping Script diagram



Answers:

Q1: Same as Program 2.

Q2: Same as Program 2.

Q3: Now this question can be answered. The shopping script is instantiated for this text, and because of the last sentence, the path through step 14 of the script is the one that is used in forming the representation

)

of the text. When the script is instantiated M^1 is bound to the structure representing the red coat. After the script has been instantiated, Integrated Text contains several events. That are taken from the script but that are not described in the original text, including the event "Mary buys a red coat" (from step 10 of the script). Thus, using the integrated text as the basis for question answering allows the program to respond **"she bought a red coat".**

We can conclude that these problems illustrate important AI techniques:

- ✓ Search- Provides a way of solving problems for which no more direct approach is available.
- ✓ Use of Knowledge- Provides a way of solving complex problems by exploiting the structures of the objects that are involved.
- ✓ Abstractions – Provides a way of separating important features and variations from the many unimportant ones.

➢ THE LEVELS OF THE MODEL / (STATE OF THE ART)

We must ask ourselves, **what is our goal in trying to produce programs that do the intelligent things that people do?** Or, are **we trying to produce programs that do the tasks the same way people do?** Or, **are we attempting to produce programs that simply do the tasks in whatever way appear easiest?** There have been **AI projects** motivated by each of these goals.

These programs are divided into **two classes** –

- ✓ Programs in the **first class** attempt to solve problems that do not really fit definition of an AI task. They are problems that a computer cloud easily solves.
- ✓ Example is **Elementary Perceiver and Memorizer (EPAM) [Feigenbaum, 1963],** which memorized associated pairs of nonsense syllables. Memorizing pairs of nonsense syllables is easy for a computer. But this task is hard for people.
- ✓ The programs in the **second class** attempt to model **human performance** and are **within AI definitions**. Reasons for this are:

  → **To test psychological theories of human performance:** An example of a program that was written for this reason is PARRY [Colby, 1975], which exploited a model of human paranoid behavior to simulate the conversational behavior of a paranoid (Suspicious) person. Themodel was good enough that when several psychologists were given the opportunity to converse with the program via a terminal, they diagnosed its behavior as paranoid.

  → **To enable computers to understand human reasoning:** For example, for a computer to be able to read a newspaper story and then answer a question, such as **"why did the terrorists kill the hostages?"** Its program must be able to simulate the reasoning processes of people.

→

)

To enable people to understand computer reasoning: In many circumstances, people are reluctant to rely on the output of a computer unless they can understand how the machine arrived at its result. If the computer's reasoning process is similar to that of people, then producing an acceptable explanation is much easier.

→ **To exploit what knowledge we can glean (Gather) from people**: Since people are the best-known performers of most of the tasks with which we are dealing, it makes a lot of sense to look to them for clues as to how to produce.

- **The following are the disciplines that contributed ideas, new points, and techniques to AI:**
  - ✓ **Philosophy: (428BC- Present)**
    1. Can formal rules be used to draw valid conclusions?
    2. How does the mental mind arise from a physical brain?
    3. Where does knowledge come from?
    4. How does knowledge lead to action?
  - ✓ **Mathematics**
    1. What are the formal rules to draw valid conclusions?
    2. What can be computed?
    3. How do we reason with uncertain information?
  - ✓ **Economics**
    1. How should we make decisions so as to maximize pay off?
    2. How should we do this when others may not go along?
    3. How should we do this when the pay off may be far in the feature?
  - ✓ **Neuroscience**
    How do brains process information?
  - ✓ **Psychology**
    How do humans and animals think & act?
  - ✓ **Computer Engineer**
    How can we an efficient as computer?
  - ✓ **Computer Theory and Cybernetics**
    How can computer operate under their own control?
  - ✓ **Linguistic**
    How does language relate to thought?

Note: The questions under each of the above foundations, shows that, what exactly their contributions

- **The History / Early works of AI:**

  - ✓ The first work in AI was done by McCalloch and Pitts (1943), who proposed artificial neurons. They worked on three sources:
    1. Knowledge of the basic Physiology.
    2. Functions of neurons in the brain.

)

1. A formal analysis of propositional logic and turing's theory of computations.

- ✓ Later in 1949, **Donald Hebb** demonstrated a simple updating rule for modifying the connection strengths between neurons, called **Hebbbian Learning.**
- ✓ **Alan Turing** (1950), was the first to articulate the complete vision of AI in his article (1950) – **"Computing Machinery and Intelligence",** he introduced the **Turing Test**, **Machine learning, genetic algorithms and reinforcement learning.**
- ✓ John McCarthy of Dart Month College called the father of AI, introduced automata theory, neural networks, and the study of intelligence (1956). Also he designed the language **LISP** in 1958.
- ✓ Newell and Simon (1976) designed a program called General Problem Solver (GPS) that imitated human problem solving protocols. GPS was the first program to embody the **"Thinking humanly"** approach.
- ✓ Rosenblatt (1962) developed perceptrons by enhancing the Hebban learning algorithm.
- ✓ **DENDRAL** and **MYCIN** were the two expert systems – one for Molecule structure analysis and the other for diagnose the blood infections, developed during 1969, at MIT.
- ✓ Recent years have seen a revolution in both the content and the methodology of work in AI.
- ✓ Complete **Agent architecture** proposed during 1990 aims to understand the workings of agents embedded real environments, with continuous sensory inputs. One of the most important environments for intelligent agents in the Internet.
- ✓ AI Systems have become so common in **web-based applications** including search engines, recommended systems, and website construction systems.

- **What can AI do today?**

These are just examples of AI systems that exist today.

- ✓ **Autonomous Planning and scheduling:** NASA'S Remote Agent program became the first **on board autonomous planning program** to control the scheduling of operations for a spacecraft (Jonsson et.al., 2000), such as detection, diagnosis and recovery from problems as that occurred.
- ✓ **Game Planning:** IBM'S **Deep Blue,** Became the first computer to defect the world champion in chess match when it beated Garry Karpasov (1997), the value of IBM'S stock increased by $18 billion.
- ✓ **Autonomous Control:** The ALVINN computer vision system was trained to steer a car to keep it following a lane (driving autonomously 98% of the time from Pittsburgh to San Diego, 2850 miles)
- ✓ **Diagnosis:** Medical diagnosis programs based on probabilistic analysis have been able to perform at the level of an expert physician is several areas of medicine.
- ✓ **Logistics Planning:** During the **Persian Gulf crisis** of 1991, US forces deployed a **Dynamic Analysis and Re-planning Tool, (DART)** in1994, to do automated logistics planning and scheduling for transportation. This involved up to 50,000 vehicles, cargo, and people at a time, and had to account for starting points, destinations, routes and conflict resolution among all parameters.

)

- ✓ **Robotics:** Many surgeries now use robot assistants in microsurgery. Ex: HIPNAV (1996) is a system that uses computer vision techniques to create a three- dimensional model of a patient's internal anatomy and the uses robotic control to guide the insertion of a hip replacement prosthesis.
- ✓ **Language Understanding and Problem Solving: PROVERB (1999)** is a program that solves crossword puzzles setter than most humans.

➢ CRITERIA FOR SUCCESS

- ✓ One of the most important questions to answer in any scientific or engineering research project is **"How will we know if we have succeeded?"** Artificial intelligence is no exception. How will we know if we have constructed a machine that is intelligent? Can we do anything to measure our progress? **Yes**
- ✓ In 1950, Alan Turing proposed the following method for determining whether a machine can think. His method has since become known as the **Turing Test.** To conduct this test, **we need two people** and **the machine** to be evaluated. One person plays the role of the **interrogator,** who is in a **separate room** from the computer and the other person. The interrogator can ask questions of either the person or the computer by typing questions and receiving typed responses. However, the interrogator knows them only as **A and B** and aims to determine which the person is and which the machine is. The goal of the machines is to fool the interrogator into believing that it is the person. If the machine succeeds at this, then we will conclude that the **machine can think**. The machine is allowed to do whatever it can to fool the interrogator. So, for example, if asked the questions "How much is 12,324 times 73,981?" it could wait several minutes and then respond with the wrong answer [Turing, 1963].
- ✓ We are forced to conclude that the question of whether a **machine has intelligence** or **can think** is too vague to answer precisely. But it is often possible to construct a computer program that meets some performance standard for a particular task. That does not mean that the program does the task in the best possible way. It means only that we understand at least one way of doing at least part of a task.

SOME GENERAL REFERENCES

There is a great many sources of information about artificial intelligence.

- ✓ **First, will have some survey books:** The broadest are the multi-volume handbook of artificial intelligence [Barr et al.. 1981] and Encyclopedia of artificial intelligence [Shapiro and Eckroth, 1987], both of which contain articles on each of the major topics in the field.
- ✓ Four other books that provide good overviews of the field are artificial intelligence [Winston, 1984], introduction to artificial intelligence [Charniak and McDermott, 1985], Logical Foundations of artificial intelligence [Genesereth and Nilsson, 1987], and The Elements of artificial intelligence [Tanimoto, 1987] of more restricted scope is principles of artificial

intelligence [Nilsson, 1980], which contains a formal treatment of some general- purpose AI techniques.

- ✓ Most the work conducted in AI has been originally reported in **journal articles, conference proceedings or technical reports**. But some of the most interesting of these papers have later appeared in special collections published as books. Computer and Thought [Feigenbaum and Feldman, 1963] is a very early collection of this sort. Later ones include Simon and Siklossy[1972], Schank and Colby [1973], Bobrow and Collins [1975], waterman and Hayes- Roth[1978], Findler [1979],Webber and Nilsson [1981], Halpern[1986], Shrobe [1988], and several others that are mentioned in later chapter in connection with specific topics. For newer AI paradigms the book fundamentals of the new artificial intelligence [Toshinori Munakata, 1998] is a good one.
- ✓ The major journal of AI research is called simply **Artificial Intelligence**. In addition, Cognitive science is devoted to papers dealing with the overlapping areas of psychology, linguistics, and artificial intelligence. AI magazine is a more ephemeral, less technical magazine that is published by the American Association for artificial intelligence (AAAI). IEEE Expert, IEEE Transactions on Systems, Man and Cybernetics, IEEE Transactions on Neural Networks and several other journals publish papers on a board spectrum of AI application domains.
- ✓ Since 1969, there has been a major AI conference, the International Joint Conference on Artificial Intelligence (IJCAI), held every two years. The proceedings of these conferences give a good picture of the work that was taking place at the time. The other important AI conference, held three out of every four years starting in 1980, is sponsored by the AAAI, and its proceedings, too, are published.
- ✓ In addition of these general references, there exists a whole array of papers and books describing individual AI projects.

➢ ONE FINAL WORD AND BEYOND

Solving the problems is the topic of discussion in AI. We need methods to help us solve AI's serious dilemma:

- ✓ An AI system must contain a lot of knowledge if it is to handle anything
- ✓ But as the amount of knowledge grows, it becomes harder to access the appropriate things when needed, so more knowledge must be added to help. But now there is even more knowledge to manage, so more must be added, and so forth.
- ✓ AI is still young discipline possibly in the sense that little has been achieved as compared to what was expected.
- ✓ Robots form the ultimate test-bed for AI. Finally one should not forget that research in AI is multidisciplinary.

Chapter 2

PROBLEMS, PROBLEM SPACES, AND SEARCH

To build a system to solve a particular problem, we need to do four things:

1. **Define the problem precisely:** This definition must include precise specifications of what the initial situation (s) will be as well as what final situations constitute acceptable solutions to the problem.
2. **Analyze the problem:** A few very important features can have an immense impact on the appropriateness of various possible techniques for solving the problem.
3. **Isolate and represent the task knowledge** that is necessary to solve the problem.
4. **Choose the best problem-solving technique(s)** and apply it (them) to the particular problem.

➢ DEFINING THE PROBLEM AS A STATE SPACE SEARCH

- **AI Problem Solving**
  - ✓ **Formal Description of a Problem**
    - Identity the **objects /entities** involved in the problem.
    - Specify the **Initial State** (status) of the objects. i.e., the initial configurations / situations from where the problem solving process starts.
    - Specify the goal state that would be acceptable as solution to the problem. i.e, one or more states / configurations acceptable as solution states.
    - Define the state space that contains all possible states / configurations / situations of the objects either in the form of **tree or graph.**
    - Specify the set of rules/productions that describe available **actions (operators)** to get from one state to another.
    - Define search Techniques that systematically consider all possible action sequences to find a path from the **initial** to **goal state.**

Also think of: what **knowledge** is given (restrictions / constraints), knowledge to be **assumed,** additional **assumptions to be made** and order of applying rules (control strategy).

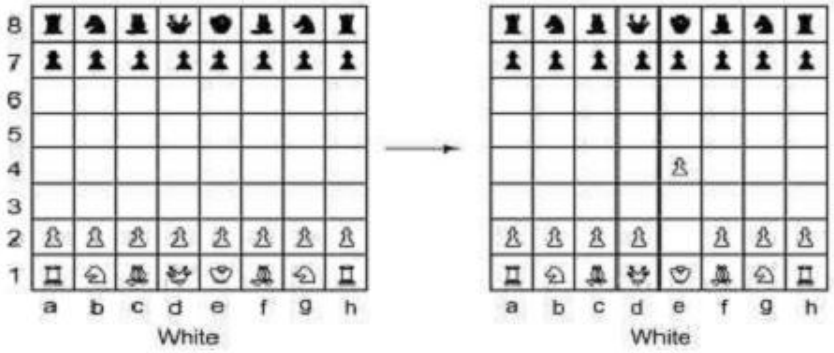
Note: AI Problem Solving → **Searching for a goal state.**

Example Problems:

1. **Playing Chess**
   - ✓ Each position can be described by an 8x8 array.
   - ✓ Initial Position is the game opening position.
   - ✓ **Goal Position** is any position in which the opponent does **not have a legal move** and his/her king is under attack, (check made).
   - ✓ **Legal Moves** can be described by a set of rules.

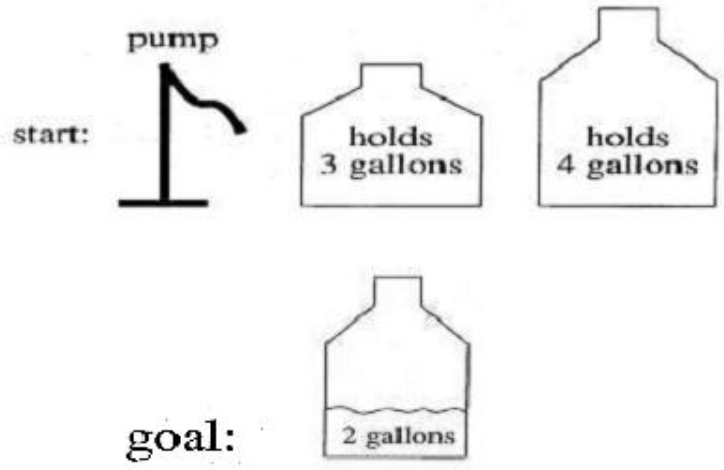
- ✓ Left side of the rule matched against the current state.
- ✓ Right side of the rule describes the new resulting state.
- ✓ **State space** is the set of legal positions.
- ✓ Starting at the initial state, using the set of rules to move from one state to another and attempting to end up in a goal state is the procedure for solving problem.

One Legal Chess Move



2. Water- Jug Problem

"You are given two jugs, a 4-litre one and a 3-litre one. Neither have any measuring markers on it. There is a pump that can be used to fill the jugs with water. How can you get exacting 2 liters of water into 4- litre jug?"



Solution: objects water jugs (2) pump let 'x' represents amount of water in 4-litre jug 'y' 3 litre jug.

→ **State** represents **pair (x,y)** where x = 0, 1, 2, 3, or 4

y = 0, 1, 2, or 3

x- Amount of water in jug 1(4-litre)y- jug 2 (3-litre)

➔

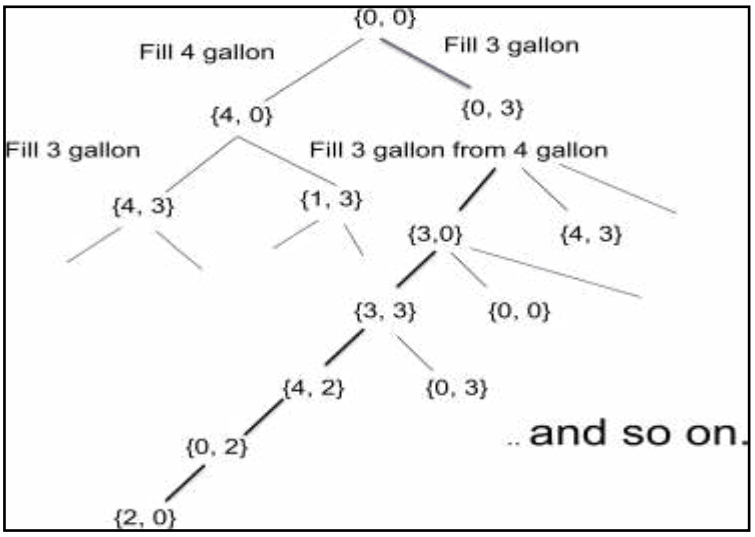
Initial / starting state: **(0,0)**
➔ Goal / final state: **(2, n)** for any 'n'
➔ Rules / productions / operators

1. (x,y) / if x < 4 ➔ (4, y) **Fill jug 1**
2. (x,y) / if y < 3 ➔ (x,3) **Fill jug 2**
3. (x,y) / if x > 0 ➔ (x-d, y) **Pour some water out of jug 1**
4. (x,y) / if y > 0 ➔ (x, y-d) **Pour some water out of jug 2**
5. (x,y) / if x > 0 ➔ (0, y) **Empty jug 1 on ground**
6. (x,y) / if y > 0 ➔ (x, 0) **Empty jug 2 on ground**
7. (x,y) / if y >0 and x+y = 4 ➔ (4, y-(4-x)) **Pour some water from jug 2 to jug 1 till jug 1 is full**
8. (x,y) / if x > 0 and x+y = 3 ➔ (x-(3-y), 3)) **Pour some water from jug 1 to jug 2 till jug 2 is full**
9. (x,y) / if y > 0 && x+y = 4 ➔ (x+y, 0) **Pour complete water from jug 2 to jug 1**
10. (x,y) / if x > 0 && x+y = 3 ➔ (0, x+y) **Pour complete water from jug 1 to jug 2**
11. (0,2) ➔ (2, 0) **Special rules to capture knowledge at**
12. (2,y) ➔ (0, y) **some stage in solving a problem**

One Solution to the Water Jug Problem

Gallons in the 4-Gallon Jug	Gallons in the 3-Gallon Jug	Rule Applied
0	0	2
0	3	9
3	0	2
3	3	7
4	2	5 or 12
0	2	9 or 11
2	0	

➔ **Solution Space Tree / State Space Tree**



Note: Starting from (0, 0), search and loop until it reaches the goal state (2, 0). Apply a rule whose left side matches the current state and set the new current state to be the resulting state.

2.1 PRODUCTION SYSTEMS (Rule based systems) or (Inferential systems)

- ✓ Since search forms the core of many intelligent process, it is useful to structure AI programs in a way that facilities describing the search process. So, production systems provide the structure for AI programs that makes searching process easy. i.e., production systems help in describing and performing the search operation in AI programs.

- ✓ **Production System Consists of**
  - **A set of rules** of the form x → y

    LHS = Patterns and **RHS** = Action

    LHS determines applicability of rule, RHS specifies action.
  - **Database** OR **knowledge base** of the domain.
  - **Control Strategy:** That specifies the order in which rules are applied on the knowledge base and derive the new knowledge.
- ✓ In general, the process of solving a problem can usefully be modeled as a production system. i.e, any computable procedure can be modeled as production system.

- ✓ **Advantages**
  - It models strong data-driven nature of intelligent action
  - New rules can easily be added to take care of new situations without disturbing the rest of the system
  - Handling of changes in knowledge base is dynamic

- Helps in making inference mechanism easy.

➔ PRODUCTION SYSTEM CHARACTERISTICS

✓ **4 categories of Production Systems**

	Monotonic	**Non-Monotonic**
Partially commutative	**Ex:** theorem proving	**Ex:** Blocks world, 8-puzzle
Non-partially commutative	**Ex:** chemical Synthesis	**Ex:** Bridge/cards problem

✓ **A Monotonic Production System**: is one in which the application of **a rule** never prevents the later application of **another rule** that could also have been applied at the time the **first rule was selected**.

✓ **A Non-Monotonic Production System:** is a production system is one in which the above statement is not true.

✓ **A partially commutative production system:** is a production system with the property that if the application of a particulars sequence of rules transforms state 'X' into state 'Y' then any permutation of those rules that is allowable also transforms state 'X' into state 'Y'. **Partially commutative monotonic production systems** are useful for solving **ignorable problems** and these can be easily implemented without the ability to backtrack to previous states when it is discovered that an incorrect path has been followed.

✓ **Non- Partially commutative production system:** are useful for many problems in which irreversible changes occur.

✓ **Non-Monotonic Partially commutative systems:** are useful for problems in which changes occur but can be reversed in which order of operations is not critical.

✓ **Commutative production system** is one which is both **monotonic** and **partially commutative.**

➔ Control Strategies

✓ Specifies what rule to be applied next and in what order / sequence during the search process. There are many control Strategies exist **uniformed search strategies** like DFS, BFS and **informed / heuristic Strategies** like Hill climbing, Backtracking, Branch and Bound, Best first search etc.,.

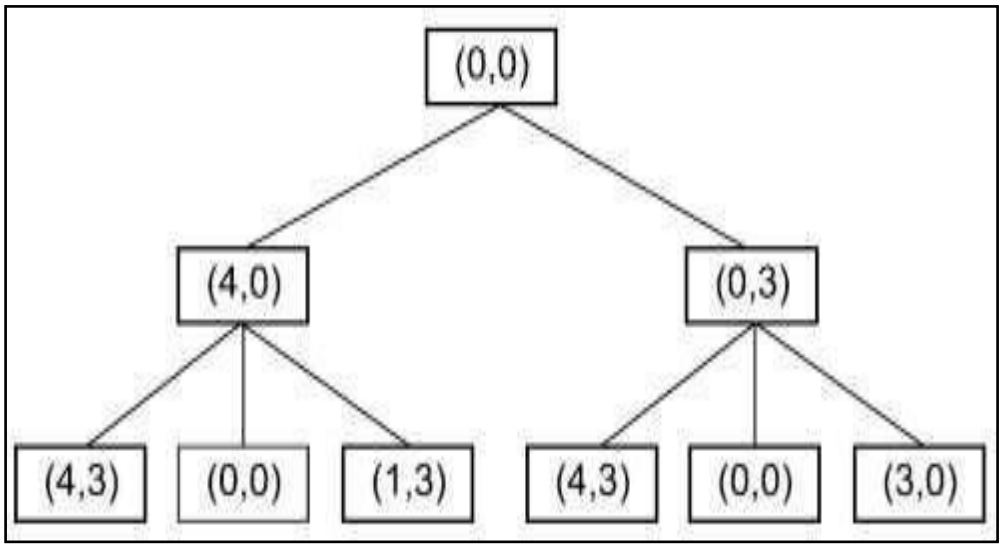
✓ **Two requirements for good control strategies are:**

- They should **cause motion**, so that it **lead to a solution**
- They should be **systematic**, so that it **lead to a solution**

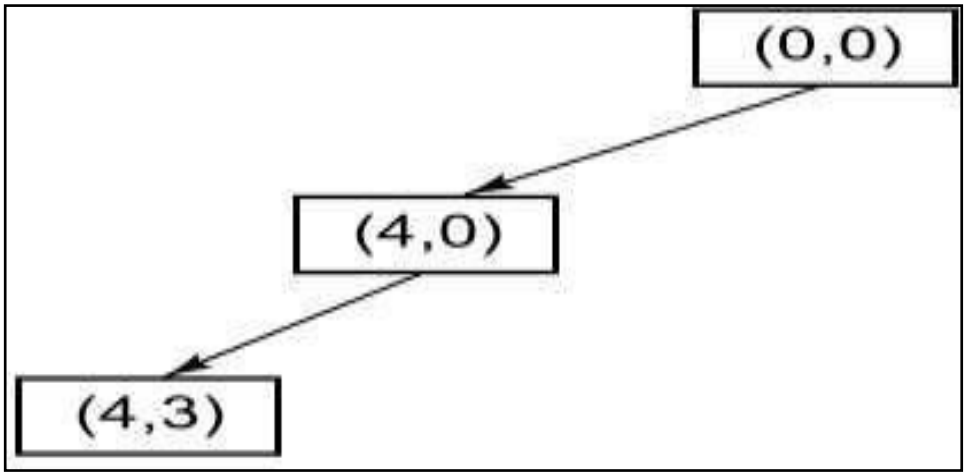
Ex: BFS and **DFS** strategies

- **Algorithm: Breadth-First Search**

1. Create a variable called NODE-LIST and set it to the initial state.
2. Until a goal state is found or NODE-LIST is empty:
   a) Remove the first element from NODE-LIST and call it E. If NODE-LIST was empty, quit.
   b) For each way that each rule can match the state described in E do.
      i) Apply the rule to generate a new state,
      ii) If the new state is a goal state, quit and return this state.
      iii) Otherwise, add the new state to the end of NODE-LIST.



- **Algorithm: Depth-First Search**



1. If the initial state is a goal state, quit and return success.
2. Otherwise, do the following until success or failure is signaled:
   a) Generate a successor, E, of the initial state. If there are no more successors, signal failure.
   b) Call depth-First Search with E as the initial state.
   c) If success is returned, signal success. Otherwise continue in this loop.

Advantages of Depth-First Search

- ✓ **Depth-first Search** requires **less memory** since only the nodes on the current path are stored. This contract with breadth-first search, where all of the tree that has so far generated must be stored- **more Memory**.
- ✓ **Depth-first search** may find a solution without examining much of the **search space** at all. This contracts with breadth-first search in which all parts of the tree must be examined to **level n before any nodes on level n+1** can be examined.

Advantages of Breadth-First Search

- ✓ Breadth-first search will not get trapped exploring a **blind alley (path)**. This contrast with depth-first searching, which may follow a single **unfruitful path** for a very long time, before the path actually terminates in a state that has **no successors.** This is a particular problem in depth-first search if there are loops (i.e., a state has a successor that is also one of its ancestors) unless **special care is expended** to test such a situation.
- ✓ If there is a solution, then **breadth-first search is guaranteed to find** it. Furthermore, if there are multiple solutions, then a **minimal solution** i.e., one that requires the **minimum number of steps** will be found. This is guaranteed by the fact that longer paths are never explored until

  all shorter ones have already been examined. These contrasts with depth-first search, which may find a **long path** to a solution in one part of the tree, when a **shorter path exists** in some other, unexplored part of the tree.

- ▶ **The Travelling Salesman Problem**

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

  A simple, **motion-causing and systematic control structure** could, in principle, solve this problem. It would simply explore all possible paths in the tree and return the one with the shortest length. This approach will work in practice for very short lists of cities. But it breaks down quickly as the number of cities grows. If there are N cities, then the **number of different paths** among them is **1.2… (N-1),** or **(N-1)!** . The time to examine a single path is proportional to N. So the total time required to perform this search is proportional to **N!** . Assuming there

are only 10 cities, 10! is 3,628,800, which is a very large number. The salesman could easily have 25 cities to visit. To solve this problem would take more time than he would be willing to spend. This phenomenon is called **combinatorial explosion**. To combat it, we need a new control strategy.

➔ Heuristic Search

- ✓ A **Heuristic function** is a function that maps from **problem state description** to **measures of desirability, usually represented as numbers.**
- ✓ Improves the **efficiency** of a search process and always finds a very goal solution for hard problems.
- ✓ In general, it points in **interesting directions.**
- ✓ Like tour guides, helps to **guide a search process. Ex: nearest Neighbor Heuristic**
- ✓ Well designed Heuristic functions can play an important part in efficiently guiding a search process toward a solution.
- ✓ Sometimes very simple Heuristic functions can provide a fairly good estimate of whether a path is good or not. In other situations, more complex Heuristic functions should be employed.

Note:

1. Production systems represent both knowledge as well as action.
2. Production systems provide a language in which the representation of expert knowledge is very natural.
3. Production systems provide a Heuristic model for human behavior.
4. A good solution to a problem requires an efficient control strategy.

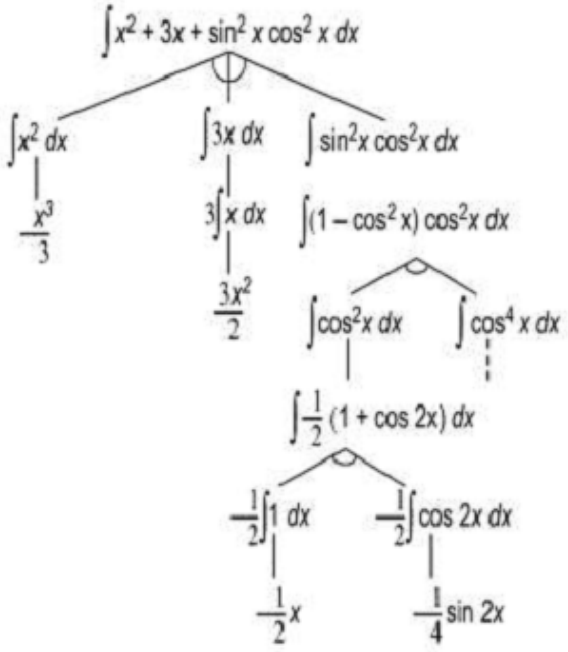
➢ AI PROBLEM CHARACTERISTICS

In order to choose the most appropriate method for a particular problem, it is necessary to analyze the problem along several key **dimensions / Characteristics:**

1. Is the problem **Decomposable** into a set of independent smaller and/or easier sub problem? **(D)**
2. Can solution steps be **Ignored** or at least undone if they prove unwise? **(I)**
3. Is the problem's universe **Predictable**? **[predictability (P)]**
4. Is a good solution to the problem obvious without **comparison** to all other possible solutions? **[Comparability (C)]**
5. Is the desired solution a **state** of the world **OR** a **path** to a state? **(R)**
6. Is a large amount of knowledge absolutely required to solve the problem, OR is knowledge important only to constrain the search? **[knowledge based consistency (C)]**
7. Can a computer that is simply given the problem return the solution, or will the solution of the problem require **interaction** between the computer and a person? **[Interactions (I)]**

Note: Above characteristics may be abbreviated as D I P C R C I for easy remembrance.

1. **Is the problem decomposable?**
   Suppose the following expression is given, then it could be solved as follows. It can be broken down or decomposed into three smaller problems, each of which can be solved using small collection of specific rules.



1. **Can solution steps be Ignored or Undone?**

These three problems: **theorem proving, 8-puzzle** and **chess** illustrate the differences between three important classes of problems:

Example:

- **Ignorable (e.g.., theorem proving)**, in which solution steps can be **ignored**
- **Recoverable (e.g., 8-puzzle),** in which solution steps can be **undone**
- **Irrecoverable (e.g.., chess),** in which solution steps **cannot** be **undone**

These three definitions make reference to the steps of the solution to a problem and thus may appear to characterize particular production system for solving a problem. This is true for each of the problem used as examples above. When this is the case, it makes sense to view the recoverability of a problem as equivalent to recoverability of a natural formulation of it.

2. **Is the problem's Universe Predictable?**

The planning process can only be done effectively for certain-outcome. A few examples of such problems are:

- ✓ **Playing bridge:** we can do fairly well since we have available accurate estimates of the probabilities of each of the possible outcomes.
- ✓ **Controlling a robot arm:** The outcome is uncertain for a variety of reasons. Someone might move something into the path of the arm. The gears of the arm might stick. A slight error could cause the arm to knock over a whole stack of things.
- ✓ **Helping a lawyer decide how to defend his client against a murder charge**: Here we probably cannot even list all the possible outcomes, much less assess their probabilities.

Examples:

In 8-puzzle problem → certain outcome

In bridge problem → uncertain outcome (cards)

In controlling a Robot arm → uncertain outcome

In helping a lawyer decides how to depend his client against a murder charge. (cannot list all possible outcomes)

3. **Is a good solution absolute OR Relative?**

Example: In the problem of drawing conclusions from a given set of premises, we may need to consider the facts in some sequence and try to relate them to find new facts which can intern be used as premises in conclusion drawing process. So, we will have a relative path from facts / premises to the conclusion.

Example: Marcus was a man.
Marcus was a Pompeian.
Marcus was born in 40 AD.
All men are mortal.

All Pompeian's died when the volcano erupted in 79 AD.
No mortal lives longer than 150 years it is now 1991 AD.
Suppose, we ask the question, "**Is Marcus alive?**" No.

Here, Solution will be a **relative path.**

1. **Is the solution a state or a path?**
   Example: In **man, Tiger, Cow, & Grass** problem, solution is a **state.**
   In **Travelling salesman** problem solution is a **path.**

2. **What is the Role of the Knowledge? Is the Knowledge base consistent?**
   Example: In playing chess, **role of knowledge is little**-just the rules for legal moves & simple control mechanisms whereas, in the problem of **scanning daily news papers** and deciding which are supporting the democrats and which are supporting republicans in some election, the role of **knowledge is extensive**. Knowledge base used for solving a problem should be **consistent.**

3. **Does the task require interactions with a person?**
   Can we compute simply the given problem and return the solution? OR will the solution of a problem requires user interaction?

4. **Problem Classification**
   There are several broad classes into which the problems may fall.
   Example: In Diagnostic tasks, i.e. both in **medical** and **fault diagnosis,** we need to examine the input and decide which of a set of known classes, the **input is an instance of**?

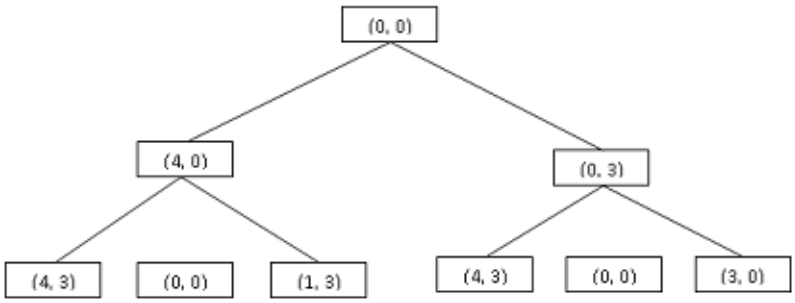
➢ ISSUES IN THE DESIGN OF SEARCH PROGRAM

Every search process can be viewed as a traversal of a tree structure in which each node represent a problem state and each **arc** represents a relationship between the states represented by the nodes it connects. For example, **Figure below** shows part of a search tree for a water jug problem. The search process must find a **path or paths** through the tree that connect an **initial state** with **one or more final states.** The tree that must be searched could in principle, be constructed in it's entirely from the rules that define allowable moves in the problem space.

Some important issues that arise in all of them:

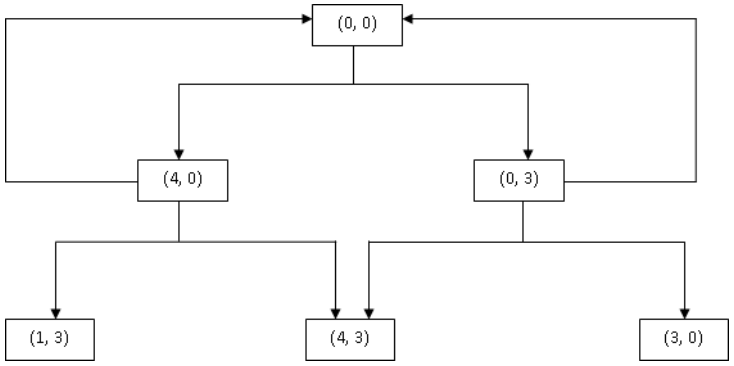
- ✓ The **direction** in which to conduct the search (forward versus backward reasoning).
- ✓ How to select **applicable rules** (matching). Production systems typically spend most of their time looking for rules to apply, so it is critical to have efficient procedures for matching rules against states.
- ✓ How to **represent each node** of the search process. For problems like chess, a node can be fully represented by a simple array. In more complex problem solving, however, it is inefficient and/or impossible to represent all of the facts in the world and to determine all of the side effects an action may have.

A Search Tree for the Water Jug Problem



For example, in the tree shown above, the node (4, 3), can be generated either by first filling the 4-gallon jug and then the 3-gallon one. This example also illustrates another problem that often arises when the search process operates as a tree walk. On the third level, the node (0, 0) appears. But this is the same as the top node of the tree, which has already been expanded. Those two paths have not gotten us anywhere, so we would like to eliminate them and continue only along the other branches. The waste of effort that arises when the same node is generated more than once can be avoided at the price of additional bookkeeping. Instead of traversing a search tree, we traverse a directed graph. This graph differs from a tree in that several paths may come together at a node. The graph corresponding to the tree is shown in figure below. Any **tree search procedure** that keeps track of all the nodes that have been generated so far can be converted to a **graph search procedure** by modifying the action performed each time a node is generated. **Graph search procedures** are useful for dealing with **partially commutative production systems.**

A Search Graph for the Water Jug Problem



Ballari Institute of Technology & Management, Bellary

Department of Electronics and Communication

Artificial Intelligence –Module I
Question Bank

1. Define i) **Intelligence** ii) **Artificial Intelligence iii) Agent** iv) **Logical Reasoning**
2. Briefly explain the Task domains of AI.
3. What are the underlying assumptions about Intelligence? At what level we are trying to model human Intelligence?
4. What is an Artificial Intelligence technique? Explain with an example.
5. What are AI problem characteristics? Explain each with an Example.
6. List the **capabilities** of a Computer in order to pass the **Turing test**.
7. Write a brief **history of AI** along with the **State of the Art**.
8. What is a **Production System**? Discuss the different categories of production systems.
9. Explain the Tic Tac Toe example with suitable example
10. Taking the **8-puzzle** problem as an example, clearly explain the concept of state space search – indicating the meanings of the following terms:
    a) State space b) Initial state c) Goal state d) Legal operation
    Draw the State space tree and show one possible solution for the following initial and goal state

To be transformed

1	2	3
8	5	6
4	7	

?

1	2	3
4	5	6
7	8	

Initial state / configuration (start) **Goal state/configuration (Final)**

11. There are 2 water jugs of 6-litre and 8-litre. Neither has any measuring marker. There is a tap that can be used to fill the jugs with the water. Indicate how 8-litre jug can be filled half. Solve this **Water-Jug** problem by giving complete set of production rules and state space tree.
12. What are the **issues** in designing search program ?
13. Explain how search can be used to solve problem in AI
14. Write the procedure for **hill-climbing search** Technique. Explain with suitable example, Briefly discuss about the problems in hill climbing and the ways of dealing with these problems
15. Explain the Depth-first-search and Breadth-First search procedures with an example for each. List their advantages and disadvantages.