Annai Women's College

(Arts and Science)
Affiliated to Bharathidasan University – Tiruchirapalli.
TNPL Road ,Punnam Chattram, Karur-639 136

Department of Computer Science, Computer Applications & IT

Faculty Name : **Mrs.V.Bhuvaneswari,Msc., M.phil.,**
Major : **I M.Sc(Computer Science)**
Paper Code : **P16CSE2B**
Title of the Paper : **Artificial Intelligence**

INDEX

Annai Women's College,Karur.

(Arts and Science)

Department of Computer Science.

Class: I M.Sc (Computer Science) **Staff Incharge: V.Bhuvaneswari M.Sc.,M.Phil., AP in CS Department.**

Artificial Intelligence

UNIT I: Introduction: AI Problems - Al techniques - Criteria for success. Problems, Problem Spaces, Search: State space search - Production Systems

UNIT II: Heuristic Search techniques: Generate and Test - Hill Climbing- Best-First - Means-end analysis. Knowledge representation issues: Representations and mappings - Approaches to Knowledge representations -Issues in Knowledge representations - Frame Problem.

UNIT III: Using Predicate logic: Representing simple facts in logic - Representing Instance and is a relationships - Computable functions and predicates - Resolution.

UNIT III: Using Predicate logic: Representing simple facts in logic - Representing Instance and is a relationships - Computable functions and predicates - Resolution.

UNIT V: Game playing – The mini max search procedure – Expert System - Perception and Action

TEXT BOOK: Elaine Rich and Kevin Knight," **Artificial Intelligence**", Tata McGraw Hill Publishers company Pvt Ltd, Second Edition, 1991.

Unit1: Chapter 1(1.1, 1.3.1.5), Chapter 2(2.1, 2.2)

Unit2: Chapter 3(3.1, 3.2, 3.P, 3.6), Chapter 4(4.1, 4.2, 4.3, 4.4).

Unit3: Chapter 5(5.1, 5.2, 5.3, 5.4).

Unit4: Chapter 6.

Unit5: Chapter 12(12.1, 12.2), Chapter 20 and Chapter 21.

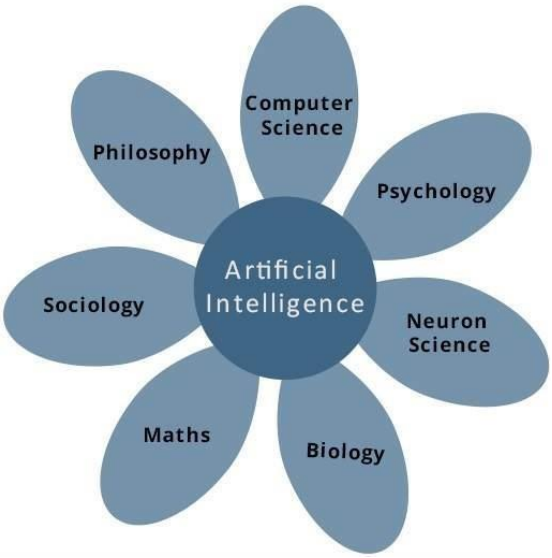
Unit 1

Introduction:

A branch of Computer Science named *Artificial Intelligence* pursues creating the computers or machines as intelligent as human beings.

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.

History of AI

Here is the history of AI during 20th century −

Year	Milestone / Innovation
1923	Karel Čapek play named "Rossum's Universal Robots" (RUR) opens in London, first use of the word "robot" in English.
1943	Foundations for neural networks laid.
1945	Isaac Asimov, a Columbia University alumni, coined the term *Robotics*.
1950	Alan Turing introduced Turing Test for evaluation of intelligence and published *Computing Machinery and Intelligence.* Claude Shannon published *Detailed Analysis of Chess Playing* as a search.
1956	John McCarthy coined the term *Artificial Intelligence*. Demonstration of the first running AI program at Carnegie Mellon University.
1958	John McCarthy invents LISP programming language for AI.

1964	Danny Bobrow's dissertation at MIT showed that computers can understand natural language well enough to solve algebra word problems correctly.
1965	Joseph Weizenbaum at MIT built *ELIZA*, an interactive problem that carries on a dialogue in English.
1969	Scientists at Stanford Research Institute Developed *Shakey*, a robot, equipped with locomotion, perception, and problem solving.
1973	The Assembly Robotics group at Edinburgh University built *Freddy*, the Famous Scottish Robot, capable of using vision to locate and assemble models.
1979	The first computer-controlled autonomous vehicle, Stanford Cart, was built.
1985	Harold Cohen created and demonstrated the drawing program, *Aaron*.
1990	Major advances in all areas of AI − • Significant demonstrations in machine learning • Case-based reasoning • Multi-agent planning • Scheduling • Data mining, Web Crawler • natural language understanding and translation • Vision, Virtual Reality • Games
1997	The Deep Blue Chess Program beats the then world chess champion, Garry Kasparov.
2000	Interactive robot pets become commercially available. MIT displays *Kismet*, a robot with a face that expresses emotions. The robot *Nomad* explores remote regions of Antarctica and locates meteorites.

Philosophy of AI

While exploiting the power of the computer systems, the curiosity of human, lead him to wonder, *"Can a machine think and behave like humans do?"*

Thus, the development of AI started with the intention of creating similar intelligence in machines that we find and regard high in humans.

Goals of AI

- **To Create Expert Systems** − The systems which exhibit intelligent behavior, learn, demonstrate, explain, and advice its users.
- **To Implement Human Intelligence in Machines** − Creating systems that understand, think, learn, and behave like humans.

What Contributes to AI?

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.

Out of the following areas, one or multiple areas can contribute to build an intelligent system.

Programming Without and With AI

The programming without and with AI is different in following ways −

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.

What is AI Technique?

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.

AI Technique is a manner to organize and use the knowledge efficiently.

It should be perceivable by the people who provide it.

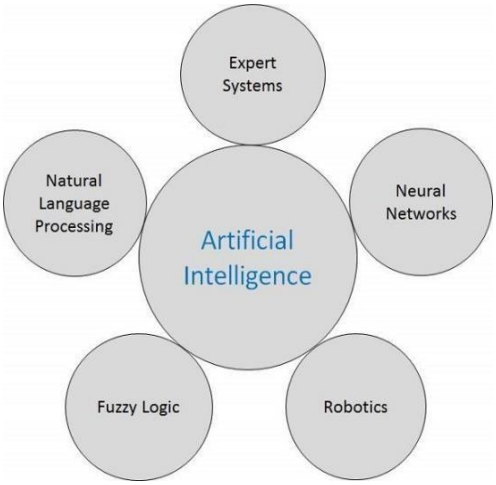
- It should be easily modifiable to correct errors.
- It should be useful in many situations though it is incomplete or inaccurate.

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

Applications of AI

AI has been dominant in various fields such as,

- **Gaming** − AI plays crucial role in strategic games such as chess, poker, tic-tac-toe, etc., where machine can think of large number of possible positions based on heuristic knowledge.
- **Natural Language Processing** − It is possible to interact with the computer that understands natural language spoken by humans.
- **Expert Systems** − There are some applications which integrate machine, software, and special information to impart reasoning and advising. They provide explanation and advice to the users.
- **Vision Systems** − These systems understand, interpret, and comprehend visual input on the computer. For example,
  - A spying aeroplane takes photographs, which are used to figure out spatial information or map of the areas.
  - Doctors use clinical expert system to diagnose the patient.
  - Police use computer software that can recognize the face of criminal with the stored portrait made by forensic artist.
- **Speech Recognition** − Some intelligent systems are capable of hearing and comprehending the language in terms of sentences and their meanings while a human talks to it. It can handle different accents, slang words, noise in the background, change in human's noise due to cold, etc.
- **Handwriting Recognition** − The handwriting recognition software reads the text written on paper by a pen or on screen by a stylus. It can recognize the shapes of the letters and convert it into editable text.
- **Intelligent Robots** − Robots are able to perform the tasks given by a human. They have sensors to detect physical data from the real world such as light, heat, temperature, movement, sound, bump, and pressure. They have efficient processors, multiple sensors and huge memory, to exhibit intelligence. In addition, they are capable of learning from their mistakes and they can adapt to the new environment.



Research Areas: The domain of artificial intelligence is huge in breadth and width. While proceeding, we consider the broadly common and prospering research areas in the domain of AI

Speech and Voice Recognition

These both terms are common in robotics, expert systems and natural language processing. Though these terms are used interchangeably, their objectives are different.

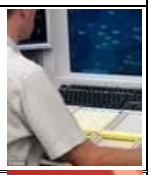
Speech Recognition	Voice Recognition
The speech recognition aims at understanding and comprehending **WHAT** was spoken.	The objective of voice recognition is to recognize **WHO** is speaking.
It is used in hand-free computing, map, or menu navigation.	It is used to identify a person by analysing its tone, voice pitch, and accent, etc.
Machine does not need training for Speech Recognition as it is not speaker dependent.	This recognition system needs training as it is person oriented.
Speaker independent Speech Recognition systems are difficult to develop.	Speaker dependent Speech Recognition systems are comparatively easy to develop.

Working of Speech and Voice Recognition Systems

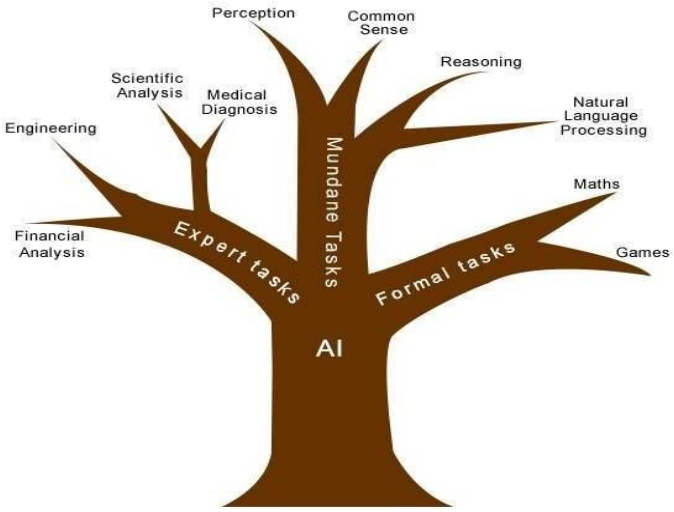
The user input spoken at a microphone goes to sound card of the system. The converter turns the analog signal into equivalent digital signal for the speech processing. The database is used to compare the sound patterns to recognize the words. Finally, a reverse feedback is given to the database. This source-language text becomes input to the Translation Engine, which converts it to the target language text. They are supported with interactive GUI, large database of vocabulary, etc.

Real Life Applications of AI Research Areas

There is a large array of applications where AI is serving common people in their day-to-day lives:

.No.	Research Areas	Example
	Expert Systems Examples − Flight-tracking systems, Clinical systems.	
	Natural Language Processing Examples: Google Now feature, speech recognition, Automatic voice output.	
	Neural Networks Examples − Pattern recognition systems such as face recognition, character recognition, handwriting recognition.	
	Robotics Examples − Industrial robots for moving, spraying, painting, precision checking, drilling, cleaning, coating, carving, etc.	
	Fuzzy Logic Systems Examples − Consumer electronics, automobiles, etc.	

Task Classification of AI



The domain of AI is classified into **Formal tasks, Mundane tasks,** and **Expert tasks.** Humans learn **mundane (ordinary) tasks** since their birth. They learn by perception, speaking, using language, and locomotives. They learn Formal Tasks and Expert Tasks later, in that order.For humans, the mundane tasks are easiest to learn. The same was considered true before trying to implement mundane tasks in machines. Earlier, all work of AI was concentrated in the mundane task domain.

Later, it turned out that the machine requires more knowledge, complex knowledge representation, and complicated algorithms for handling mundane tasks. This is the reason **why**

AI work is more prospering in the Expert Tasks domain now, as the expert task domain needs expert knowledge without common sense, which can be easier to represent and handle.

AI Techniques:

Artificial Intelligence problems span a very broad spectrum.They apperar to have very little in common expected that they are hard. One of the few hard and fast results to come out of the first three decades of AI research is that intelligence requires knowledge.

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

Tic-Tac-Toe:

Consider a board with the nine positions numbered as follows:

1	2	3
4	5	6
7	8	9

When X plays 1 as their opening move, then O should take 5. Then X takes 9 (in this situation, O should not take 3 or 7, O should take 2, 4, 6 or 8):

- X1 → O5 → X9 → O2 → X8 → O7 → X3 → O6 → X4, this game will be a draw.

or 6 (in this situation, O should not take 4 or 7, O should take 2, 3, 8 or 9. In fact, taking 9 is the best move, since a non-perfect player X may take 4, then O can take 7 to win).

- X1 → O5 → X6 → O2 → X8, then O should not take 3, or X can take 7 to win, and O should not take 4, or X can take 9 to win, O should take 7 or 9.
- X1 → O5 → X6 → O2 → X8 → O7 → X3 → O9 → X4, this game will be a draw.
- X1 → O5 → X6 → O2 → X8 → O9 → X4 (7) → O7 (4) → X3, this game will be a draw.
- X1 → O5 → X6 → O3 → X7 → O4 → X8 (9) → O9 (8) → X2, this game will be a draw.
- X1 → O5 → X6 → O9, then X should not take 4, or O can take 7 to win, X should take 2, 3, 7 or 8.

- X1 → O5 → X6 → O9 → X2 → O3 → X7 → O4 → X8, this game will be a draw.
- X1 → O5 → X6 → O9 → X3 → O2 → X8 → O4 (7) → X7 (4), this game will be a draw.
- X1 → O5 → X6 → O9 → X7 → O4 → X2 (3) → O3 (2) → X8, this game will be a draw.
- X1 → O5 → X6 → O9 → X8 → O2 (3, 4, 7) → X4/7 (4/7, 2/3, 2/3) → O7/4 (7/4, 3/2, 3/2) → X3 (2, 7, 4), this game will be a draw.

In both of these situations (X takes 9 or 6 as second move), X has a property to win.

If X is not a perfect player, X may take 2 or 3 as second move. Then this game will be a draw, X cannot win.

- X1 → O5 → X2 → O3 → X7 → O4 → X6 → O8 (9) → X9 (8), this game will be a draw.
- X1 → O5 → X3 → O2 → X8 → O4 (6) → X6 (4) → O9 (7) → X7 (9), this game will be a draw.

If X plays 1 opening move, and O is not a perfect player, the following may happen:

Although O takes the only good position (5) as first move, but O takes a bad position as second move:

- X1 → O5 → X9 → O3 → X7, then X can take 4 or 8 to win.
- X1 → O5 → X6 → O4 → X3, then X can take 2 or 9 to win.
- X1 → O5 → X6 → O7 → X3, then X can take 2 or 9 to win.

Although O takes good positions as the first two moves, but O takes a bad position as third move:

- X1 → O5 → X6 → O2 → X8 → O3 → X7, then X can take 4 or 9 to win.
- X1 → O5 → X6 → O2 → X8 → O4 → X9, then X can take 3 or 7 to win.

O takes a bad position as first move (except of 5, all other positions are bad):

- X1 → O3 → X7 → O4 → X9, then X can take 5 or 8 to win.
- X1 → O9 → X3 → O2 → X7, then X can take 4 or 5 to win.
- X1 → O2 → X5 → O9 → X7, then X can take 3 or 4 to win.
- X1 → O6 → X5 → O9 → X3, then X can take 2 or 7 to win.

Many board games share the element of trying to be the first to get *n*-in-a-row, including Three Men's Morris, Nine Men's Morris, pente, gomoku, Qubic, Connect

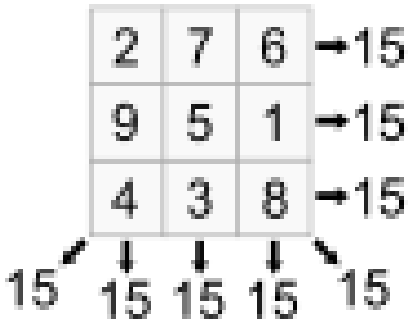
Four, Quarto, Gobblet, Order and Chaos, Toss Across, and Mojo. Tic-tac-toe is an instance of an m,n,k-game, where two players alternate taking turns on an *m*×*n* board until one of them gets *k* in a row. Harary's generalized tic-tac-toe is an even broader generalization.

Other variations of tic-tac-toe include:

- 3-dimensional tic-tac-toe on a 3×3×3 board. In this game, the first player has an easy win by playing in the centre if 2 people are playing.

One can play on a board of 4x4 squares, winning in several ways. Winning can include: 4 in a straight line, 4 in a diagonal line, 4 in a diamond, or 4 to make a square.

Another variant, Qubic, is played on a 4×4×4 board; it was solved by Oren Patashnik in 1980 (the first player can force a win). Higher dimensional variations are also possible.



- In misère tic-tac-toe the player wins if the opponent gets *n* in a row. A 3×3 game is a draw. More generally, the first player can draw or win on any board (of any dimension) whose side length is odd, by playing first in the central cell and then mirroring the opponent's moves.

- In "wild" tic-tac-toe, players can choose to place either X or O on each move.
- There is a game that is isomorphic to tic-tac-toe, but on the surface appears completely different. It is called Pick15 or Number Scrabble Two players in turn say a number between one and nine. A particular number may not be repeated. The game is won by the player who has said three numbers whose sum is 15. If all the numbers are used and no one gets three numbers that add up to 15 then the game is a draw.Plotting these numbers on a 3×3 magic square shows that the game exactly corresponds with tic-tac-toe, since three numbers will be arranged in a straight line if and only if they total 15.

Question Answering:

In this series of programs that read in English text and then answer questions, also Stated in English about that text.

A QA implementation, usually a computer program, may construct its answers by querying a structured database of knowledge or information, usually a knowledge base. More commonly, QA systems can pull answers from an unstructured collection of natural language documents.Some examples of natural language document collections used for QA systems include:

A local collection of reference texts,

- Internal organization documents and web pages
- Compiled newswire reports
- A set of Wikipedia pages
- A subset of World Wide Web pages

For Example: 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.

Consider the above paragraph, we make the Questions,

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

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 to find appropriate answers in the text.

The Algorithm:

To answer a question, do the following:

1. Compare each element of the question patterns against the Question and use all those that match successfully to generate a set of the text patterns.
2. Pass each of these patterns through a substitution process that generates alternative forms of verbs.
3. Apply each of these text patterns to text, and collect all the resulting answers.
4. Reply with the st of answers just collected

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.
2. Analyze 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.

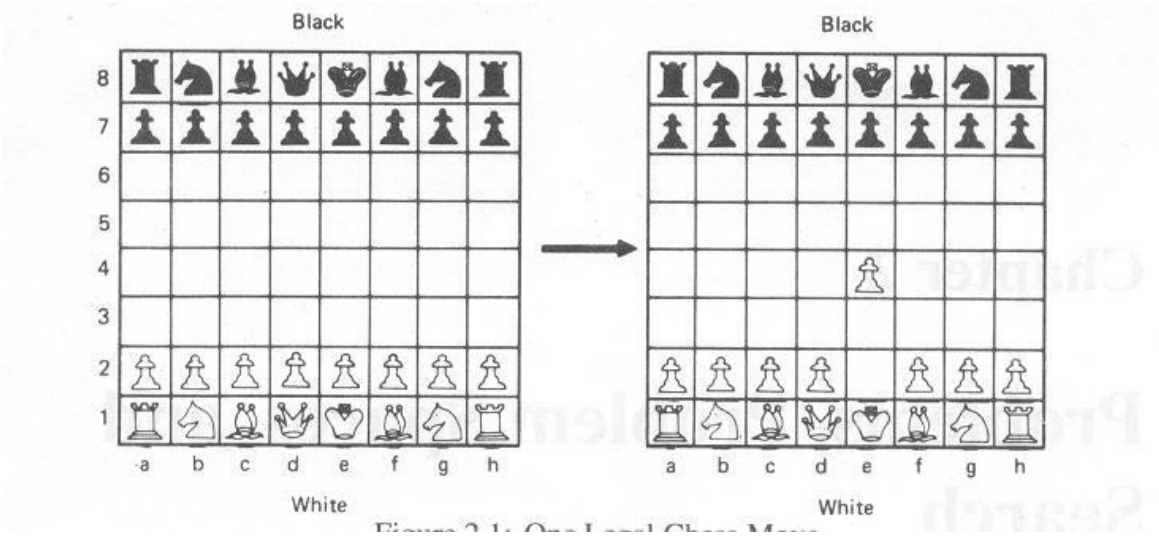
Defining a Problem as a State Space Search:

• State space search

• Search strategies

• Problem characteristics

• Design of search programsProblem solving = Searching for a goal state



State Space Search: Playing Chess

• Each position can be described by an 8-by-8 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 or her king is under attack.

• Legal moves can be described by a set of rules:

- Left sides are matched against the current state.
- Right sides describe the new resulting state.

• State space is a set of legal positions.

• Starting at the initial state.

• Using the set of rules to move from one state to another.

• Attempting to end up in a goal state.

State Space Search: 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 exactly 2 litres of water into 4-litre jug?"

State Space Search: Water Jug Problem

• State: (x, y)

x = 0, 1, 2, 3, or 4

y = 0, 1, 2, 3

• Start state: (0, 0).

• Goal state: (2, n) for any n.

• Attempting to end up in a goal state.

1. $(x, y) \rightarrow (4, y)$ if $x < 4$
2. $(x, y) \rightarrow (x, 3)$ if $y < 3$
3. $(x, y) \rightarrow (x - d, y)$ if $x > 0$
4. $(x, y) \rightarrow (x, y - d)$ if $y > 0$
5. $(x, y) \rightarrow (0, y)$ if $x > 0$
6. $(x, y) \rightarrow (x, 0)$ if $y > 0$
7. $(x, y) \rightarrow (4, y - (4 - x))$ if $x + y \geq 4, y > 0$
8. $(x, y) \rightarrow (x - (3 - y), 3)$ if $x + y \geq 3, x > 0$
9. $(x, y) \rightarrow (x + y, 0)$ if $x + y \leq 4, y > 0$
10. $(x, y) \rightarrow (0, x + y)$ if $x + y \leq 3, x > 0$
11. $(0, 2) \rightarrow (2, 0)$
12. $(2, y) \rightarrow (0, y)$

1. current state = (0, 0)

2. Loop until reaching the goal state (2, 0)

- Apply a rule whose left side matches the current state − Set the new current state to be the resulting state

(0, 0) (0, 3) (3, 0) (3, 3) (4, 2) (0, 2) (2, 0)

The role of the condition in the left side of a rule ⇒ restrict the application of the rule ⇒ more efficient

1. $(x, y) \rightarrow (4, y)$ if $x < 4$

2. $(x, y) \rightarrow (x, 3)$ if $y < 3$

Special-purpose rules to capture special-case knowledge that can be used at some stage in solving a problem

11. $(0, 2) \rightarrow (2, 0)$

12. $(2, y) \rightarrow (0, y)$

State Space Search: Summary

1. Define a state space that contains all the possible configurations of the relevant objects.

2. Specify the initial states.

3. Specify the goal states.

4. Specify a set of rules:

- What are unstated assumptions?
- How general should the rules be?
- How much knowledge for solutions should be in the rules?

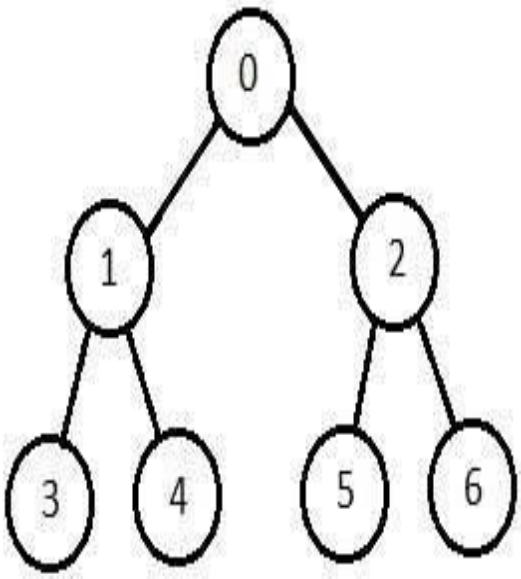
Control Strategies

Requirements of a good search strategy:

1. It causes motion Otherwise; it will never lead to a solution.

2. It is systematic Otherwise; it may use more steps than necessary.

3. It is efficient Find a good, but not necessarily the best, answer.



Breadth-First Search

It starts from the root node, explores the neighboring nodes first and moves towards the next level neighbors. It generates one tree at a time until the solution is found. It can be implemented using FIFO queue data structure. This method provides shortest path to the solution.

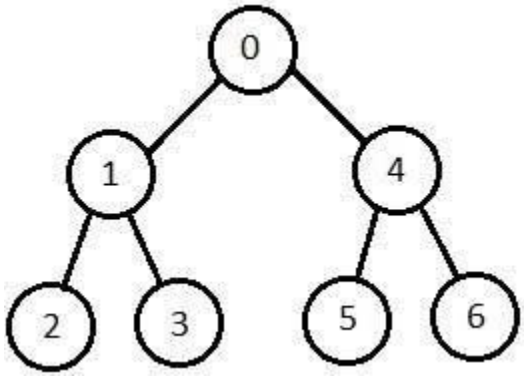
If branching factor (average number of child nodes for a given node) = b and depth = d, then number of nodes at level d = b^d.

The total no of nodes created in worst case is $b + b^2 + b^3 + \ldots + b^d$.

Disadvantage

Since each level of nodes is saved for creating next one, it consumes a lot of memory space. Space requirement to store nodes is exponential. Its complexity depends on the number of nodes. It can check duplicate nodes.

Depth-First Search



It is implemented in recursion with LIFO stack data structure. It creates the same set of nodes as Breadth-First method, only in the different order.

As the nodes on the single path are stored in each iteration from root to leaf node, the space requirement to store nodes is linear. With branching factor *b*and depth as *m*, the storage space is *bm*.

Disadvantage:

This algorithm may not terminate and go on infinitely on one path. The solution to this issue is to choose a cut-off depth. If the ideal cut-off is*d*, and if chosen cut-off is lesser than *d*, then this algorithm may fail. If chosen cut-off is more than *d*, then execution time increases.

Its complexity depends on the number of paths. It cannot check duplicate nodes.

Bidirectional Search

It searches forward from initial state and backward from goal state till both meet to identify a common state.

The path from initial state is concatenated with the inverse path from the goal state. Each search is done only up to half of the total path.

Uniform Cost Search

Sorting is done in increasing cost of the path to a node. It always expands the least cost node. It is identical to Breadth First search if each transition has the same cost. It explores paths in the increasing order of cost.

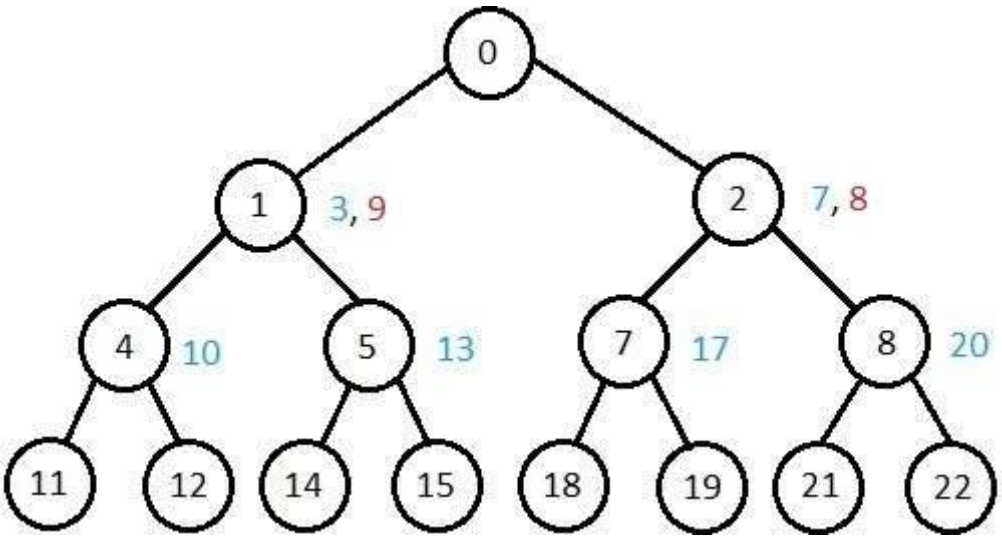
Disadvantage:

There can be multiple long paths with the cost ≤ C*. Uniform Cost search must explore them all.

Iterative Deepening Depth-First Search

It performs depth-first search to level 1, starts over, executes a complete depth-first search to level 2, and continues in such way till the solution is found.

It never creates a node until all lower nodes are generated. It only saves a stack of nodes. The algorithm ends when it finds a solution at depth *d*. The number of nodes created at depth *d* is b^d and at depth *d-1* is $b^{d-1.}$



Comparison of Various Algorithms Complexities

Let us see the performance of algorithms based on various criteria −

Criterion	Breadth First	Depth First	Bidirectional	Uniform Cost	Interactive Deepening
Time	b^d	b^m	$b^{d/2}$	b^d	b^d
Space	b^d	b^m	$b^{d/2}$	b^d	b^d
Optimality	Yes	No	Yes	Yes	Yes
Completeness	Yes	No	Yes	Yes	Yes

Search Strategies: Heuristic Search

• Heuristic: involving or serving as an aid to learning, discovery, or problem-solving by experimental and especially trial-and-error methods. (Merriam-Webster's dictionary)

• Heuristic technique improves the efficiency of a search process, possibly by sacrificing claims of completeness or optimality.

• Heuristic is for combinatorial explosion.

• Optimal solutions are rarely needed.

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 1 10 D 5 B5

Nearest neighbour heuristic:

1. Select a starting city.

2. Select the one closest to the current city.

3. Repeat step 2 until all cities have been visited.

Nearest neighbour heuristic:

1. Select a starting city.

2. Select the one closest to the current city.

3. Repeat step 2 until all cities have been visited.

O(n2) vs. O(n!)

• Heuristic function: state descriptions → measures of desirability

Problem Characteristics:

To choose an appropriate method for a particular problem:

• Is the problem decomposable?

• Can solution steps be ignored or undone?

• Is the universe predictable?

• Is a good solution absolute or relative?

• Is the solution a state or a path?

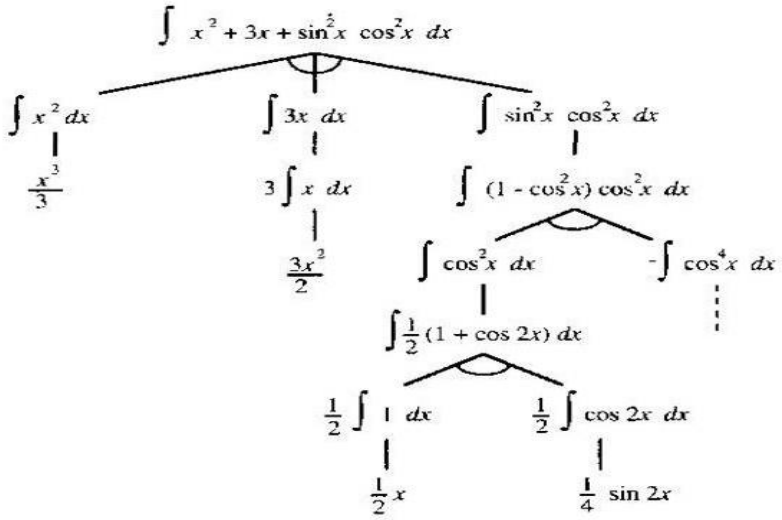
• What is the role of knowledge?

• Does the task require human-interaction?

Is the problem decomposable?

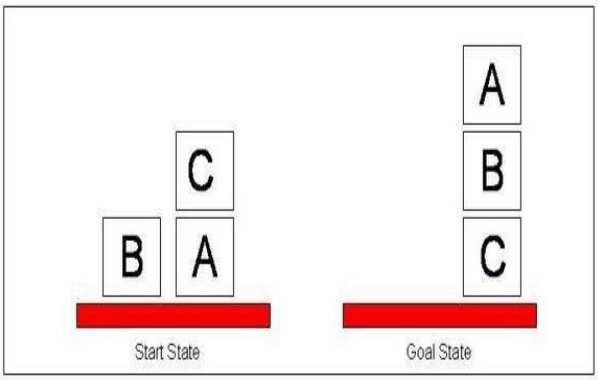
• Can the problem be broken down to smaller problems to be solved independently?

A Decomposable Problem



2015-8-26 EIE426-AICV 35

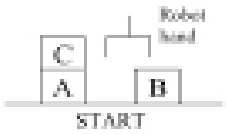
• Decomposable problem can be solved easily.



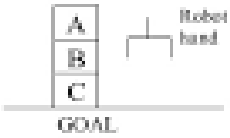
The "Sussman anomaly" blocks-world planning problem.

Example : Blocks World

•STRIPS : A planning system – Has rules with precondition deletion list and addition list



on(B, table)
on(A, table)
on(C, A)
hand empty
clear(C)
clear(B)



on(C, table)
on(B, C)
on(A, B)
hand empty
clear(A)

Start Goal

Blocks World

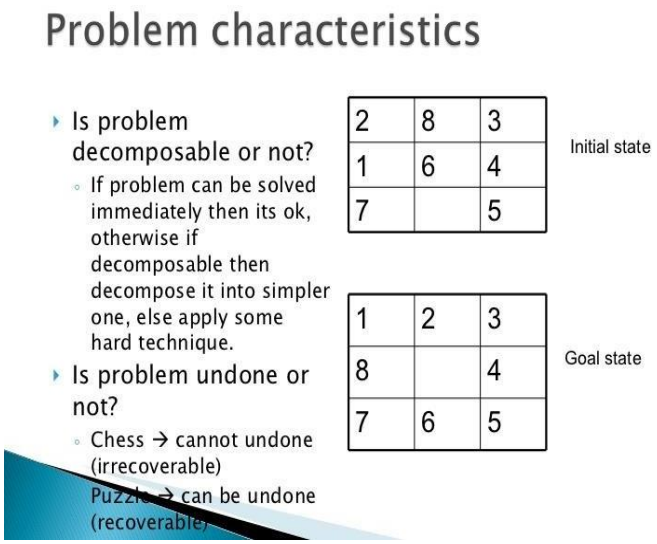
CLEAR(x) → ON(x, Table)

CLEAR(x) and CLEAR(y) → ON(x, y)

ON(B, C) and ON(A, B)

ON(B, C) ON(A, B)

CLEAR(A) ON(A, B)



Can solution steps be ignored or undone?

Theorem proving a lemma that has been proved can be ignored for next steps.

Ignorable!

Can solution steps be ignored or undone?

The 8-Puzzle

3 8 2

3 2 1

4 6 1

Moves can be undone and backtracked.

Recoverable!

7

5

Can solution steps be ignored or undone?

Playing Chess Moves cannot be retracted. Irrecoverable!

• Ignorable problems can be solved using a simple control structure that never backtracks.

• Recoverable problems can be solved using backtracking.

• Irrecoverable problems can be solved by recoverable style methods via planning.

Is the universe predictable?

The 8-Puzzle Every time we make a move, we know exactly what will happen.

Certain outcome!

Few Examples:

- Playing bridge.
- Controlling a robot arm
- Helping a lawyer decide how to defend his against a murder charge.

Playing Bridge We cannot know exactly where all the cards are or what the other players will do on their turns.

Uncertain outcome!

• For certain-outcome problems, planning can be used to generate a sequence of operators that is guaranteed to lead to a solution.

• For uncertain-outcome problems, a sequence of generated operators can only have a good probability of leading to a solution.

Plan revision is made as the plan is carried out and the necessary feedback is provided.

Is a good solution absolute or relative?

1. Marcus was a man.
2. Marcus was a Pompeian.
3. Marcus was born in 40 A.D.
4. All men are mortal.
5. All Pompeians died when the volcano erupted in 79 A.D.
6. No mortal lives longer than 150 years.
7. It is now 2004 A.D.

Is a good solution absolute or relative?

1. Marcus was a man.
2. Marcus was a Pompeian.
3. Marcus was born in 40 A.D.
4. All men are mortal.
5. All Pompeians died when the volcano erupted in 79 A.D.
6. No mortal lives longer than 150 years.
7. It is now 2004 A.D

Is Marcus alive?

1. Marcus was a man.

2. Marcus was a Pompeian.

3. Marcus was born in 40 A.D.

4. All men are mortal.

5. All Pompeians died when the volcano erupted in 79 A.D.

6. No mortal lives longer than 150 years.

7. It is now 2004 A.D.

Is Marcus alive?

Different reasoning paths lead to the answer. It does not matter which path we follow.

The Travelling Salesman Problem We have to try all paths to find the shortest one.

• Any-path problems can be solved using heuristics that suggest good paths to explore.

• For best-path problems, much more exhaustive search will be performed.

Is the solution a state or a path?

Finding a consistent intepretation "The bank president ate a dish of pasta salad with the fork".

– "bank" refers to a financial situation or to a side of a river?

– "dish" or "pasta salad" was eaten?

– Does "pasta salad" contain pasta, as "dog food" does not contain "dog"?

– Which part of the sentence does "with the fork" modify?

What if "with vegetables" is there?

No record of the processing is necessary.

The Water Jug Problem The path that leads to the goal must be reported.

• A path-solution problem can be reformulated as a state-solution problem by describing a state as a partial path to a solution.

• The question is whether that is natural or not.

What is the role of knowledge?

Playing Chess Knowledge is important only to constrain the search for a solution.

Reading Newspaper Knowledge is required even to be able to recognize a solution.

Does the task require human-interaction?

• Solitary problem, in which there is no intermediate communication and no demand for an explanation of the reasoning process.

• Conversational problem, in which intermediate communication is to provide either additional assistance to the computer or additional information to the user.

Problem Classification

• There is a variety of problem-solving methods, but there is no one single way of solving all problems.

• Not all new problems should be considered as totally new. Solutions of similar problems can be exploited.

Search Strategies

Requirements of a good search strategy:

1. It causes motion Otherwise, it will never lead to a solution.

2. It is systematic Otherwise, it may use more steps than necessary.

3. It is efficient Find a good, but not necessarily the best, answer.

Such as BFS,DFS….

Search and Search Techniques:

AI - Popular Search Algorithms

Searching is the universal technique of problem solving in AI. There are some single-player games such as tile games, Sudoku, crossword, etc. The search algorithms help you to search for a particular position in such games.

Single Agent Pathfinding Problems

The games such as 3X3 eight-tile, 4X4 fifteen-tile, and 5X5 twenty four tile puzzles are single-agent-path-finding challenges. They consist of a matrix of tiles with a blank tile. The player is required to arrange the tiles by sliding a tile either vertically or horizontally into a blank space with the aim of accomplishing some objective.

The other examples of single agent pathfinding problems are Travelling Salesman Problem, Rubik's Cube, and Theorem Proving.

Search Terminology

- **Problem Space −** It is the environment in which the search takes place. (A set of states and set of operators to change those states)
- **Problem Instance** − It is Initial state + Goal state.
- **Problem Space Graph** − It represents problem state. States are shown by nodes and operators are shown by edges.
- **Depth of a problem** − Length of a shortest path or shortest sequence of operators from Initial State to goal state.
- **Space Complexity** − The maximum number of nodes that are stored in memory.
- **Time Complexity** − The maximum number of nodes that are created.
- **Admissibility** − A property of an algorithm to always find an optimal solution.
- **Branching Factor** − The average number of child nodes in the problem space graph.
- **Depth −** Length of the shortest path from initial state to goal state.

State Space Search: Summary

1. Define a state space that contains all the possible configurations of the relevant objects.
2. Specify the initial states.
3. Specify the goal states.
4. Specify a set of rules:

*********************Unit 1 Completed*********************

Unit II

Heuristic Search Strategies:

Heuristic (Informed) Search Strategies:

To solve large problems with large number of possible states, problem-specific knowledge needs to be added to increase the efficiency of search algorithms.

Heuristic Evaluation Functions

They calculate the cost of optimal path between two states. A heuristic function for sliding-tiles games is computed by counting number of moves that each tile makes from its goal state and adding these number of moves for all tiles.

Pure Heuristic Search

It expands nodes in the order of their heuristic values. It creates two lists, a closed list for the already expanded nodes and an open list for the created but unexpanded nodes.

In each iteration, a node with a minimum heuristic value is expanded, all its child nodes are created and placed in the closed list. Then, the heuristic function is applied to the child nodes and they are placed in the open list according to their heuristic value. The shorter paths are saved and the longer ones are disposed.

Hill-Climbing Search

It is an iterative algorithm that starts with an arbitrary solution to a problem and attempts to find a better solution by changing a single element of the solution incrementally. If the change produces a better solution, an incremental change is taken as a new solution. This process is repeated until there are no further improvements.

function Hill-Climbing (problem), returns a state that is a local maximum.

inputs: problem, a problem

local variables: current, a node neighbor, a node

current <-Make_Node(Initial-State[problem])

loop

do neighbor <- a highest_valued successor of *current*

if Value[neighbor] ≤ Value[current] then

return State[current]

current <- neighbor

end

Disadvantage –

This algorithm is neither complete, nor optimal.

Local Beam Search

In this algorithm, it holds k number of states at any given time. At the start, these states are generated randomly. The successors of these k states are computed with the help of objective function. If any of these successors is the maximum value of the objective function, then the algorithm stops.

Otherwise the (initial k states and k number of successors of the states = 2k) states are placed in a pool. The pool is then sorted numerically. The highest k states are selected as new initial states. This process continues until a maximum value is reached.

function BeamSearch(problem, k), returns a solution state.

start with k randomly generated states

loop

generate all successors of all k states

if any of the states = solution, then return the state

else select the k best successors

end

Simulated Annealing

Annealing is the process of heating and cooling a metal to change its internal structure for modifying its physical properties. When the metal cools, its new structure is seized, and the

metal retains its newly obtained properties. In simulated annealing process, the temperature is kept variable.

We initially set the temperature high and then allow it to 'cool' slowly as the algorithm proceeds. When the temperature is high, the algorithm is allowed to accept worse solutions with high frequency.

Start

1. **Initialize k = 0; L = integer number of variables;**
2. **From i → j, search the performance difference Δ.**
3. **If Δ <= 0 then accept else if exp(-Δ/T(k)) > random(0,1) then accept;**
4. **Repeat steps 1 and 2 for L(k) steps.**
5. **k = k + 1;**

Repeat steps 1 through 4 till the criteria is met.

End

Travelling Salesman Problem

In this algorithm, the objective is to find a low-cost tour that starts from a city, visits all cities en-route exactly once and ends at the same starting city.

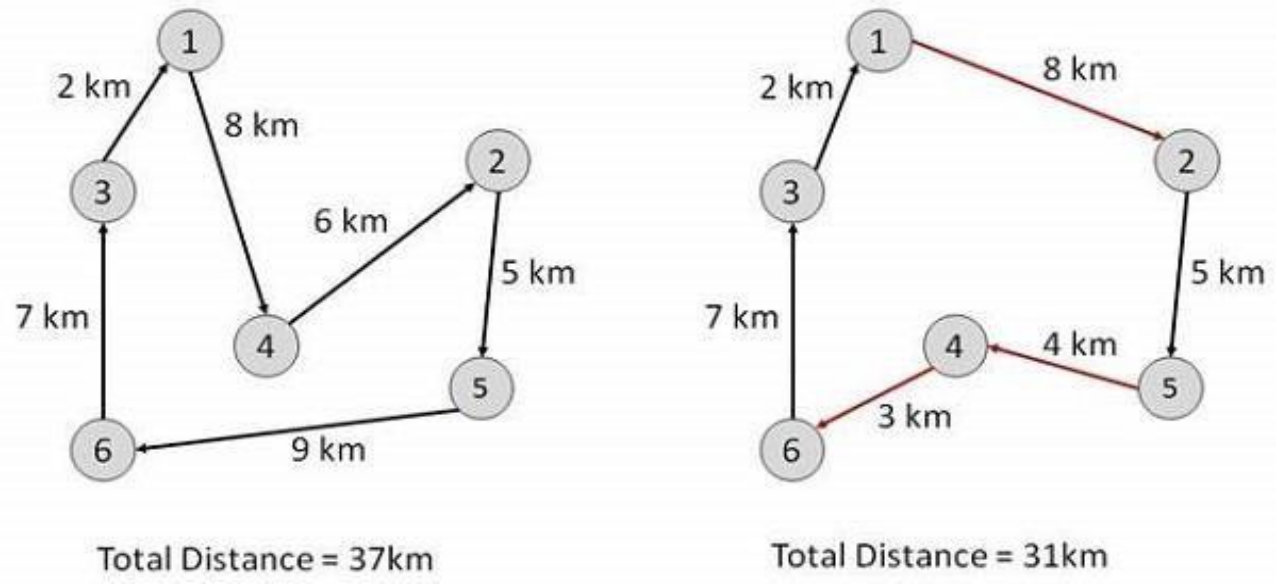
Start

Find out all (n -1)! Possible solutions, where n is the total number of cities.

Determine the minimum cost by finding out the cost of each of these (n -1)! solutions.

Finally, keep the one with the minimum cost.

end



A * Search

It is best-known form of Best First search. It avoids expanding paths that are already expensive, but expands most promising paths first.

f(n) = g(n) + h(n), where

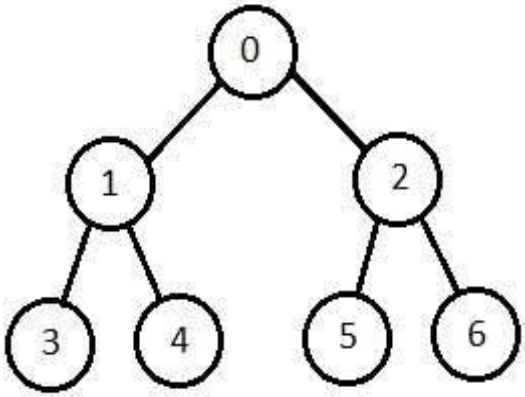
- g(n) the cost (so far) to reach the node
- h(n) estimated cost to get from the node to the goal
- f(n) estimated total cost of path through n to goal. It is implemented using priority queue by increasing f(n).

Greedy Best First Search

It expands the node that is estimated to be closest to goal. It expands nodes based on f(n) = h(n). It is implemented using priority queue.

Disadvantage

It can get stuck in loops. It is not optimal.



Breadth-First Search

It starts from the root node, explores the neighboring nodes first and moves towards the next

level neighbors. It generates one tree at a time until the solution is found. It can be implemented using FIFO queue data structure. This method provides shortest no of nodes created in worst case is $b + b^2 + b^3 + \ldots + b^d$.

Disadvantage –

Since each level of nodes is saved for creating next one, it consumes a lot of memory space. Space requirement to store nodes is exponential.

Its complexity depends on the number of nodes. It can check duplicate nodes.

Bidirectional Search

It searches forward from initial state and backward from goal state till both meet to identify a common state.

The path from initial state is concatenated with the inverse path from the goal state. Each search is done only up to half of the total path.

Knowledge Representation

Knowledge-representation is the field of artificial intelligence that focuses on designing computer representations that capture information about the world that can be used to solve complex problems. The justification for knowledge representation is that conventional procedural code is not the best formalism to use to solve complex problems. Knowledge representation makes complex software easier to define and maintain than procedural code and can be used in expert systems.

For example, talking to experts in terms of business rules rather than code lessens the semantic gap between users and developers and makes development of complex systems more practical.

In simple terms knowledge, representation is a technique which is used in artificial intelligence with the fundamental goal of representing knowledge in a standard manner such that it facilitates inferencing or reasoning or resolution from that knowledge.

It analyzes how to think formally and how to use a symbol to represent a knowledge along with different situation that allows inferencing knowledge representation to help to address the different problems like how do we represent facts about the world, how do we reason about the facts, what kinds of representations are appropriate and how can an agent perform well in reasoning.

There are the variety of ways to represent knowledge or facts which have been exploited in AI programs. In all variety of knowledge representations we deal with two kinds of entities and they are facts which are referred to the truth in some relevant world.

These are the things we want to represent and the second entity will be the representations of facts which are acquired from some chosen formalism. These are the things which we will actually be able to manipulate.

The above two entities can be structured at two levels : The knowledge level of which facts is described. The symbol level at which representations of objects at the knowledge level are defined in terms of symbols that can be manipulated by programs.

The facts and the representations are linked with the two-way mappings. This link is called representation mappings. The forward representation mapping maps from the facts to the representations. The backward representation mapping goes the other way that is from representations to facts.

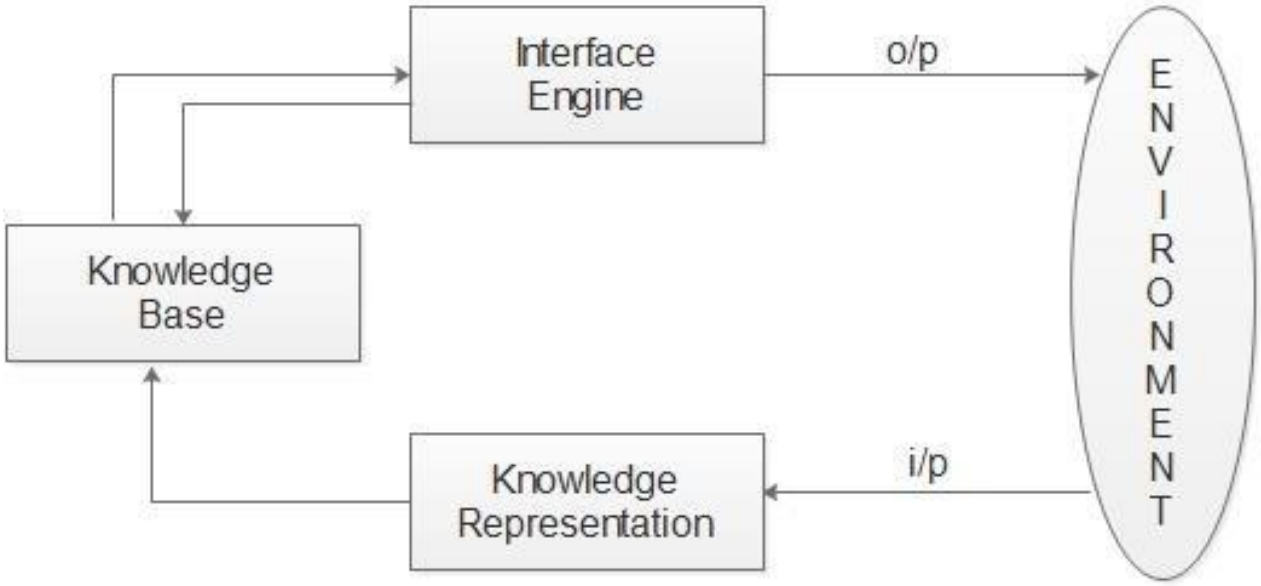
Knowledge representation goes hand in hand with automated reasoning because one of the main purposes of explicitly representing knowledge is to be able to reason about that knowledge, to make inferences, assert new knowledge, etc. Virtually all knowledge representation languages have a reasoning or inference engine as part of the system.

Knowledge representation Issues:

Representation and Mapping:

In simple terms knowledge, representation is a technique which is used in artificial intelligence with the fundamental goal of representing knowledge in a standard manner such that it facilitates inferencing or reasoning or resolution from that knowledge. It analyzes how to think formally and how to use a symbol to represent a knowledge along with different situation that allows inferencing knowledge representation to help to address the different problems like:

1. How do we represent facts about the world?
2. How do we reason about the facts?
3. What kinds of representations are appropriate?
4. How can an agent perform well in reasoning?



For the purpose of solving complex problems that are encountered in artificial intelligence, we need both a large amount of knowledge and some mechanism for manipulating that knowledge in order to create solutions to those new problems. There are the variety of ways to represent knowledge or facts which have been exploited in AI programs. In all variety of knowledge representations we deal with two kinds of entities and they are:

1. Facts: It is referred to the truth in some relevant world. These are the things we want to represent.
2. The second entity will be the representations of facts which are acquired from some chosen formalism. These are the things which we will actually be able to manipulate.

The above two entities can be structured at two levels:

1. The knowledge level of which facts is described.
2. The symbol level at which representations of objects at the knowledge level are defined in terms of symbols that can be manipulated by programs.

The facts and the representations are linked with the two-way mappings. This link is called representation mappings. The forward representation mapping maps from the facts to the representations. The backward representation mapping goes the other way that is from representations to facts.

One common representation is natural language (particularly English) sentences. Regardless of the representation of facts that we use in a program, we may also need to be concerned with an English representation of those facts in order to facilitate for getting the information into and out of the system. We need the mapping functions from English sentences to the representation which we actually use and from it back to sentences.

A good knowledge representation system should have the following approach to representing the given information so that the reasoning becomes easy.

1. **Representational Adequacy**: It is the ability to represent all kinds of knowledge that are needed in a particular domain. This means a good knowledge representation should be able to represent any kind of knowledge in a standard manner.

2. **Inferential Adequacy**: It is the ability to manipulate the different facts that are represented in a standard format in such a way that it derives new structured knowledge from an old one. In short, a good knowledge representation system should be able to infer the new facts from the given facts.

3. **Inferential Efficiency**: It is the ability to derive the new facts from the given fact in an efficient or optimal manner that is a good knowledge representation should be able to incorporate some additional information which can be used to focus the attention of inference engine in the most promising direction.

4. **Acquisitional Adequacy**: It is the ability to acquire new knowledge from the environment in an efficient manner.

Types of Knowledge:

1. **Simple Relational Knowledge**:It is the simplest way of storing facts by using a relational method where each fact about a set of objects is set out systematically in columns. This representation gives little opportunity for inference but it can be used as the knowledge basis for inference engines.A simple way to store facts. Each fact about a set of objects is set out systematically in columns.Little opportunity for inference. Knowledge basis for inference engines.

2. **Inheritable Knowledge**: It is a referential knowledge which is made up of objects that consistof: Attributes Corresponding associated values.

3. **Inferential Knowledge**: It is a representation knowledge as formal logic. For example:

   All dogs have tails.
   $\forall x$: dog(x) $\rightarrow$ has a tail(x).

4. **Procedural Knowledge**: The basic idea of procedural knowledge is to encode the knowledge in some procedures. These procedures may include small programs that know how to do specific things and how to proceed.

Advantages:

- It sets certain rules which are very strict which can be used to derive more facts. The truth of the new statement can be verified.
- It guarantees the correctness.
- Many inference procedures available to in implement standard rules of logic. *e.g* Automated theorem proving.

Approaches to Knowledge Representation:

A good knowledge representation system should have the following approach to representing the given information so that the reasoning becomes easy. Representational Adequacy is the ability to represent all kinds of knowledge that are needed in a particular domain. This means a good knowledge representation should be able to represent any kind of knowledge in a standard manner.

Inferential Adequacy is the ability to manipulate the different facts that are represented in a standard format in such a way that it derives new structured knowledge from an old one. In short, a good knowledge representation system should be able to infer the new facts from the given facts.

Inferential Efficiency is the ability to derive the new facts from the given fact in an efficient or optimal manner that is a good knowledge representation should be able to incorporate some additional information which can be used to focus the attention of inference engine in the most promising direction.

Acquisitional Adequacy is the ability to acquire new knowledge from the environment in an efficient manner. The types of knowledge are simple relational knowledge, inheritable knowledge, inferential knowledge and procedural knowledge. Simple Relational Knowledge is the simplest way of storing facts by using a relational method where each fact about a set of objects is set out systematically in columns.

Inheritable Knowledge is a referential knowledge which is made up of objects that consist of attributes and corresponding associated values. Inferential Knowledge is a representation knowledge as formal logic. The basic idea of procedural knowledge is to encode the knowledge in some procedures. These procedures may include small programs that know how to do specific things and how to proceed.

Issues in Knowledge Representation:

The issues in Knowledge representation are: Are there any attributes of the object which are so basic that they have occurred in almost every problem domain. Are there any important

relationships that exist among the different attributes of objects. At what level should the knowledge is represented and what are the primitives. How should the set of objects be represented? How can the relevant part of the information can be accessed when they are required.

Frame Problem:

It is a technique that is used for knowledge representation. A semantic network is a graphical representation of knowledge consisting of nodes and arcs where the node is the object and arc are the relations between different objects. The most
commonly used relations are:

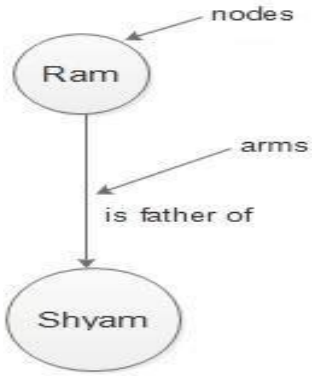
is a,

has a,

member of,

belongs to,

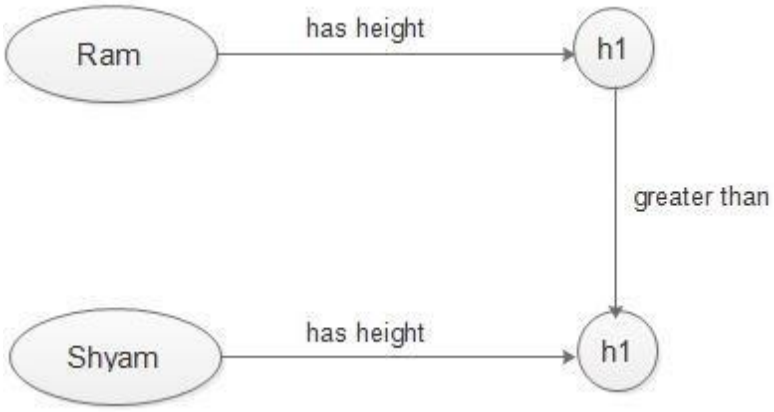
inheritance etc.



Example1: p: Ram is the father of Shyam.

Example2: Ram's height is greater than Shyam's height.

.



Frames:

A frame is a collection of attributes which are usually called slots and their associated values which describe the same entity in the real world. We build a frame system as a collection of frames which are connected to each other by virtue of the fact that the value of an attribute of one frame may be another frame. Frames are also an extensive part of knowledge representation and reasoning scheme. Frames were originally derived from the semantic network. and are therefore part of structure based knowledge representation.

For example:

Person is a: mammal.

Cardinality: 6,00,00,000.

Adult-male is a: person

Cardinality: 2,00,00,000

*height: 6.0".

Football-Player is an: Adult-male.

Cardinality: 1000

*height: 6.2".

Upendra instance: Football-Player.

team: MMFC

score: 4

height: 6.1".

uniform-color: Black.

Each friend represents either a class or an instance. In this example, the friends, person, adult-male, football player are classes whereas the frame Upendra is an instance. There are two types of attributes, one is the attributes associated with class and attributes that are to be inherited by each instance of the class. This is called frame.

*********************Unit 2 Completed*********************

Unit III

Using Predicate Logic:

Proposition is a declarative statement which is either true or false but not both. Truth function is a function to check whether a given statement or expression is true or false. Logic is a set of approach with specific reasoning. The logic that deals with propositions is called a propositional logic.

Let 'p' and 'q' be two propositions. Then, the converse of 'p → q' is 'q → p'. The inverse of 'p → q' is '¬p → ¬q' and the contrapositive of 'p → q' is '¬q → ¬p'.

Tautology is a compound statement which is always true no matter what the truth values of the constituent propositions is. Contradiction is a compound statement which is always false no matter what the truth value of constituent propositions is.

Contingency is a compound statement which is either true or false no matter what the truth value of constituent propositions is. Let 'p' and 'q' be two compound propositions. Then, 'p' is logical equivalence to 'q' if the truth values of 'p' and 'q' are equal.

Representing simple facts in Logic:

It is raining RAINING

It is sunny SUNNY

It is windy WINDY

If it is raining, then it is not sunny RAINING → ¬SUNNY

Predicate Logic Syntax

- Theorem proving is decidable
- Cannot represent objects and quantification
- Can represent objects and quantification
- Theorem proving is semi-decidable
- Constant symbols: a, b, c, John, ...

To represent primitive objects

- Variable symbols: x, y, z, ...

To represent unknown objects

- Predicate symbols: safe, married, love, ...

To represent relations

- married(John) love(John, Mary)
- Function symbols: square, father, ...

To represent simple objects

- safe(square(1, 2))
- love(father(John), mother(John))
- Terms:

To represent complex objects – Constant symbols – If f is a function symbol, and t_1, t_2,... t_n are terms, then so is f(t1, t2, ..., tn) love(mother(father(John)), John)

- Logical connectives: ¬, ∧, ∨, ⇒, ⇔
- Universal quantifier: ∀x: p(x) ∀x: love(father(x), mother(x))
- Existential quantifier: ∃x: p(x) ≡ ¬∀x: ¬p(x)∃x: ¬married(x)
- Sentences:
  - Atomic sentences: p(t1, t2, ..., tn)
  - If α is a sentence, then so are ¬α and (α)
  - If α and β are sentences, then so are α ∧ β, α ∨ β, α ⇒ β,and α ⇔β
  - If α is a sentence, then so are ∀α and ∃α

1. Marcus was a man.
2. Marcus was a Pompeian.
3. All Pompeians were Romans.
4. Caesar was a ruler.
5. All Pompeians were either loyal to Caesar or hated him.
6. Everyone is loyal to someone.
7. People only try to assassinate rulers they are not loyal to.

8. Marcus tried to assassinate Caesar.

9 Using Predicate Logic

1. Marcus was a man. man(Marcus)

2. Marcus was a Pompeian. Pompeian(Marcus)

3. All Pompeians were Romans. ∀x: Pompeian(x) → Roman(x)

4. Caesar was a ruler. ruler(Caesar)

5. All Pompeians were either loyal to Caesar or hated him.

(inclusive-or)

∀x: Roman(x) → loyalto(x, Caesar) ∨ hate(x, Caesar) (exclusive-or)

∀x: Roman(x) → (loyalto(x, Caesar) ∧ ¬hate(x, Caesar)) ∨ (¬loyalto(x, Caesar) ∧ hate(x, Caesar))

6. Every one is loyal to someone.

∀x: ∃y: loyalto(x, y) ∃y: ∀x: loyalto(x, y)

7. People only try to assassinate rulers they are not loyal to.

∀x: ∀y: person(x) ∧ ruler(y) ∧ tryassassinate(x, y)→ ¬loyalto(x, y)

7. People only try to assassinate rulers they are not loyal to.

∀x: ∀y: person(x) ∧ ruler(y) ∧ tryassassinate(x, y)→ ¬loyalto(x, y)

8. Marcus tried to assassinate Caesar. Tryassassinate (Marcus, Caesar)

Was Marcus loyal to Caesar?

man(Marcus)

ruler(Caesar)

tryassassinate(Marcus, Caesar)

⇓ ∀x: man(x) → person(x) ¬loyalto(Marcus, Caesar)

• Many English sentences are ambiguous.

• There is often a choice of how to represent knowledge.

Reasoning:

1. Marcus was a Pompeian.

2. All Pompeians died when the volcano erupted in 79 A.D.

3. It is now 2008 A.D.

Is Marcus alive?

21 26 March, 2009

1. Marcus was a Pompeian.

Pompeian(Marcus)

2. All Pompeians died when the volcano erupted in 79 A.D.

erupted(volcano, 79) ∧ ∀x: Pompeian(x) → died(x, 79)

3. It is now 2008 A.D.

now = 2008

==========

1. Marcus was a Pompeian.

Pompeian(Marcus)

2. All Pompeians died when the volcano erupted in 79 A.D.

erupted(volcano, 79) ∧ ∀x: Pompeian(x) → died(x, 79)

3. It is now 2008 A.D.

now = 2008

∀x: ∀t1: ∀t2: died(x, t1) ∧ greater-than(t2, t1) → dead(x, t2)

• Obvious information may be necessary for reasoning

• We may not know in advance which statements to deduce (P or ¬P).

KB |= α (α is a logical consequence of KB)

How to prove it automatically?

Resolution:

Proof by refutation

KB |= α ⇔ KB ∧ ¬α|= false (empty clause)

Resolution inference rule

(α ∨ ¬β) ∧ (γ ∨ β) premise (α ∨ γ) conclusion

Resolution in Propositional Logic:

1. Convert all the propositions of KB to clause form (S).

L1∨ L2∨ ...∨ Ln P or ¬P

1. Convert all the propositions of KB to clause form (S).

2. Negate α and convert it to clause form. Add it to S.

3. Repeat until either a contradiction is found or no progress can be made:

   a. Select two clauses $(\alpha \vee \neg P)$ and $(\gamma \vee P)$.

   b. Add the resolvent $(\alpha \vee \gamma)$ to S.

Example:

KB = $\{P, (P \wedge Q) \rightarrow R, (S \vee T) \rightarrow Q, T\}$ $\alpha = R$

Example:

KB = $\{P(a), \forall x: (P(x) \wedge Q(x)) \rightarrow R(x), \forall y: (S(y) \vee T(y)) \rightarrow Q(y), T(a)\}$

$\alpha = R(a)$

Unification:

- UNIFY(p, q) = unifier θ where θ(p) = θ(q)
- ∀x: knows(John, x) → hates(John, x) knows(John, Jane) ∀y: knows(y, Leonid) ∀y: knows(y, mother(y)) ∀x: knows(x, Elizabeth)
- ∀x: knows(John, x) → hates(John, x) knows(John, Jane) ∀y: knows(y, Leonid) ∀y: knows(y, mother(y)) ∀x: knows(x, Elizabeth)
- UNIFY(knows(John, x), knows(John, Jane)) = {Jane/x} UNIFY(knows(John, x), knows(y, Leonid)) = {Leonid/x, John/y} UNIFY(knows(John, x), knows(y, mother(y))) = {John/y, mother(John)/x} UNIFY(knows(John, x), knows(x, Elizabeth)) = FAIL 3

Unification: Standardization

UNIFY(knows(John, x), knows(y, Elizabeth)) = {John/y, Elizabeth/x}

Unification: Occur check

UNIFY(knows(x, x), knows(y, mother(y))) = FAIL

Unification: Most general unifier

UNIFY(knows(John, x), knows(y, z)) = {John/y, John/x, John/z} = {John/y, Jane/x, Jane/z} = {John/y, v/x, v/z} = {John/y, z/x, Jane/v} = {John/y, z/x}

Conversion to Clause Form

1. Eliminate →. $P \rightarrow Q \equiv \neg P \vee Q$
2. Reduce the scope of each ¬ to a single term.

$\neg(P \vee Q) \equiv \neg P \wedge \neg Q$ $\neg(P \wedge Q) \equiv \neg P \vee \neg Q$ $\neg \forall x: P \equiv \exists x: \neg P$ $\neg \exists x: p \equiv \forall x: \neg P$ $\neg\neg P \equiv P$

3. Standardize variables so that each quantifier binds a unique variable.

$(\forall x: P(x)) \vee (\exists x: Q(x)) \equiv (\forall x: P(x)) \vee (\exists y: Q(y))$

4. Move all quantifiers to the left without changing their relative order.

$\forall x: (P(x) \vee \exists y: Q(y)) \equiv \forall x: \exists y: (P(x) \vee (Q(y))$ $(\forall x: P(x)) \vee (\exists y: Q(y))$: don't move!

5. Eliminate ∃ (Skolemization).

$\exists x: P(x) \equiv P(c)$ Skolem constant $\forall x: \exists y\ P(x, y) \equiv \forall x: P(x, f(x))$ Skolem function

6. Drop ∀. $\forall x: P(x) \equiv P(x)$.
7. Convert the formula into a conjunction of disjuncts. $(P \wedge Q) \vee R \equiv (P \vee R) \wedge (Q \vee R)$
8. Create a separate clause corresponding to each conjunct.
9. Standardize apart the variables in the set of obtained clauses.

---

1. Eliminate →.
2. Reduce the scope of each ¬ to a single term.
3. Standardize variables so that each quantifier binds a uniquevariable.
4. Move all quantifiers to the left without changing their relativeorder.
5. Eliminate ∃ (Skolemization).
6. Drop ∀.
7. Convert the formula into a conjunction of disjuncts.
8. Create a separate clause corresponding to each conjunct.
9. Standardize apart the variables in the set of obtained clauses.

Resolution in Predicate Logic

1. Convert all the propositions of KB to clause form (S).
2. Negate α and convert it to clause form. Add it to S.
3. Repeat until a contradiction is found:

a. Select two clauses (α ∨ ¬ p(t1)).

b. θ = mgu(p(t1, t2, ..., tn)) and (γ ∨ p(t'1, t'2, ..., t'n , t2, ..., tn), p(t'1, t'2, ..., t'n))

c. Add the resolvent θ(α ∨ γ) to S.

Example:

KB = {P(a), ∀x: (P(x) ∧ Q(x)) → R(x), ∀y: (S(y) ∨ T(y)) → Q(y), T(a)}

α = R(a)

Example

1. Marcus was a man.
2. Marcus was a Pompeian.
3. All Pompeians were Romans.
4. Caesar was a ruler.
5. All Pompeians were either loyal to Caesar or hated him.
6. Every one is loyal to someone.
7. People only try to assassinate rulers they are not loyal to.
8. Marcus tried to assassinate Caesar.

Example

1. Man(Marcus).
2. Pompeian(Marcus).
3. ∀x: Pompeian(x) → Roman(x).
4. ruler(Caesar).
5. ∀x: Roman(x) → loyalto(x, Caesar) ∨ hate(x, Caesar).
6. ∀x: ∃y: loyalto(x, y).
7. ∀x: ∀y: person(x) ∧ ruler(y) ∧ tryassassinate(x, y)→ ¬loyalto(x, y).
8. tryassassinate(Marcus, Caesar).

Example

Prove: hate(Marcus, Caesar)

Question Answering

1. When did Marcus die?
2. Whom did Marcus hate?
3. Who tried to assassinate a ruler?
4. What happen in 79 A.D.?.
5. Did Marcus hate everyone?

Soundness and Completeness

• Soundness of a reasoning algorithm/system R: if KB derives α using R, then KB |=|=|=|= α

• Completeness of a reasoning algorithm/system R: if KB |=|=|=|= α, then KB derives α using R

- Resolution algorithm is sound and complete

• In general:

– Soundness: any returned answer is a correct answer.

– Completeness: all correct answers are returned.

Programming in Logic

PROLOG:

• Only Horn sentences are acceptable

A ← B1, B2, ..., Bm≡ A ∨ ¬B1∨ ¬B2∨ ...∨ ¬Bm A, B i : atoms

PROLOG:

• The occur-check is omitted from the unification: unsound

test ← P(x, x) P(x, f(x))

PROLOG:

• Backward chaining with depth-first search: incomplete

P(x, y) ← Q(x, y) P(x, x) Q(x, y) ← Q(y, x)

PROLOG:

• Unsafe cut: incomplete

A ← B, C ← A

B ← D, !, E

D ← ← B, C← D, !, E, C← !, E, C

PROLOG:

• Negation as failure: ¬P if fails to prove P

Unit III Complted

Unit IV

Representing Knowledge Using Rules:

Types of rules are mostly used in the Rule-based production systems.

1) Knowledge Declarative Rules :

These rules state all the facts and relationships about a problem.

Example :

IF inflation rate declines

THEN the price of gold goes down.

These rules are a part of the knowledge base.

2) Inference Procedural Rules

These rules advise on how to solve a problem, while certain facts areknown.

Example :

IF the data needed is not in the system

THEN request it from the user.

These rules are part of the inference engine.

3) Meta rules

These are rules for making rules. Meta-rules reason about which rules should be considered for firing.

Example :

IF the rules which do not mention the current goal in their premise,

AND there are rules which do mention the current goal in their premise,

THEN the former rule should be used in preference to the latter.

− Meta-rules direct reasoning rather than actually performing reasoning.

− Meta-rules specify which rules should be considered and in which order they should be invoked.

KR – procedural & declarative versus Declarative Knowledge

⇨ **Procedural Knowledge : knowing 'how to do'**

Includes : rules, strategies, agendas, procedures, models.

These explains what to do in order to reach a certain conclusion.

Example

Rule: To determine if Peter or Robert is older, first find their ages.

It is knowledge about 'how to do' something. It manifests itself in the doing of something, e.g., manual or mental skills cannot reduce to words. It is held by individuals in a way which does not allow it to be communicated directly to other individuals.

Accepts a description of the steps of a task or procedure. It Looks similar to declarative knowledge, except that tasks or methods are being described instead of facts or things.

⇨ **Declarative Knowledge : knowing 'what', knowing 'that'**

Includes : concepts, objects, facts, propositions, assertions, models.

It is knowledge about facts and relationships, that It can be expressed in simple and clear statements,

and It can be added and modified without difficulty.

Examples : A car has four tyres; Peter is older than Robert.

Declarative knowledge and explicit knowledge are articulated knowledge and may be treated as synonyms for most practical purposes. Declarative knowledge is represented in a format that can be manipulated, decomposed and analyzed independent of its content.

⇨ **Comparison :**

Comparison between Procedural and Declarative Knowledge:

Procedural Knowledge Declarative Knowledge

- Hard to debug
- Easy to validate
- Black box
- White box
- Obscure
- Explicit
- Process oriented
- Data - oriented
- Extension may effect stability
- Extension is easy
- Fast , direct execution
- Slow (requires interpretation)
- Simple data type can be used
- May require high level data type
- Representations in the form of sets of rules, organized into routines and subroutines.
- Representations in the form of production system, the entire set of rules for executing the task.

Comparison between Procedural and Declarative Language :

Procedural Language Declarative Language

- Basic, C++, Cobol, etc
- SQL
- Most work is done by interpreter of the languages
- Most work done by Data Engine within the DBMS
- For one task many lines of code
- For one task one SQL statement
- Programmer must be skilled in translating the objective into lines of procedural code
- Programmer must be skilled in clearly stating the objective as a SQL statement
- Requires minimum of management around the actual data
- Relies on SQL-enabled DBMS to hold the data and execute the SQL statement .
- Programmer understands and has access to each step of the code
- Programmer has no interaction with the execution of the SQL statement
- Data exposed to programmer during execution of the code
- Programmer receives data at end as an entire set
- More susceptible to failure due to changes in the data structure
- More resistant to changes in the data structure
- Traditionally faster, but that is changing
- Originally slower, but now setting speed records
- Code of procedure tightly linked to front end
- Same SQL statements will work with most front ends Code loosely linked to front end.
- Code tightly integrated with structure of the data store
- Code loosely linked to structure of data; DBMS handles structural issues
- Programmer works with a pointer or cursor
- Programmer not concerned with positioning
- Knowledge of coding tricks applies only to one language
- Knowledge of SQL tricks applies to any language using SQLprogramming offers a formalism for specifying a computation in termsof logical relations between entities.

- logic program is a collection of logic statements.
- programmer describes all relevant logical relationships between the various entities.
- computation determines whether or not, a particular conclusion follows from those logical statements.

• Characteristics of Logic program

Logic program is characterized by set of relations and inferences.

− program consists of a set of axioms and a goal statement.

− rules of inference determine whether the axioms are sufficient to ensure the truth of the goal statement.

− execution of a logic program corresponds to the construction of a proof of the goal statement from the axioms.

− programmer specify basic logical relationships, does not specify the manner in which inference rules are applied.

Thus Logic + Control = Algorithms

• Examples of Logic Statements

− Statement

A grand-parent is a parent of a parent.

− Statement expressed in more closely related logic terms as,

A person is a grand-parent if she/he has a child and that child is a parent.

− Statement expressed in first order logic as(for all) x: grandparent (x, y) :- parent (x, z), parent (z, y) read as x is the grandparent of y if x is a parent of z and z is a parent of y

Programming Language

language includes :

− *the syntax*

− the semantics of programs and

− *the computational model.*

There are many ways of organizing computations. The most familiar paradigm is procedural. The program specifies a computation by saying *"how" it is to be performed. FORTRAN, C, and Object-oriented languages* fall under this general approach.

Another paradigm is declarative. The program specifies a computation by giving the properties of a correct answer. Prolog and logic data language (LDL) are examples of declarative languages, emphasize the logical properties of a computation. Prolog and LDL are called logic programming languages.

PROLOG (PROgramming LOGic) is the most popular Logic programming language rose within the realm of Artificial Intelligence (AI). It became popular with AI researchers, who know more about "what" and "how" intelligent behavior is achieved.

KR – Logic Programming and Terminology (relevant to Prolog programs) language, the formation of components (expressions, statements,etc.), is guided by syntactic rules. The components are divided into two parts: (A) data components and (B) program components.

(A) Data components :

Data components are collection of data objects that follow hierarchy. Data object of any kind is also called

a term. A term is a constant, a variable or a compound term.

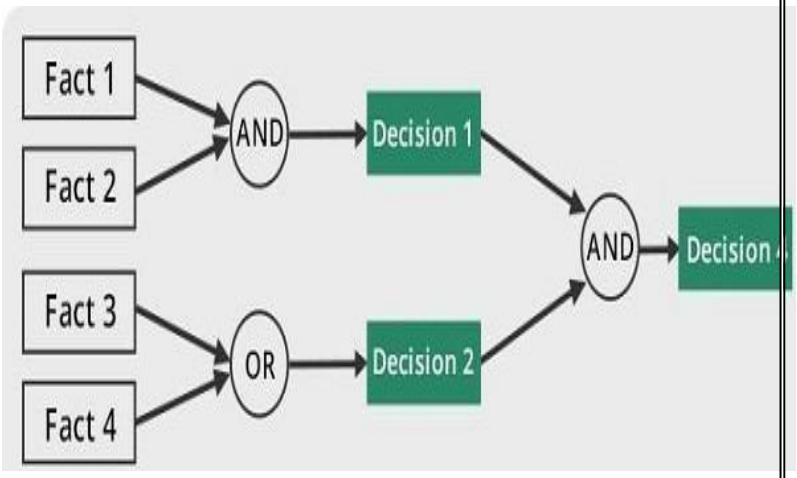
Simple data object is not decomposable; e.g. atoms, numbers, *constants, variables.* Syntax distinguishes the data objects, hence no need for declaring them. **Structured data object are made of** several components.

Forward and Backward Chaining:

Given a set of rules, there are essentially two ways to generate new knowledge: one, forward chaining and the other, backward chaining.

⇨ Forward chaining : also called data driven. It starts with the facts, and sees what rules apply.

⇨ Backward chaining : also called goal driven. It starts with something to find out, and looks for rules that will help in answering it.

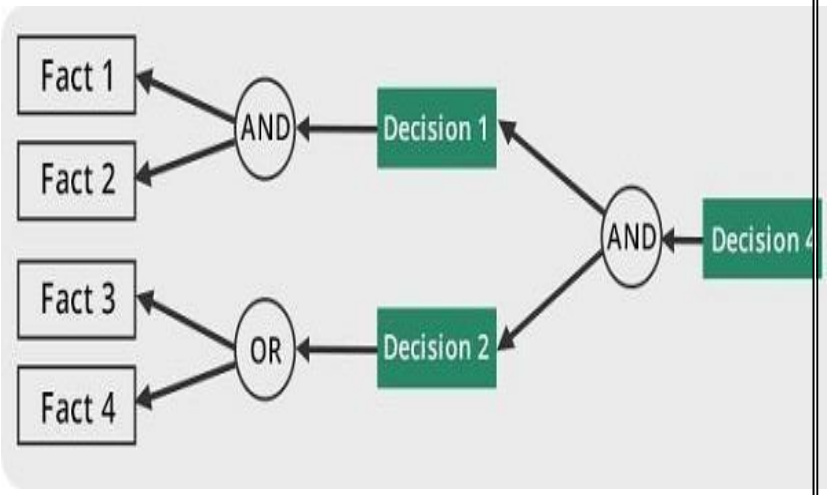


Forward Chaining

It is a strategy of an expert system to answer the question, **"What can happen next?"**

Here, the Inference Engine follows the chain of conditions and derivations and finally deduces the outcome. It considers all the facts and rules, and sorts them before concluding to a solution.

This strategy is followed for working on conclusion, result, or effect. For example, prediction of share market status as an effect of changes in interest rates.



Backward Chaining

With this strategy, an expert system finds out the answer to the question,**"Why this happened?"**

On the basis of what has already happened, the Inference Engine tries to find out which conditions could have happened in the past for this result. This strategy is followed for finding out cause or reason. For example, diagnosis of blood cancer in humans.

Example Rule 1

R1 : KR – forward-backward reasoning ,

IF hot AND smoky THEN fire

Rule R2 : IF alarm_beeps THEN smoky

Rule R3 : IF fire THEN switch_on_sprinklers

Fact F1 : alarm_beeps [Given]

Fact F2 : hot [Given]

■ **Example 2**

Rule R1 : IF hot AND smoky THEN ADD fire

Rule R2 : IF alarm_beeps THEN ADD smoky

Rule R3 : IF fire THEN ADD switch_on_sprinklers

Fact F1 : alarm_beeps [Given]

Fact F2 : hot [Given]

Example Rule KR – forward-backward reasoning

3 : A typical Forward Chaining,

R1 : IF hot AND smoky THEN ADD fire

Rule R2 : IF alarm_beeps THEN ADD smoky

Rule R3 : If fire THEN ADD switch_on_sprinklers

Fact F1 : alarm_beeps [Given]

Fact F2 : hot [Given]

Fact F4 : smoky [from F1 by R2]

Fact F2 : fire [from F2, F4 by R1]

Fact F6 : switch_on_sprinklers [from F2 by R3]

■ **Example 4 : A typical Backward Chaining**

Rule R1 : IF hot AND smoky THEN fire

Rule R2 : IF alarm_beeps THEN smoky

Rule R3 : If _fire THEN switch_on_sprinklers

Fact F1 : hot [Given]

Fact F2 : alarm_beeps [Given]

Goal : Should I switch sprinklers on?

Chaining Forward chaining system, KR – forward chaining ,properties , algorithms, and conflict

resolution strategy are illustrated.

■ **Forward chaining system**

⇨ facts are held in a working memory

⇨ condition-action rules represent actions to be taken when specified facts occur in working memory.

⇨ typically, actions involve adding or deleting facts from the working memory.

■ **Properties of Forward Chaining**

⇨ all rules which can fire do fire.

⇨ can be inefficient - lead to spurious rules firing, unfocused problem solving

⇨ set of rules that can fire known as conflict set.

⇨ decision about which rule to fire is conflict resolution.

Engine facts

KR – forward chaining

■ **Forward chaining algorithm - I**

Repeat

⇨ Collect the rule whose condition matches a fact in WM.

⇨ Do actions indicated by the rule.(add facts to WM or delete facts from WM) Until problem is solved or no condition match

Apply on the Example 2 extended (adding 2 more rules and 1 fact)

Rule R1 : IF hot AND smoky THEN ADD fire

Rule R2 : IF alarm_beeps THEN ADD smoky

Rule R3 : If fire THEN ADD switch_on_sprinklers

Rule R4 : IF dry THEN ADD switch_on_humidifier

Rule R5 : IF sprinklers_on THEN DELETE dry

Fact F1 : alarm_beeps [Given]

Fact F2 : hot [Given]

Fact F2 : Dry [Given]

Now, two rules can fire (R2 and R4)

Rule R4 ADD humidifier is on [from F2]

ADD smoky [from F1]

ADD fire [from F2 by R1]

ADD switch_on_sprinklers [by R3]

Rule R2[followed by sequence of actions]

DELEATE dry, ie humidifier is off a conflict ![by R5]

■ **Forward chaining algorithm - II (applied to example 2 above)**

Repeat

⇨ Collect the rules whose conditions match facts in WM.

⇨ If more than one rule matches as stated above then

Use conflict resolution strategy to eliminate all but one

Do actions indicated by the rules (add facts to WM or delete facts from WM) Until problem is solved or no condition match

Conflict Conflict Resolution set is Strategy

the set of rules that have KR – forward chaining ,their conditions satisfied by working memory elements. Conflict resolution normally selects a single rule to fire.

The popular conflict resolution mechanisms are :*Refractory, Recency, Specificity.*

- **Refractory:** It is a rule should not be allowed to fire more than once on the samedata.

  It discard executed rules from the conflict set.

  It prevents undesired loops.

- **Recency:** It rank instantiations in terms of the recency of the elements in the premise of the rule. The rules which use more recent data are preferred. It is working memory elements are time-tagged indicating at what cycle each fact was added to working memory.

- **Specificity:** The rules which have a greater number of conditions and are therefore more difficult to satisfy, are preferred to more general rules with fewer conditions. There are more specific rules are ‘better’ because they take more of the data into account.

Alternative Instead KR – forward chaining to Conflict Resolution – Use Meta Knowledge,of conflict resolution strategies, sometimes we want to useknowledge in deciding which rules to fire. Meta-rules reason aboutwhich rules should be considered for firing. They direct reasoning ratherthan actually performing reasoning.

Meta-knowledge : knowledge about knowledge to guide search.

Example of meta-knowledgeIF conflict set contains any rule (c , a) such that

a = "animal is mammal" THEN fire (c , a)

This example says meta-knowledge encodes knowledge about how to guide search for solution.

Meta-knowledge, explicitly coded in the form of rules with "object level" knowledge.

KR – backward chaining Chaining:

Chaining system and the algorithm are illustrated.

■ **Backward chaining system**

⇨ Backward chaining means reasoning from goals back to facts.The idea is to focus on the search.

⇨ Rules and facts are processed using backward chaining interpreter.

⇨ *Checks hypothesis, e.g. "should I switch the sprinklers on?"*

■ **Backward chaining algorithm**

⇨ Prove goal G If G is in the initial facts , it is proven. Otherwise, find a rule which can be used to conclude G, and try to prove each of that rule's conditions.

Encoding of rules

Rule R1 : IF hot AND smoky THEN fire

Rule R2 : IF alarm_beeps THEN smoky

Rule R3 : If fire THEN switch_on_sprinklers

Fact F1 : hot [Given]

Fact F2 : alarm_beeps [Given]

Goal : Should I switch sprinklers on?

Depends vs Backward on Chaining *problem, and KR – backward chaining* ,on properties of rule set.

⇨ Backward chaining is likely to be better if there is clear hypotheses.Examples : Diagnostic problems or classification problems, Medical expert systems

⇨ Forward chaining may be better if there is less clear hypothesis and want to see what can be concluded from current situation;

Examples : Synthesis systems - design / configuration.

Knowledge algorithm consists KR – control knowledge**, of**: logic component, that specifies the knowledge to be used in solving problems, and control component, that determines the problem-solving strategies by means of which that knowledge is used.

Thus Algorithm = Logic + Control .

The logic component determines the meaning of the algorithm whereas the control component only affects its efficiency. An algorithm may be formulated in different ways,

producing same behavior. One formulation, may have a clear statement in logic component but employ a sophisticated problem solving strategy in the control component.

The other formulation, may have a complicated logic component but employ a simple problem-solving strategy. The efficiency of an algorithm can often be improved by improving the control component without changing the logic of the algorithm and therefore without changing the meaning of the algorithm. The trend in databases is towards the separation of logic and control.

The programming languages today do not distinguish between them. The programmer specifies both logic and control in a single language. The execution mechanism exercises only the most rudimentary problem-solving capabilities.

Computer programs will be more often correct, more easily improved, and more readily adapted to new problems when programming languages separate logic and control, and when execution mechanisms provide more powerful problem-solving facilities of the kind provided by intelligent theorem-proving systems.

------Unit IV Completed------

Unit V

Game Playing:

What is Game?

• The term Game means a sort of conflict in which n individuals orgroups (known as players) participate.

• Game theory denotes games of strategy.

• John von Neumann is acknowledged as father of game theory. Neumanndefined Game theory in 1928 and 1937 and established the mathematicalframework for all subsequent theoretical developments.

• Game theory allows decision-makers (players) to cope with otherdecision-makers (players) who have different purposes in mind. Inother words, players determine their own strategies in terms of the strategies and goals of their opponent.

• Games are integral attribute of human beings. Games engage the intellectual faculties of humans.

• If computers are to mimic people they should be able to play games.

Overview:

Besides the topic of attraction to the people, has close relation to "intelligence", and its well-defined states and rules. The most commonly used AI technique in game is "Search".

A "Two-person zero-sum game" is most studied game where the two players have exactly opposite goals. Besides there are "Perfect information games"(such as chess and Go) and "Imperfect information games" (such as bridge and games where a dice is used).

Given sufficient time and space, usually an optimum solution can be obtained for the former by exhaustive search, though not for the latter. However, for many interesting games, such a solution is usually too inefficient to be practically used.

Applications of game theory are wide-ranging. Von Neumann and Morgenstern indicated the utility of game theory by linking with economic behavior.

▪ Economic models : For markets of various commodities with differing numbers of buyers and sellers, fluctuating values of supply and demand, seasonal and cyclical variations, analysis of conflicts of interest in maximizing profits and promoting the widest distribution of goods and services.

▪ Social sciences : The n-person game theory has interesting uses in studying the distribution of power in legislative procedures, problems of majority rule,individual and group decision making.

▪ Epidemiologists : Make use of game theory, with respect to immunization procedures and methods of testing a vaccine or other medication.

▪ Military strategists : Turn to game theory to study conflicts of interest resolved through "battles" where the outcome or payoff of a war game is either victory or defeat.

Solitaire is not considered a game by game theory.

The term 'solitaire' is used for single-player games of concentration.

• An instance of a game begins with a player choosing from a set of specified (game rules) alternatives. This choice is called a move.

• After first move, the new situation determines which player to make next move and alternatives available to that player.

▪ In many board games, the next move is by other player.

▪ In many multi-player card games, the player making next move depends on who dealt, who took last trick, won last hand, etc.

• The moves made by a player may or may not be known to other players. Games in which all moves of all players are known to everyone are called *games of perfect information.*

▪ Most board games are games of perfect information.

▪ Most card games are not games of perfect information.

• Every instance of the game must end.

• When an instance of a game ends, each player receives a payoff.

A payoff is a value associated with each player's final situation. A zero-sum game is one in which elements of payoff matrix sum to zero.

In a typical zero-sum game :

▪ win = 1 point,

▪ draw = 0 points, and

▪ loss = -1 points.

Overview Theory

Theory does not prescribe a way or say how to play a game. Game theory is a set of ideas and techniques for analyzing conflict situations between two or more parties. The outcomes are determined by their decisions.

General game theorem : In every two player, zero sum, non-random, perfect knowledge game, there exists a perfect strategy guaranteed to at least result in a tie game.

The frequently used terms :

▪ The term "game" means a sort of conflict in which n individuals or groups (known as players) participate.

▪ A list of "rules" stipulates the conditions under which the game begins.

▪ A game is said to have "perfect information" if all moves are known to each of the players involved.

▪ A "strategy" is a list of the optimal choices for each player at every stage of a given game.

▪ A "move" is the way in which game progresses from one stage to another, beginning with an initial state of the game to the final state.

▪ The total number of moves constitute the entirety of the game.

▪ The payoff or outcome, refers to what happens at the end of a game.

▪ Minimax - The least good of all good outcomes.

▪ Maximin - The least bad of all bad outcomes.

The primary game theory is the Mini-Max Theorem. This theorem says :

"If a Minimax of one player corresponds to a Maximin of the other player, then that outcome is the best both players can hope for Game Playing"

▪ Games can be Deterministic or non-deterministic.

▪ Games can have perfect information or imperfect information.

Games Deterministic Non- Deterministic Perfect information Chess, Checkers, Go, Othello, Tic-tac-toe

Backgammon, Monopoly Imperfect information Navigating a maze Bridge, Poker, Scrabble

Relevance Game Theory and Game Plying:

How relevant the Game theory is to Mathematics, Computer science and Economics is shown in the Fig below.

Glossary of terms in the context of Game Theory

⇨ **Game**

Denotes games of strategy. It allows decision-makers (players) to cope with other decision-makers (players) who have different purposes in mind. In other words, players determine their own strategies in terms of the strategies and goals of their opponent.

⇨ **Player**

Could be one person, two persons or a group of people who share identical interests with respect to the game.

⇨ **Strategy**

A player's strategy in a game is a complete plan of action for whatever situation might arise. It is the complete description of how one will behave under every possible circumstance.

You need to analyze the game mathematically and create a table with "outcomes" listed for each strategy.

A two player strategy table

Players Strategies

Player A Strategy 1

Player A Strategy 2

Player A Strategy 3 etc

Player B Strategy 1 Tie A wins B wins . . .

Player B Strategy 2 B wins Tie A wins . . .

Player B Strategy 3 A wins B wins Tie . . . etc . . . . . .

The MiniMax Search Procedure:

A game can be thought of as a tree of possible future game states. For example, in Gomoku the game state is the arrangement of the board, plus information about whose move it is. The current state of the game is the root of the tree (drawn at the top). In general this node has several children, representing all of the possible moves that we could make.

Each of those nodes has children representing the game state after each of the opponent's moves. These nodes have children corresponding to the possible second moves of the current player, and so on.

The leaves of the tree are final states of the game: states where no further move can be made because one player has won, or perhaps the game is a tie. Actually, in general the tree is a graph, because there may be more than one way to get to a particular state. In some games (e.g., checkers) it is even possible to revisit a prior game state.

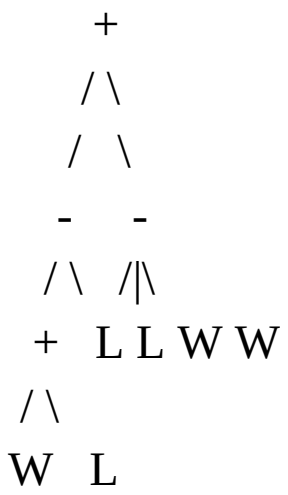
Minimax search

Suppose that we assign a value of positive infinity to a leaf state in which we win, negative infinity to states in which the opponent wins, and zero to tie states. We define a function evaluate that can be applied to a leaf state to determine which of these values is correct.

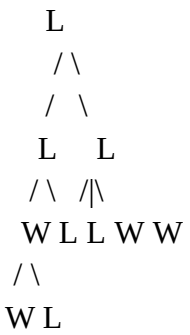
If we can traverse the entire game tree, we can figure out whether the game is a win for the current player assuming perfect play: we assign a value to the current game state by we recursively walking the tree. At leaf nodes we return the appropriate values. At nodes where we get to move, we take the max of the child values because we want to pick the best move; at nodes where the opponent moves we take the min of child values. This gives us the following pseudo-code procedure for minimax evaluation of a game tree.

```
fun minimax(n: node): int =
  if leaf(n) then return evaluate(n)
  if n is a max node
    v := L
    for each child of n
      v' := minimax (child)
      if v' > v, v:= v'
    return v
  if n is a min node
    v := W
    for each child of n
      v' := minimax (child)
      if v' < v, v:= v'
    return v
```

Consider the following game tree, where the leaves are annotated with W or L to indicate a winning or losing position for the current player (L < W), and interior nodes are labeled with + or - to indicate whether they are "max" nodes where we move or "min" nodes where the opponent moves. In this game tree, the position at the root of the tree is a losing position because the opponent can force the game to proceed to an "L" node:



We can see this by doing a minimax evaluation of all the nodes in the tree. Each node is labeled with its minimax value in red:

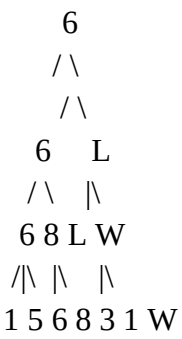


Static evaluation

Usually expanding the entire game tree is infeasible because there are so many possible states. The solution is to only search the tree to a specified depth. The evaluate function (the static evaluator) is extended so it returns a value between L and W for game positions that are not final positions. For game positions that look better for the current player, it returns larger numbers. When the depth limit of the search is exceeded, the static evaluator is applied to the node as if it were a leaf:

```
(* the minimax value of n, searched to depth d *)
 fun minimax(n: node, d: int): int =
   if leaf(n) or depth=0 return evaluate(n)
   if n is a max node
      v := L
      for each child of n
         v' := minimax (child,d-1)
         if v' > v, v:= v'
      return v
   if n is a min node
      v := W
      for each child of n
         v' := minimax (child,d-1)
         if v' < v, v:= v'
      return v
```

For example, consider the following game tree searched to depth 3, where the static evaluator is applied to a number of nodes that are not leaves in the game tree:



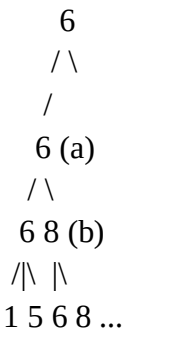
The value of the root of the tree is 6 because the current player can force the game to go to a "leaf" node (as defined by the depth cutoff) whose value is at least 6. Notice that by finding out the value of the current position, the player also learns what is the best move to make: the move that transitions the game to the immediate child with maximum value.

Designing the static evaluator is an art: a good static evaluator should be very fast, because it is the limiting factor in how quickly the search algorithm runs. But it also needs to capture a reasonable approximation of how good the current board position is, because it captures what the player is trying to achieve during play. In practice, game AI designers have found that it doesn't pay to build intelligence into the static evaluator when the same information can be obtained by searching a level or two deeper in the game tree.

How deeply should the tree be searched? This depends on how much time is available to do the search. Each increase in depth multiplies the total search time by about the number of moves available at each level.

Alpha-Beta Pruning

The full minimax search explores some parts of the tree it doesn't have to. For example, consider the tree above and suppose that our search is proceeding in left-to-right order.



Once we have seen the node whose static evaluation is 8, we know that there is no point to exploring any of the rest of the children of the max node above it. Those children could only increase the value of the max node (b) above, but the min node above that (a) is going to have

value at most 6 anyway. No matter what happens in the part of the tree under the ..., it can't affect the minimax value of the min node labeled 6. Avoiding searching a part of a tree is called pruning; this is an example of alpha-beta pruning.

In general the minimax value of a node is going to be worth computing only if it lies within a particular range of values. We can capture this by extending the code of the minimax function with a pair of arguments min and max. The new spec of minimax is that it always returns a value in the range [min, max]. For example, when evaluating the node (b) above, we can set max to 6 because there is no reason to find out about values greater than 6. There are corresponding cases where there is no reason to find out about values less than some minimum value. The min and maxbounds are used to prune away subtrees by terminating a call to search early. Once a child node has been seen that pushes the node's value outside the range of interest, there is no point in exploring the rest of the children. This idea is captured by adding the tests

if v > max return max and if v < min return min in the following code:

```
(* the minimax value of n, searched to depth d.
 * If the value is less than min, returns min.
 * If greater than max, returns max. *)
 fun minimax(n: node, d: int, min: int, max: int): int =
  if leaf(n) or depth=0 return evaluate(n)
  if n is a max node
    v := min
    for each child of n
      v' := minimax (child,d-1,...,...)
      if v' > v, v:= v'
      if v > max return max
    return v
  if n is a min node
    v := max
    for each child of n
      v' := minimax (child,d-1,...,...)
      if v' < v, v:= v'
      if v < min return min
    return v
```

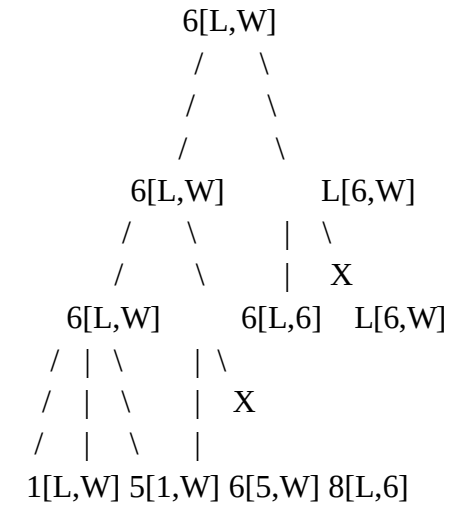
Because we don't care about values less than min or greater than max, we also initialize the variable v to min in the max case and max in the min case, rather than to L and W. Notice that if this procedure is invoked as minimax(n,d,L,W), it will behave just like the minimax procedure without min and max bounds, assuming that the static evaluator only

returns values between Land W. Thus, a top-level search is invoked in this way so that we get the same answer as before pruning.

The only thing missing from our search algorithm now is to compute the right min and max values to pass down. Clearly we could safely pass down the same min and max received in the call, but then we wouldn't have achieved anything. Consider the max node case after we have gone around the loop. In general the variable v will be greater than min. In the recursive invocation ofminimax there is no point to finding out about values less or equal to v; they can't possibly affect the value of v that is returned. Therefore, instead of passing min down in the recursive call, we pass v itself. Conversely, in the min node case, we pass v in place of max:

```
(* the minimax value of n, searched to depth d.
 * If the value is less than min, returns min.
 * If greater than max, returns max. *)
 fun minimax(n: node, d: int, min: int, max: int): int =
   if leaf(n) or depth=0 return evaluate(n)
   if n is a max node
      v := min
      for each child of n
         v' := minimax (child,d-1,v,max)
         if v' > v, v:= v'
         if v > max return max
      return v
   if n is a min node
      v := max
      for each child of n
         v' := minimax (child,d-1,min,v)
         if v' < v, v:= v'
         if v < min return min
      return v
```

This is pseudo-code for minimax search with alpha-beta pruning, or simply alpha-beta search. We can verify that it works as intended by checking what it does on the example tree above. Each node is shown with the [min,max] range that minimax is invoked with. Pruned parts of the tree are marked with X.



In general the [min,max] bounds become tighter and tighter as you proceed down the tree from the root.

Making pruning effective:

How effective is alpha-beta pruning? It depends on the order in which children are visited. If children of a node are visited in the worst possible order, it may be that no pruning occurs. For max nodes, we want to visit the best child first so that time is not wasted in the rest of the children exploring worse scenarios. For min nodes, we want to visit the worst child first (from our perspective, not the opponent's.) There are two obvious sources of this information:

1. The static evaluator function can be used to rank the child nodes
2. Previous searches of the game tree (for example, from previous moves) performed minimax evaluations of many game positions. If available, these values may be used to rank the nodes.

When the optimal child is selected at every opportunity, alpha-beta pruning causes all the rest of the children to be pruned away at every other level of the tree; only that one child is explored. This means that on average the tree can searched twice as deeply as before—a huge increase in searching performance.

Expert Systems:

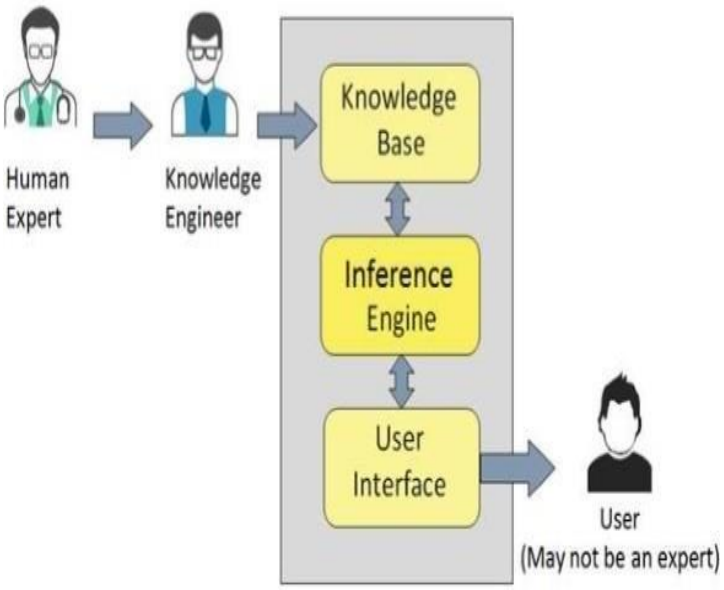
Expert systems (ES) are one of the prominent research domains of AI. It is introduced by the researchers at Stanford University, Computer Science Department.

What are Expert Systems?

The expert systems are the computer applications developed to solve complex problems in a particular domain, at the level of extra-ordinary human intelligence and expertise.

Characteristics of Expert Systems

- High performance
- Understandable
- Reliable
- Highly responsive



Capabilities of Expert Systems

The expert systems are capable of

- Advising
- Instructing and assisting human in decision making
- Demonstrating
- Deriving a solution
- Diagnosing
- Explaining
- Interpreting input
- Predicting results
- Justifying the conclusion
- Suggesting alternative options to a problem

They are incapable of

- Substituting human decision makers
- Possessing human capabilities
- Producing accurate output for inadequate knowledge base
- Refining their own knowledge

Components of Expert Systems

The components of ES include

- Knowledge Base
- Inference Engine
- User Interface

Let us see them one by one briefly

Knowledge Base

It contains domain-specific and high-quality knowledge. Knowledge is required to exhibit intelligence. The success of any ES majorly depends upon the collection of highly accurate and precise knowledge.

What is Knowledge?

The data is collection of facts. The information is organized as data and facts about the task domain. **Data, information,** and **past experience** combined together are termed as knowledge.

Components of Knowledge Base

The knowledge base of an ES is a store of both, factual and heuristic knowledge.

- **Factual Knowledge** − It is the information widely accepted by the Knowledge Engineers and scholars in the task domain.
- **Heuristic Knowledge** − It is about practice, accurate judgement, one's ability of evaluation, and guessing.
-

Knowledge representation

It is the method used to organize and formalize the knowledge in the knowledge base. It is in the form of IF-THEN-ELSE rules.

Knowledge Acquisition

The success of any expert system majorly depends on the quality, completeness, and accuracy of the information stored in the knowledge base.

The knowledge base is formed by readings from various experts, scholars, and the **Knowledge Engineers**. The knowledge engineer is a person with the qualities of empathy, quick learning, and case analyzing skills.

He acquires information from subject expert by recording, interviewing, and observing him at work, etc. He then categorizes and organizes the information in a meaningful way, in the form of IF-THEN-ELSE rules, to be used by interference machine. The knowledge engineer also monitors the development of the ES.

Knowledge acquisition is the process used to define the rules and ontologies required for a knowledge-based system. The phrase was first used in conjunction with expert systems to describe the initial tasks associated with developing an expert system, namely finding and interviewing domain experts and capturing their knowledge via rules, objects, and frame-based ontologies.

Expert systems were one of the first successful applications of artificial intelligence technology to real world business problems. Researchers at Stanford and other AI laboratories worked with doctors and other highly skilled experts to develop systems that could automate complex tasks such as medical diagnosis. Until this point computers had mostly been used to automate highly data intensive tasks but not for complex reasoning. Technologies such as inference engines allowed developers for the first time to tackle more complex problems.

As expert systems scaled up from demonstration prototypes to industrial strength applications it was soon realized that the acquisition of domain expert knowledge was one of if

not the most critical task in the knowledge engineering process. This knowledge acquisition process became an intense area of research on its own. One of the earlier works on the topic used Batesonian theories of learning to guide the process.

One approach to knowledge acquisition investigated was to use natural language parsing and generation to facilitate knowledge acquisition. Natural language parsing could be performed on manuals and other expert documents and an initial first pass at the rules and objects could be developed automatically. Text generation was also extremely useful in generating explanations for system behavior. This greatly facilitated the development and maintenance of expert systems.

A more recent approach to knowledge acquisition is a re-use based approach. Knowledge can be developed in ontologies that conform to standards such as the Web Ontology Language (OWL). In this way knowledge can be standardized and shared across a broad community of knowledge workers. One example domain where this approach has been successful is bioinformatics.

Inference Engine

Use of efficient procedures and rules by the Inference Engine is essential in deducting a correct, flawless solution.

In case of knowledge-based ES, the Inference Engine acquires and manipulates the knowledge from the knowledge base to arrive at a particular solution.

In case of rule based ES, it

- Applies rules repeatedly to the facts, which are obtained from earlier rule application.
- Adds new knowledge into the knowledge base if required.
- Resolves rules conflict when multiple rules are applicable to a particular case.

To recommend a solution, the Inference Engine uses the following strategies

- Forward Chaining
- Backward Chaining

Forward Chaining:

It is a strategy of an expert system to answer the question, **"What can happen next?"**

Here, the Inference Engine follows the chain of conditions and derivations and finally deduces the outcome. It considers all the facts and rules, and sorts them before concluding to a solution.

This strategy is followed for working on conclusion, result, or effect. For example, prediction of share market status as an effect of changes in interest rates.

Backward Chaining

With this strategy, an expert system finds out the answer to the question,**"**Why this happened?"

On the basis of what has already happened, the Inference Engine tries to find out which conditions could have happened in the past for this result. This strategy is followed for finding out cause or reason. For example, diagnosis of blood cancer in humans.

User Interface

User interface provides interaction between user of the ES and the ES itself. It is generally Natural Language Processing so as to be used by the user who is well-versed in the task domain. The user of the ES need not be necessarily an expert in Artificial Intelligence.

It explains how the ES has arrived at a particular recommendation. The explanation may appear in the following forms

- Natural language displayed on screen.
- Verbal narrations in natural language.
- Listing of rule numbers displayed on the screen.

The user interface makes it easy to trace the credibility of the deductions.

Requirements of Efficient ES User Interface

- It should help users to accomplish their goals in shortest possible way.
- It should be designed to work for user's existing or desired work practices.
- Its technology should be adaptable to user's requirements; not the other way round.
- It should make efficient use of user input.

Expert Systems Limitations

No technology can offer easy and complete solution. Large systems are costly; require significant development time, and computer resources. ESs has their limitations which include

- Limitations of the technology
- Difficult knowledge acquisition
- ES are difficult to maintain
- High development costs

Applications of Expert System

The following table shows where ES can be applied.

Application	Description
Design Domain	Camera lens design, automobile design.
Medical Domain	Diagnosis Systems to deduce cause of disease from observed data, conduction medical operations on humans.
Monitoring Systems	Comparing data continuously with observed system or with prescribed behavior such as leakage monitoring in long petroleum pipeline.
Process Control Systems	Controlling a physical process based on monitoring.
Knowledge Domain	Finding out faults in vehicles, computers.
Finance/Commerce	Detection of possible fraud, suspicious transactions, stock market trading, Airline scheduling, cargo scheduling.

Expert System Technology

There are several levels of ES technologies available. Expert systems technologies include ,

- **Expert System Development Environment** − The ES development environment includes hardware and tools. They are
  - ✓ Workstations, minicomputers, mainframes.
  - ✓ High level Symbolic Programming Languages such as **LIS**t**P**rogramming (LISP) and **PRO**grammation en **LOG**ique (PROLOG).
  - ✓ Large databases.
- **Tools** − they reduce the effort and cost involved in developing an expert system to large extent.
  - ✓ P owerful editors and debugging tools with multi-windows.
  - ✓ They provide rapid prototyping

- ✓ Have Inbuilt definitions of model, knowledge representation, and inference design.

- **Shells** − A shell is nothing but an expert system without knowledge base. A shell provides the developers with knowledge acquisition, inference engine, user interface, and explanation facility. For example, few shells are given below −

✓ Java Expert System Shell (JESS) that provides fully developed Java API for creating an expert system.

✓ *Vidwan*, a shell developed at the National Centre for Software Technology, Mumbai in 1993. It enables knowledge encoding in the form of IF-THEN rules.

Development of Expert Systems: General Steps

The process of ES development is iterative. Steps in developing the ES include ,

Identify Problem Domain

- The problem must be suitable for an expert system to solve it.
- Find the experts in task domain for the ES project.
- Establish cost-effectiveness of the system.

Design the System

- Identify the ES Technology
- Know and establish the degree of integration with the other systems and databases.
- Realize how the concepts can represent the domain knowledge best.

Develop the Prototype From Knowledge Base: The knowledge engineer works to −

- Acquire domain knowledge from the expert.
- Represent it in the form of If-THEN-ELSE rules.

Test and Refine the Prototype

- The knowledge engineer uses sample cases to test the prototype for any deficiencies in performance.
- End users test the prototypes of the ES.

Develop and Complete the ES

- Test and ensure the interaction of the ES with all elements of its environment, including end users, databases, and other information systems.
- Document the ES project well.
- Train the user to use ES.

Maintain the ES

- Keep the knowledge base up-to-date by regular review and update.

- Cater for new interfaces with other information systems, as those systems evolve.

Benefits of Expert Systems

- **Availability** − They are easily available due to mass production of software.
- **Less Production Cost** − Production cost is reasonable. This makes them affordable.
- **Speed** − They offer great speed. They reduce the amount of work an individual puts in.
- **Less Error Rate** − Error rate is low as compared to human errors.
- **Reducing Risk** − They can work in the environment dangerous to humans.
- **Steady response** − They work steadily without getting motional, tensed or fatigued.

Explanation :

* The 1st line is the statement "Socrates is a man."

* The 2nd line is a phrase "all human are mortal"into the equivalent "for all X, if X is a man then X is mortal".

* The 3rd line is added to the set to determine the mortality of Socrates.

* The 4th line is the deduction from lines 2 and 3. It is justified by the inference rule modus tollens which states that if the conclusion of a rule is known to be false, then so is the hypothesis.

* Variables X and Y are unified because they have same value.

* By unification, Lines 5, 4b, and 1 produce contradictions and identify Socrates as mortal.

* Note that, resolution is an inference rule which looks for a contradiction and it is facilitated by unification which determines if there is a substitution which makes two terms the same.

Logic model formalizes the reasoning process. It is related to relational data bases and expert systems.

Example Rule 1

R1 : KR – forward-backward reasoning ,

IF hot AND smoky THEN fire

Rule R2 : IF alarm_beeps THEN smoky

Rule R3 : IF fire THEN switch_on_sprinklers

Fact F1 : alarm_beeps [Given]

Fact F2 : hot [Given]

■ **Example 2**

Rule R1 : IF hot AND smoky THEN ADD fire

Rule R2 : IF alarm_beeps THEN ADD smoky

Rule R3 : IF fire THEN ADD switch_on_sprinklers

Fact F1 : alarm_beeps [Given]

Fact F2 : hot [Given]

Example Rule KR – forward-backward reasoning

3 : A typical Forward Chaining,

R1 : IF hot AND smoky THEN ADD fire

Rule R2 : IF alarm_beeps THEN ADD smoky

Rule R3 : If fire THEN ADD switch_on_sprinklers

Fact F1 : alarm_beeps [Given]

Fact F2 : hot [Given]

Fact F4 : smoky [from F1 by R2]

Fact F2 : fire [from F2, F4 by R1]

Fact F6 : switch_on_sprinklers [from F2 by R3]

■ **Example 4 : A typical Backward Chaining**

Rule R1 : IF hot AND smoky THEN fire

Rule R2 : IF alarm_beeps THEN smoky

Rule R3 : If _fire THEN switch_on_sprinklers

Fact F1 : hot [Given]

Fact F2 : alarm_beeps [Given]

Goal : Should I switch sprinklers on?

Chaining Forward chaining system, KR – forward chaining ,properties , algorithms, and conflict

resolution strategy are illustrated.

- **Forward chaining system**

⇨ facts are held in a working memory

⇨ condition-action rules represent actions to be taken when specified facts occur in working memory.

⇨ typically, actions involve adding or deleting facts from the working memory.

- **Properties of Forward Chaining**

⇨ all rules which can fire do fire.

⇨ can be inefficient - lead to spurious rules firing, unfocused problem solving

⇨ set of rules that can fire known as conflict set.

⇨ decision about which rule to fire is conflict resolution.

KR – forward chaining

- **Forward chaining algorithm - I**

Repeat

⇨ Collect the rule whose condition matches a fact in WM.

⇨ Do actions indicated by the rule.(add facts to WM or delete facts from WM) Until problem is solved or no condition match

Apply on the Example 2 extended (adding 2 more rules and 1 fact)

Rule R1 : IF hot AND smoky THEN ADD fire

Rule R2 : IF alarm_beeps THEN ADD smoky

Rule R3 : If fire THEN ADD switch_on_sprinklers

Rule R4 : IF dry THEN ADD switch_on_humidifier

Rule R5 : IF sprinklers_on THEN DELETE dry

Fact F1 : alarm_beeps [Given]

Fact F2 : hot [Given]

Fact F2 : Dry [Given]

Now, two rules can fire (R2 and R4)

Rule R4 ADD humidifier is on [from F2]

ADD smoky [from F1]

ADD fire [from F2 by R1]

ADD switch_on_sprinklers [by R3]

Rule R2[followed by sequence of actions]

DELEATE dry, ie humidifier is off a conflict ![by R5]

■ **Forward chaining algorithm - II (applied to example 2 above)**

Repeat

⇨ Collect the rules whose conditions match facts in WM.

⇨ If more than one rule matches as stated above then

Use conflict resolution strategy to eliminate all but one

Do actions indicated by the rules (add facts to WM or delete facts from WM) Until problem is solved or no condition match

Conflict Conflict Resolution set is Strategy

the set of rules that have KR – forward chaining ,their conditions satisfied by

working memory elements. Conflict resolution normally selects a single rule to fire.

The popular conflict resolution mechanisms are :*Refractory, Recency, Specificity.*

➢ **Refractory:** It is a rule should not be allowed to fire more than once on the samedata.

It discard executed rules from the conflict set.

It prevents undesired loops.

➢ **Recency:** It rank instantiations in terms of the recency of the elements in the premise of the rule. The rules which use more recent data are preferred. It is working memory elements are time-tagged indicating at what cycle each fact was added to working memory.

➢ **Specificity:** The rules which have a greater number of conditions and are therefore more difficult to satisfy, are preferred to more general rules with fewer conditions. There are more specific rules are 'better' because they take more of the data into account.

➢ **Alternative Instead KR – forward chaining to Conflict Resolution – Use Meta Knowledge,**of conflict resolution strategies, sometimes we want to useknowledge in deciding which rules to fire. Meta-rules reason aboutwhich rules should be considered for firing. They direct reasoning ratherthan actually performing reasoning.

Meta-knowledge : knowledge about knowledge to guide search.

Example of meta-knowledgeIF conflict set contains any rule (c , a) such that a = "animal is mammal" THEN fire (c , a) This example says meta-knowledge encodes knowledge about how to guide search for solution. Meta-knowledge, explicitly coded in the form of rules with "object level" knowledge.

KR – backward chaining Chaining

Chaining system and the algorithm are illustrated.

■ Backward chaining system

⇨ Backward chaining means reasoning from goals back to facts.The idea is to focus on the search.

⇨ Rules and facts are processed using backward chaining interpreter.

⇨ *Checks hypothesis, e.g. "should I switch the sprinklers on?"*

■ Backward chaining algorithm

⇨ Prove goal G If G is in the initial facts , it is proven. Otherwise, find a rule which can be used to conclude G, and try to prove each of that rule's conditions.

Encoding of rules

Rule R1 : IF hot AND smoky THEN fire

Rule R2 : IF alarm_beeps THEN smoky

Rule R3 : If fire THEN switch_on_sprinklers

Fact F1 : hot [Given]

Fact F2 : alarm_beeps [Given]

Goal : Should I switch sprinklers on?

Depends vs Backward on Chaining

problem, and KR – backward chaining ,on properties of rule set.

⇨ Backward chaining is likely to be better if there is clear hypotheses.Examples : Diagnostic problems or classification problems, Medical expert systems

⇨ Forward chaining may be better if there is less clear hypothesis and want to see what can be concluded from current situation;

Examples : Synthesis systems - design / configuration.

Knowledge algorithm consists KR – control knowledge**, of**: logic component, that specifies the knowledge to be used in solving problems, and control component, that determines the problem-solving strategies by means of which that knowledge is used.

Thus Algorithm = Logic + Control .

The logic component determines the meaning of the algorithm whereas the control component only affects its efficiency. An algorithm may be formulated in different ways, producing same behavior. One formulation, may have a clear statement in logic component but employ a sophisticated problem solving strategy in the control component.

The other formulation, may have a complicated logic component but employ a simple problem-solving strategy. The efficiency of an algorithm can often be improved by improving the control component without changing the logic of the algorithm and therefore without changing the meaning of the algorithm. The trend in databases is towards the separation of logic and control.

The programming languages today do not distinguish between them. The programmer specifies both logic and control in a single language. The execution mechanism exercises only the most rudimentary problem-solving capabilities.

Computer programs will be more often correct, more easily improved, and more readily adapted to new problems when programming languages separate logic and control, and when execution mechanisms provide more powerful problem-solving facilities of the kind provided by intelligent theorem-proving systems.

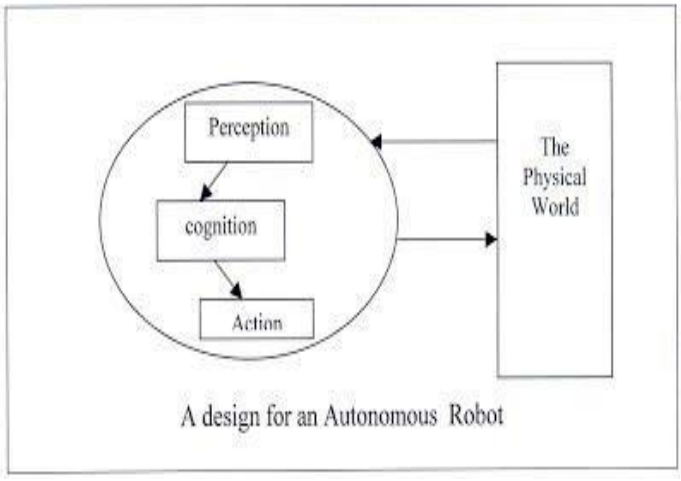
Perception and Action:

Perception:

The definition of AI is based on the nature of the problems it tackles, namely those for which humans currently outperform computers. Also , it includes cognitive tasks. A part from those two aspects, there are many other tasks(that also fall with in this realm) such as basic perceptual and motor skills in which even lower animals posses phenomenal capabilities compared to computers.

Perception involves interpreting sights, sounds, smells and touch. Action includes the ability to negative through the world and manipulate objects. If we want to build robots that live in the world, we must understand these processes. Figure 4.3 shows a design for a complete autonomous robot. Most of AI is concerned with only cognition, we will simply add sensors and effectors to them. But the problems in perception and action are substantial in their own right and are being tackled by researchers in the field of robotics.

In the past, robotics and AI have been largely independent endeavors, and they have developed different techniques to solve different problems. One key difference between AI programs and robots is that AI programs usually operate in computer-stimulated worlds, robots must operate in physical world. For example, in the case of moves in chess, an AI program can search millions of nodes in a game tree without ever having to sense or touch anything in the real world. A complete chess-playing robot, on the



A design for an Autonomous Robot

other hand, must be capable of grasping pieces, visually interpreting board positions, and carrying on a host of other actions. The distinction between real and simulated worlds has several implications as given below:

A design for an Autonomous Robot:

1. The input to an AI program is symbolic in form (example : a typed English sentence), whereas the input to a robot is typically an analog signal ,such as a two dimensional video image or a speech wave form.

2. Robots require special hardware for perceiving and affecting the world, while AI programs require only general-purpose computers.

3. Robot sensors are inaccurate, and their effectors are limited in precision.

4. Many robots must react in real time. A robot fighter plane, for example, cannot afford to search optimally or o stop monitoring the world during a LISP garbage collection.

5. The real world is unpredictable, dynamic, and uncertain. A root cannot hope to maintain a correct and complete description of the world. This means that a robot must consider the trade-off between devising and executing plans. This trade-off has several aspects. For one thing a robot may not possess enough information about the world for it to do any useful planning. In that case, it must first engage in information gathering activity. Furthermore, once it begins executing a plan, the robot must continually the results of its actions. If the results are unexpected, then re-planning may be necessary.

6. Because robots must operate in the real world, searching and back tracking can be costly.

Recent years have seen efforts to integrate research in robotics and AI. The old idea of simply sensors and effectors to existing AI programs has given way to a serious rethinking of basic AI algorithms in light of the problems involved in dealing with the physical world. Research in robotics is likewise affected by AI techniques, since reasoning about goals and plans is essential for mapping perceptions onto appropriate actions.

At this point one might ask whether physical robots are necessary for research purposes. Since current AI programs already operate in simulated worlds, why not build more realistic simulations, which better model the real world? Such simulators do exist. There are several advantages to using a simulated world: Experiment can be conducted very rapidly, conditions can easily be replicated, programs can return to previous states at no cost, and sensory input can be treated no fragile, expensive mechanical parts. The major drawback to simulators is figuring out exactly which factors to build in. experience with real robots continue4s to expose tough problems that do not arise even in the most sophisticated simulators. The world turns out – not surprisingly to be an excellent model of itself, and a readily available one.

We perceive our environment through many channels: sight, sound, touch, smell, taste. Many animals process these same perceptual capabilities, and others also able to monitor entirely different channels. Robots, too, can process visual and auditory information, and they can also equip with more exotic sensors. Such as laser rangefinders, speedometers and radar.

Two extremely important sensory channels for human are vision and spoken language. It is through these two faculties that we gather almost all of the knowledge that drives our problem-solving behaviors.

Vision: Accurate machine vision opens up a new realm of computer applications. These applications include mobile robot navigation, complex manufacturing tasks analysis of satellite images, and medical image processing. The question is that how we can transform raw camera images into useful information about the world.

A Video Camera provides a computer with an image represented as a two-dimensional grid of intensity levels. Each grid element, or pixel, may store a single bit of information (that is , black/white) or many bits(perhaps a real-valued intensity measure and color information). A visual image is composed of thousands of pixels. What kinds of things might we want to do with such an image? Here are four operations, in order of increasing complexity:

1. Signal Processing:- Enhancing the image, either for human consumption or as input to another program.

2. Measurement Analysis:- For images containing a single object, determining the two-dimensional extent of the object depicted.

3. Pattern Recognition:- For single – object images, calssifying the object into a category drawn from a finite set of possibilities.

4. image Understanding :- For images containing many objects, locating the object in the image, classifying them, and building a three-dimensional model of the scene.

There are algorithms that perform the first two operations. The third operation, pattern recognition varies in its difficulty. It is possible to classify two-dimensional (2-D) objects, such as machine parts coming down a conveyor belt, but classifying 3-D objects is harder because of the large number of possible orientations for each object. Image understanding is the most difficult visual task, and it has been the subject of the most study in AI. While some aspects of image understanding reduce to measurement analysis and pattern recognition, the entire problem remains unsolved , because of difficulties that include the following:

1. An image is two-dimensional, while the world is three-dimensional some information is necessarily lost when an image is created.

2. One image may contain several objects, and some objects may partially occlude others.

3. The value of a single pixel is affected by many different phenomena, including the color of the object, the source of the light , the angale and distance of the camera, the pollution in the air, etc. it is hard to disentangle these effects.

As a result, 2-D images are highly ambiguous. Given a single image, we could construct any number of 3-D worlds that would give rise to the image . it is impossible to decide what 3-D solid it should portray. In order to determine the most likely interpretation of a scene , we have to apply several types of knowledge.

Speech Recognition:

Natural Language understanding systems usually accept typed input, but for a number of applications this is not acceptable. Spoken language is a more natural form of communication in many human-computer interfaces. Speech recognition systems have been available for some time, but their limitations have prevented widespread used . Below are five major design issues in speech systems. These issues also provide dimensions along which systems can be compared with one another.

1. Speaker Dependence versus Speaker Independence : A speaker –independent system can liten to any speakear and translate the sounds into written text. Speaker independence ishard to achieve because of the wide variations in pitch and accent. It is easier to build a speaker –dependent system, which can be trained on the voice

Robotics:

Robotics is a domain in artificial intelligence that deals with the study of creating intelligent and efficient robots.

What are Robots?

Robots are the artificial agents acting in real world environment.

What is Robotics?

Robotics is a branch of AI, which is composed of Electrical Engineering, Mechanical Engineering, and Computer Science for designing, construction, and application of robots.

Aspects of Robotics

- The robots have mechanical construction, form, or shape designed to accomplish a particular task.
- They have electrical components which power and control the machinery.
- They contain some level of computer program that determines what, when and how a robot does something.

Difference in Robot System and Other AI Program

Here is the difference between the two

AI Programs	Robots
They usually operate in computer-stimulated worlds.	They operate in real physical world
The input to an AI program is in symbols and rules.	Inputs to robots is analog signal in the form of speech waveform or images
They need general purpose computers to operate on.	They need special hardware with sensors and effectors.

Robot Locomotion

Locomotion is the mechanism that makes a robot capable of moving in its environment. There are various types of locomotions

- Legged
- Wheeled
- Combination of Legged and Wheeled Locomotion
- Tracked slip/skid

Legged Locomotion

- This type of locomotion consumes more power while demonstrating walk, jump, trot, hop, climb up or down, etc.
- It requires more number of motors to accomplish a movement. It is suited for rough as well as smooth terrain where irregular or too smooth surface makes it consume more power for a wheeled locomotion. It is little difficult to implement because of stability issues.
- It comes with the variety of one, two, four, and six legs. If a robot has multiple legs then leg coordination is necessary for locomotion.

The total number of possible gaits (a periodic sequence of lift and release events for each of the total legs) a robot can travel depends upon the number of its legs.

If a robot has k legs, then the number of possible events N = (2k-1)! .

In case of a two-legged robot (k=2), the number of possible events is

N = (2k-1)! = (2*2-1)! = 3! = 6.

Hence there are six possible different events −

- Lifting the Left leg
- Releasing the Left leg
- Lifting the Right leg
- Releasing the Right leg
- Lifting both the legs together
- Releasing both the legs together

In case of k=6 legs, there are 39916800 possible events. Hence the complexity of robots is directly proportional to the number of legs.

Wheeled Locomotion

It requires fewer number of motors to accomplish a movement. It is little easy to implement as there are less stability issues in case of more number of wheels. It is power efficient as compared to legged locomotion.

- Standard wheel − Rotates around the wheel axle and around the contact
- Castor wheel − Rotates around the wheel axle and the offset steering joint.
- Swedish 45^o and Swedish 90^o wheels − Omni-wheel, rotates around the contact point, around the wheel axle, and around the rollers.
- Ball or spherical wheel − Omnidirectional wheel, technically difficult to implement.

Slip/Skid Locomotion

In this type, the vehicles use tracks as in a tank. The robot is steered by moving the tracks with different speeds in the same or opposite direction. It offers stability because of large contact area of track and ground.

Components of a Robot

Robots are constructed with the following −

- Power Supply − The robots are powered by batteries, solar power, hydraulic, or pneumatic power sources.
- Actuators − They convert energy into movement.

- Electric motors (AC/DC) − They are required for rotational movement.
- Pneumatic Air Muscles − They contract almost 40% when air is sucked in them.
- Muscle Wires − They contract by 5% when electric current is passed through them.
- Piezo Motors and Ultrasonic Motors − Best for industrial robots.
- Sensors − They provide knowledge of real time information on the task environment. Robots are equipped with vision sensors to be to compute the depth in the environment. A tactile sensor imitates the mechanical properties of touch receptors of human fingertips.

Computer Vision

This is a technology of AI with which the robots can see. The computer vision plays vital role in the domains of safety, security, health, access, and entertainment.

Computer vision automatically extracts, analyzes, and comprehends useful information from a single image or an array of images. This process involves development of algorithms to accomplish automatic visual comprehension.

Hardware of Computer Vision System

This involves,

- Power supply
- Image acquisition device such as camera
- a processor
- a software
- A display device for monitoring the system
- Accessories such as camera stands, cables, and connectors

Tasks of Computer Vision

- OCR − In the domain of computers, Optical Character Reader, a software to convert scanned documents into editable text, which accompanies a scanner.
- Face Detection − Many state-of-the-art cameras come with this feature, which enables to read the face and take the picture of that perfect expression. It is used to let a user access the software on correct match.
- Object Recognition − They are installed in supermarkets, cameras, high-end cars such as BMW, GM, and Volvo.
- Estimating Position − It is estimating position of an object with respect to camera as in position of tumor in human's body.

Application Domains of Computer Vision

- Agriculture
- Autonomous vehicles
- Biometrics
- Character recognition
- Forensics, security, and surveillance
- Industrial quality inspection
- Face recognition
- Gesture analysis
- Geo science
- Medical imagery
- Pollution monitoring
- Process control
- Remote sensing
- Robotics
- Transport

Applications of Robotics

The robotics has been instrumental in the various domains such as,

- Industries − Robots are used for handling material, cutting, welding, color coating, drilling, polishing, etc.
- Military − Autonomous robots can reach inaccessible and hazardous zones during war. A robot named Daksh, developed by Defense Research and Development Organization (DRDO), is in function to destroy life-threatening objects safely.
- Medicine − The robots are capable of carrying out hundreds of clinical tests simultaneously, rehabilitating permanently disabled people, and performing complex surgeries such as brain tumors.
- Exploration − The robot rock climbers used for space exploration, underwater drones used for ocean exploration are to name a few.
- Entertainment − Disney's engineers have created hundreds of robots for movie making.

==========Unit V Completed==========