MCA/MSc CS 302

ARTIFICIAL INTELLIGENCE AND EXPERT SYSTEMS

Dr. A K Saxena
Professor and Head
GG Central University Bilaspur CG

About course material presented here

- Students are sincerely advised to read various books and consult internet sites for a detailed clarification of topics presented here. Contents are illustrated in class rooms so they must attend classes regularly and attentively. They can however contact teacher at any time in department.
- On suggestions/discussions, the present notes will be modified, so please keep in touch regularly.

Readings

1. **Artificial Intelligence:** A Modern Approach; Stuart Jonathan Russell, Peter Norvig, Prentice Hall, 2010
2. Artificial Intelligence; Elaine Rich, Kevin Knight; Tata McGraw – Hill Publishing Company, 2005

Few internet sites (With Acknowledgments to known / unknown sites for figures / useful literature for academic purpose only)

Unit-1

Introduction

Unit 1- Introduction

Definition and Approaches

(By different Scientists/researchers)

- The goal of work in artificial intelligence is to build machines that perform tasks normally requiring human intelligence. (Nilsson, Nils J. (1971), *Problem-Solving Methods in Artificial Intelligence* (New York: McGraw-Hill): vii.)

- Research scientists in Artificial Intelligence try to get machines to exhibit behavior that we call intelligent behavior when we observe it in human beings. (Slagle, James R. (1971), *Artificial Intelligence: The Heuristic Programming Approach* (New York: McGraw-Hill): 1.)

- **Artificial intelligence** (**AI**) is technology and a branch of computer science that studies and develops intelligent machines and software.
- The exciting new effort to make computers think … *machines with minds*, in the full and literal sense" (Haugeland, 1985)
- The automation of activities that we associate with human thinking, activities such as decision-making, problem solving, learning …" (Bellman, 1978)

Definition and Approaches

- The art of creating machines that perform functions that require intelligence when performed by people" (Kurzweil, 1990)
- The study of how to make computers do things at which, at the moment, people are better" (Rich and Knight, 1991)
- The study of mental faculties through the use of computational models" (Charniak and McDermott, 1985)
- The study of the computations that make it possible to perceive, reason, and act" (Winston, 1992)
- A field of study that seeks to explain and emulate intelligent behavior in terms of computational processes" (Schalkoff, 1990)

- AI seeks to understand the working of the mind in mechanistic terms
- The branch of computer science that is concerned with the automation of intelligent behavior" (Luger and Stubblefield, 1993)

Conclusion about definition of AI

- After an extensive survey of definitions given by scientists and researchers at different times, we can conclude that
- AI is the science of making a machine think and act like an intelligent person.
- Term intelligent person is important to ensure that the actions and thinking that are being imitated and incorporated in machine should be rational.

What is AI?

Views of AI fall into four categories:

- Thinking humanly
- Thinking rationally
- Acting humanly
- Acting rationally

Acting humanly: Turing Test

- The **Turing Test**, proposed by Alan Turing (Turing, 1950), was designed to provide a satisfactory operational definition of intelligence. Turing defined intelligent behavior as the ability to achieve human-level performance in all cognitive tasks, sufficient to fool an interrogator. Roughly speaking, the test he proposed is that the computer should be interrogated by a human via a teletype, and passes the test if the interrogator cannot tell if there is a computer or a human at the other end.

Applications of Artificial Intelligence

Details of following Applications

1. Finance
2. Medical
3. Industries
4. Telephone maintenance
5. Telecom
6. Transport
7. Entertainment
8. Pattern Recognition
9. Robotics
10. Data Mining

Challenges before AI

It is easy said but difficult done, in order to achieve the task of imitating human behaviour or acquiring human intelligence, a machine (a computer in our case) must reflect the following capabilities which are commonly inherited by an intelligent person:

- **Natural language processing** : Like a human, a machine should understand the spirit or the meaning of sentences spoken or written freely in natural language by humans. We don't mind grammar as well as composition of sentences while reading or talking informally.
- **knowledge representation** : It is another great challenge how to express knowledge which can be presented in mathematical or some logical format. Ultimate goal to get a work done by a computer will be to translate the informal sentences into formal ones which could be well interpreted by a computer. We use production systems, semantic nets, frames like structures to express knowledge.
- **automated reasoning** : The capability to use the stored information to answer questions and to draw new conclusions;
- **machine learning** : Learning is an important property of humans. Whatever we wish to act, we learn first then exercise for perfection. A machine should also be able to learn to adapt to new circumstances and to detect and extrapolate patterns.

Time Machine for AI Developments

1943	McCulloch & Pitts: Boolean circuit model of brain
1950	Turing's "Computing Machinery and Intelligence"
1956	Dartmouth meeting: "Artificial Intelligence" adopted
1952—69	Look, Ma, no hands!
1950s	Early AI programs, including Samuel's checkers program, Newell & Simon's Logic Theorist, Gelernter's Geometry Engine
1965	Robinson's complete algorithm for logical reasoning
1966—73	AI discovers computational complexity Neural network research almost disappears
1969—79	Early development of knowledge-based systems
1980--	AI becomes an industry
1986--	Neural networks return to popularity
1987--	AI becomes a science
1995--	The emergence of intelligent agents

Achievements in AI

- Deep Blue defeated the reigning world chess champion Garry Kasparov in 1997
- Proved a mathematical conjecture (Robbins conjecture) unsolved for decades
- No hands across America (driving autonomously 98% of the time from Pittsburgh to San Diego)
- During the 1991 Gulf War, US forces deployed an AI logistics planning and scheduling program that involved up to 50,000 vehicles, cargo, and people
- NASA's on-board autonomous planning program controlled the scheduling of operations for a spacecraft
- Proverb solves crossword puzzles better than most humans

Intelligent Agents

- Agent: In our daily life, an agent is commonly a person who can do our job usually on some obligation. An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors. A human agent has eyes, ears, and other organs for sensors, and hands, legs, mouth, and other body parts for effectors. A robotic agent substitutes cameras and infrared range finders for the sensors and various motors for the effectors. A software agent has encoded bit strings as its percepts and actions; it can produce the square root of any number of any positive number as an example.
- A rational Agent is one which does the things rightly (rationally).
- Performance Evaluation of an agent: How correctly or efficiently an agent serves to our expectation. It could be relative depending on individuals expectations.
- Intelligent Agents: Agents which can transform percepts into actions rationally.

Intelligent Agents

- A calculator is also an agent but it provides no intelligence, just a hard core calculation corrected up to maximum possible value. An intelligent agent on the other hand involves capability to take decision not up to perfection like a hard core agent. E.g. diagnosing a patient on the basis of symptoms and predict disease.
- An agent is composed of following two components
- Agent = architecture + program
- An architecture on which an agent resides is a hardware infrastructure like camera, sensors, videos , computer or any machine. A program usually is a software program to control the architecture to initiate agent. An example of a taxi drive agent is below

Agent Type	Percepts	Actions	Goals	Environment
Taxi driver	Cameras, speedometer, GPS, sonar, microphone	Steer, accelerate, brake, talk to passenger	Safe, fast, legal, comfortable trip, maximize profits	Roads, other traffic, pedestrians, customers

Intelligent Agents

- After reading the table shown for a taxi driver agent, it is apparent that a taxi driver does not require to be recognized as a a human or a program. The percepts are components which agent requires as the inputs like cameras, speedometer to control speed etc.
- **Types of agent programs**
- Simple Reflex Agent: When the actions of nearest object are clearly visible then what response has to be taken, e.g.
  - If the car going ahead applies brake (as appears from brake lights of the front car), then car following it should also initiate brake. In other words, take a counter action against an action (reaction vs action)

Agents that keep track of the world

- To know around if some other car is over taking our car then what to do

Goal based agents

- The goal should be known to the agent by means of a sequence of actions to follow during operation. E.g. the destination should be known to a taxi driver accordingly paths can be derived.

Utility based agents

- The goal should be achieved with some performance measure set by user. The cost, the degree of comfort, safety could be associated with achieving goals.

Unit-1 Environments

- Accessible
  - Whether the sensors of the agent can access complete environment or partially.
- Deterministic
  - Whether the next state can be determined by the current state specifically.
- Episodic
  - Whether the environment's states are available in episodes (serial parts) or all at one time
- Static
  - Whether the environment is changing while the agent is working or remains unchanged.
- Discrete
  - Whether the percepts and actions are distinct and limited like moves in a chess game, or continuous like a running ship/train/non-digital clock (especially seconds arm)/ceiling fan.

- Please see notes on Intelligent Agents(Ask me to collect)
- Details and further reading at various internet CL/DL books, sites

Unit -2

Problem Solving

Production Systems: Systems that generate (produce) rules (states) to reach a solution

A production system commonly consists of following four basic components:

1. A *set of rules of the form* $C_i \longrightarrow A_i$ *where* C_i *refers to starting state and* A_i *represents consequent state. Also* C_i *the condition part and* A_i *is the action* part.

2. One or more *knowledge databases that contain whatever information is relevant* for the given problem.

3. A *control strategy that ascertains the order in which the rules must be applied to the available* database

4. A *rule applier which is the computational system that implements the control* strategy and applies the rules to reach to goal (if it is possible).

State Space

- **State space search** is a process used in which successive or *states* of an instance are considered, with the goal of finding a *goal state* with a desired property.
- State space search often differs from traditional search (sequential, indexed sequential, binary search etc) methods because the state space is *implicit*: the typical state space graph is much too large to generate and store in memory. Instead, nodes are generated as they are explored, and typically discarded thereafter.
- E.g. in a tic tack toe game, every move by a player forms a state space and the three similar (O or X) consecutive (row, column or diagonal) symbols forms goal state.
- In a chess game also state space forms a set of moves by players.
- We discuss water jug problem in the next section.

State Space Search

The water jug problem: You are given two jugs, a 4-litre one and a 3-litre one. Neither has 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.

Let x and y be the amounts of water in 4-Lt and 3-Lt Jugs respectively

Then (x,y) refers to water available at any time in 4-Lt and 3-Lt jugs.

Also (x,y) ⟶ (x-d,y+dd) means drop some unknown amount d of water from 4-Lt jug and add dd onto 3-Lt jug.

All possible production rules can be written as follows

1. (x, y) → (4, y) if x<4, fill it to 4; y remains unchanged
   if x < 4
2. (x, y) → (x, 3) if y<3, fill it to 3; x remains unchanged
   if y < 3
3. (x, y) → (x – d, y) if there is some water in 4 Lt jug, drop
   if x > 0 some more water from it
4. (x, y) → (x, y – d) if there is some water in 3 Lt jug, drop
   if y > 0 some more water from it

State Space Search

5. $(x, y) \rightarrow (0, y)$ if there is some water in 4-Lt, empty it, y remains unchanged
   if $x > 0$

6. $(x, y) \rightarrow (x, 0)$ if there is some water in 3-Lt, empty it, x remains unchanged
   if $y > 0$

7. $(x, y) \rightarrow (4, y - (4 - x))$ if there is some water in 3-Lt, the sum of water of 4-Lt and 3-Lt jug is >=4, then fill water in 4-Lt jug to its capacity from 3-Lt jug
   if $x + y \geq 4, y > 0$

8. $(x, y) \rightarrow (x - (3 - y), 3)$ same as 7 with suitable change in x,y
   if $x + y \geq 3, x > 0$

9. $(x, y) \rightarrow (x + y, 0)$ if sum of water in both jugs <=4, then drop whole water from 3-Lt into 4-Lt
   if $x + y \leq 4, y > 0$

10. $(x, y) \rightarrow (0, x + y)$ if sum of water in both jugs <=3, then drop whole water from 4-Lt into 3-Lt
    if $x + y \leq 3, x > 0$

11. $(0, 2) \rightarrow (2, 0)$ Transfer 2-Lt from 3-Lt jug into empty 4-Lt jug

12. $(2, y) \rightarrow (0, y)$ Empty 2 Lt water onto ground from 4-Lt jug without disturbing 3 Lt jug

State Space Search

Solution of Water Jug Problem

Obviously to solve water jug problem, we can perform following sequence of actions,

$(0,0) \longrightarrow (0,3) \longrightarrow (3,0) \longrightarrow (3,3) \longrightarrow (4,2) \longrightarrow (0,2) \longrightarrow (2,0)$

By applying rules 2,9,2,7,5 and 9 with initial empty jugs

Remember: There is NO hard and fast rules to follow this sequence. In any state space search problem, there can be numerous ways to solve, your approach can be different to solve a problem and sequence of actions too.

State Space Search

Other problems

1. **Cannibals and missionaries problems:** In the missionaries (humans)and cannibals (human eaters) problem, three missionaries and three cannibals must cross a river using a boat which can carry at most two people. At any time number of cannibals on either side should not be greater than number of missionaries otherwise former will eat latter. Also The boat cannot cross the river by itself with no people on board.
2. **Tower of Hanoi Problem**: It consists of three pegs, and a number of disks (usually 60) of different sizes which can slide onto any peg. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a conical shape. The objective of the puzzle is to move the entire stack to another rod, obeying the following rules:
   - Only one disk must be moved at a time.
   - Each move consists of taking the upper disk from one of the rods and sliding it onto another rod, on top of the other disks that may already be present on that rod.
   - No disk may be placed on top of a smaller disk.
3. **Monkey Banana Problem**: A monkey is in a room. A bunch of bananas is hanging from the ceiling and is beyond the monkey's reach. However, in the room there are also a chair and a stick. The ceiling is just the right height so that a monkey standing on a chair could knock the bananas down with the stick. The monkey knows how to move around, carry other things around, reach for the bananas, and wave a stick in the air. What is the best sequence of actions for the monkey?

Search Techniques

- Un-informed (Blind) Search Techniques do not take into account the location of the goal. Intuitively, these algorithms ignore where they are going until they find a goal and report success. Uninformed search methods use only information available in the problem definition and past explorations, e.g. cost of the path generated so far. Examples are
  - **– Breadth-first search (BFS)**
  - **– Depth-first search (DFS)**
  - **– Iterative deepening (IDA)**
  - **– Bi-directional search**

For the minimum cost path problem:

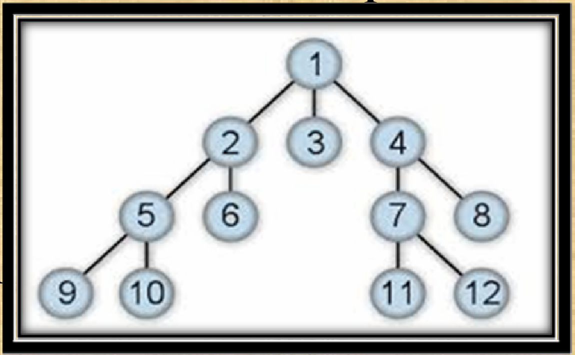
- **Uniform cost search**
- **We will discuss BFS and DFS in next section.**

Un informed Search Techniques

Breadth-first search (BFS): At each level, we expand all nodes (possible solutions), if there exists a solution then it will be found.

Space complexity Order is: O(| V |) where as time complexity is O(| V | + | E |) a graph with V vertex vector and E Edges, |V| means cardinality of V

- **It is complete, optimal, best when space is no problem as it takes much space**



Algorithm BFS

The algorithm uses a queue data structure to store intermediate results as it traverses the graph, as follows:

1. Create a queue with the root node and add its direct children
2. Remove a node in order from queue and examine it
   - If the element sought is found in this node, quit the search and return a result.
   - Otherwise append any successors (the direct child nodes) that have not yet been discovered.
3. If the queue is empty, every node on the graph has been examined – quit the search and return "not found".
4. If the queue is not empty, repeat from Step 2.

Un-informed Search Techniques

Depth-first search (DFS): We can start with a node and explore with all possible solutions available with this node.

Time and Space complexity: Time Order is: O(| V | + | E |) , Space Order is: O(| V |)

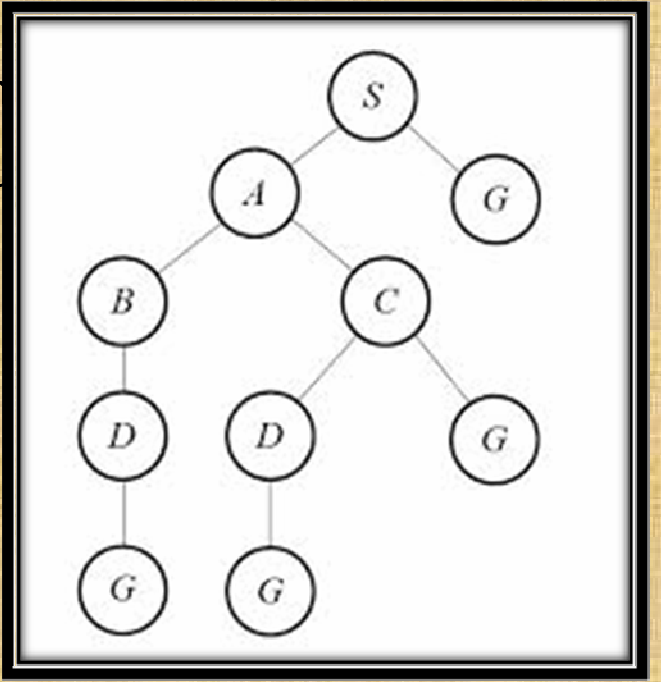
- **It is not complete, non-optimal, may stuck in infinite loop**

DFS starts at the root node and explores as far as possible along each branch before backtracking

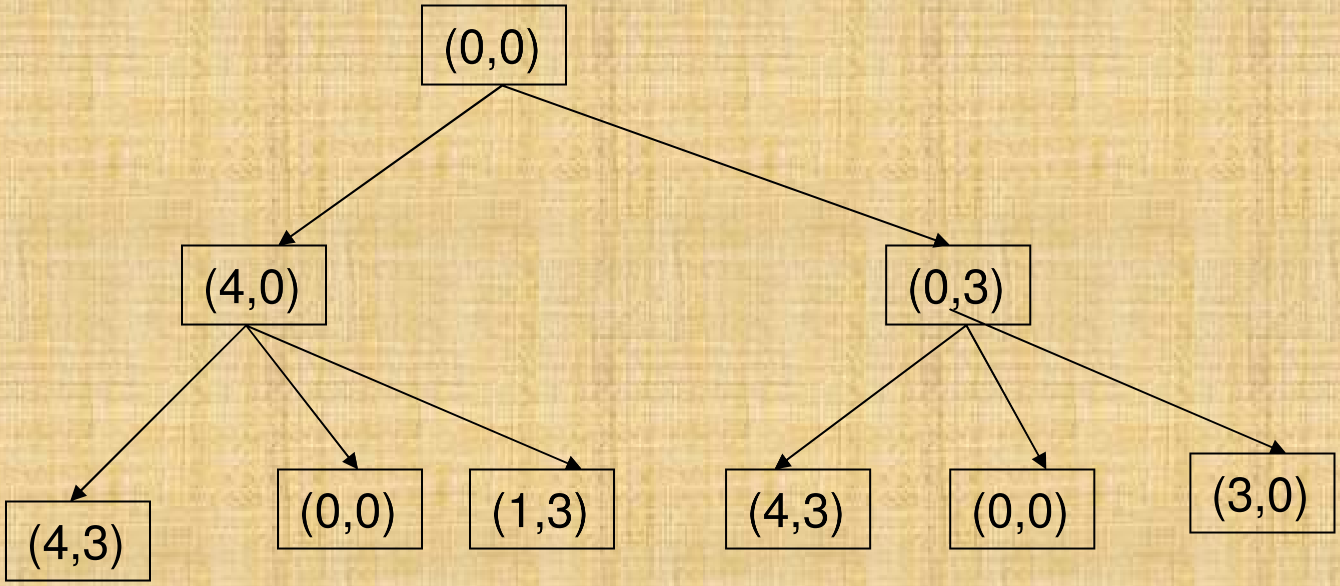
1. Create a stack with the root node and add its direct children
2. Remove a node in order from stack and examine it

   If the element sought is found in this node, quit the search and return a result.

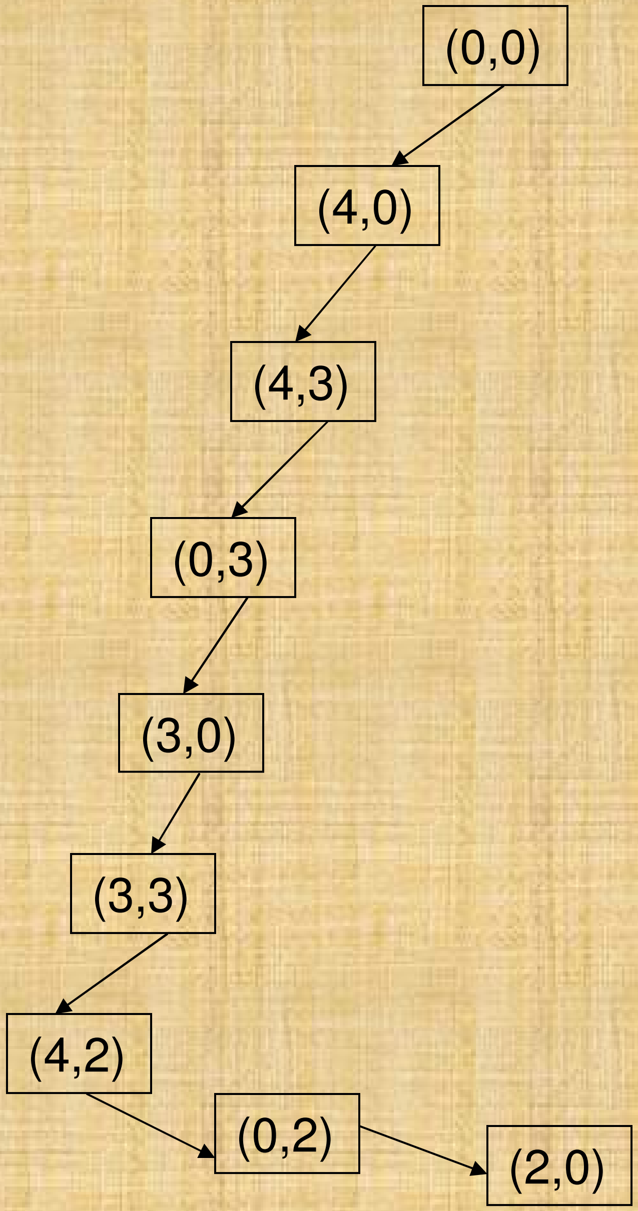
   Otherwise insert any successors (the direct child nodes) that have not yet been discovered before existing nodes.
3. If the stack is empty, every node on the graph has been examined – quit the search and return "not found".
4. If the stack is not empty, repeat from Step 2.



BFS for a water jug problem



DFS for a water jug problem



Search Techniques

- Informed Search Techniques A search strategy which is better than another at identifying the most promising branches of a search-space is said to be more *informed*. It incorporates additional measure of a potential of a specific state to reach the goal. The potential of a state (node) to reach a goal is measured through a **heuristic function.** These are also called intelligent search
- Best first search
- Greedy Search
- A^* search

- In every informed search (Best First or A^* Search), there is a heuristic function and or a local function g(n). The heuristic function at every state decides the direction where next search is to be made.

Algorithm for Greedy Best First Search

Let h(n) be the heuristic function in a graph. In simple case, let it be the straight line distance SLD from a node to destination.

1. Start from source node S, determine all nodes outward from S and queue them.
2. Examine a node from queue (as generated in 1) .
   - If this node is desired destination node, stop and return success.
   - Evaluate h(n) of this node. The node with optimal h(n) gives the next successor, term this node as S.
3. Repeat steps 1 and 2.

Algorithm for A* Search

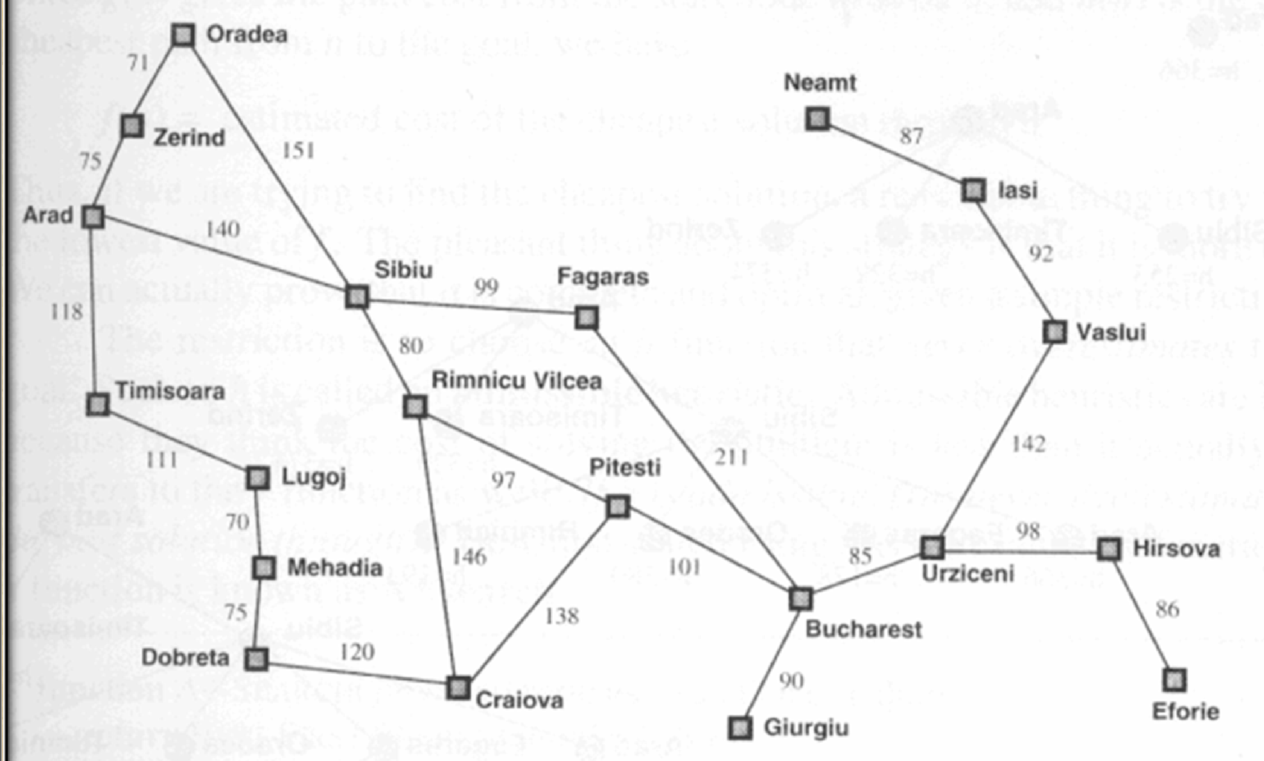
Let h(n) be the heuristic function in a graph. In simple case, let it be the straight line distance SLD from a node to destination. Let g(n) be the function depending on the distance from source to current node. Thus $f(n) = g(n) + h(n)$

1. Start from source node S, determine all nodes outward from S and queue them.
2. Examine a node from queue (as generated in 1) .
   - * If this node is desired destination node, stop and return success.
   - * Evaluate f(n) at this node. The node with optimal f(n) gives the next successor, term this node as S.
3. Repeat steps 1 and 2.

Time = O(log f(n)) where h(n) is the actual distance travelled from n to goal

Best First Search An Example

There are cities in a country (Romania). The task is to reach from A(rad) to B(ucharest)

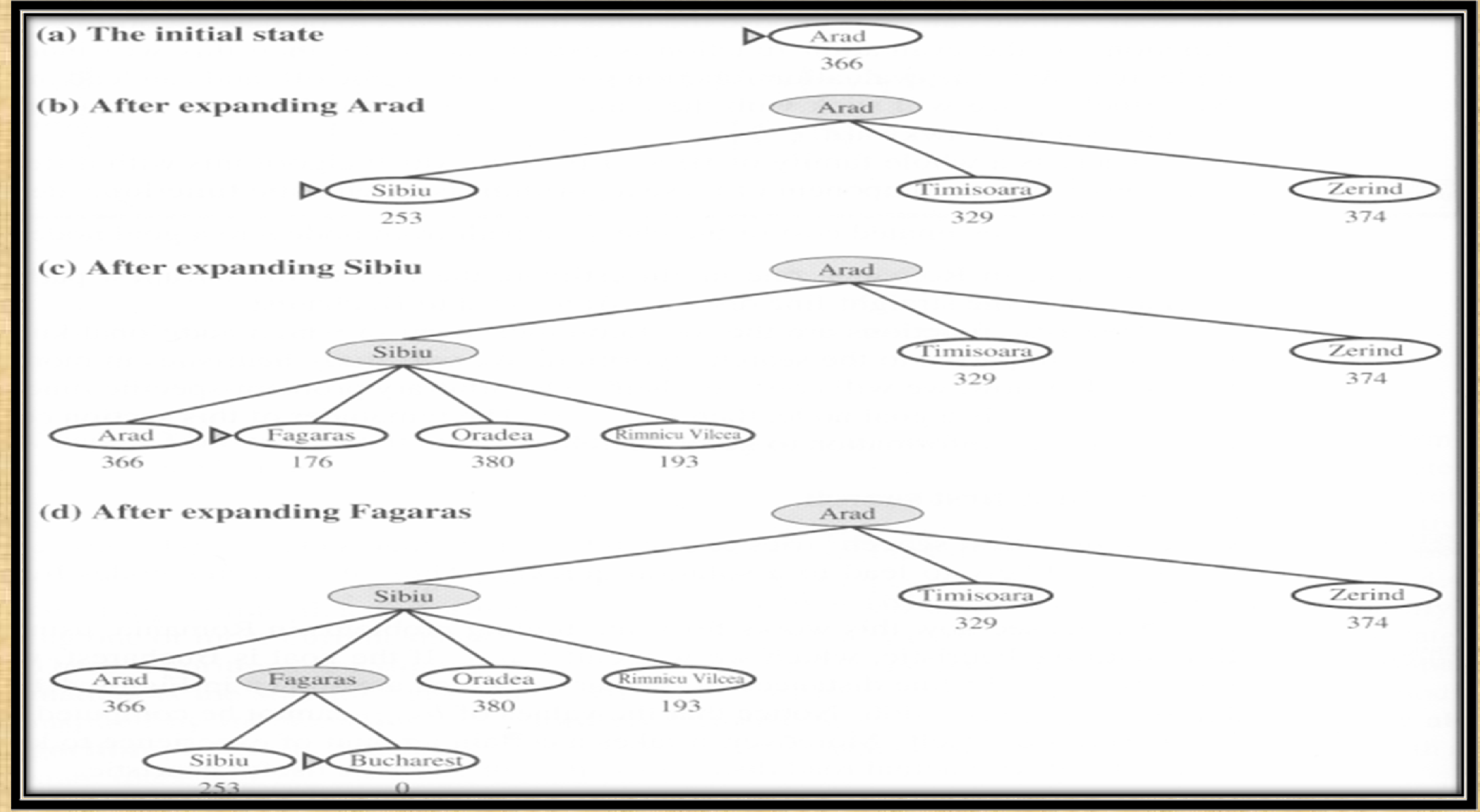


Straight−line distance to Bucharest

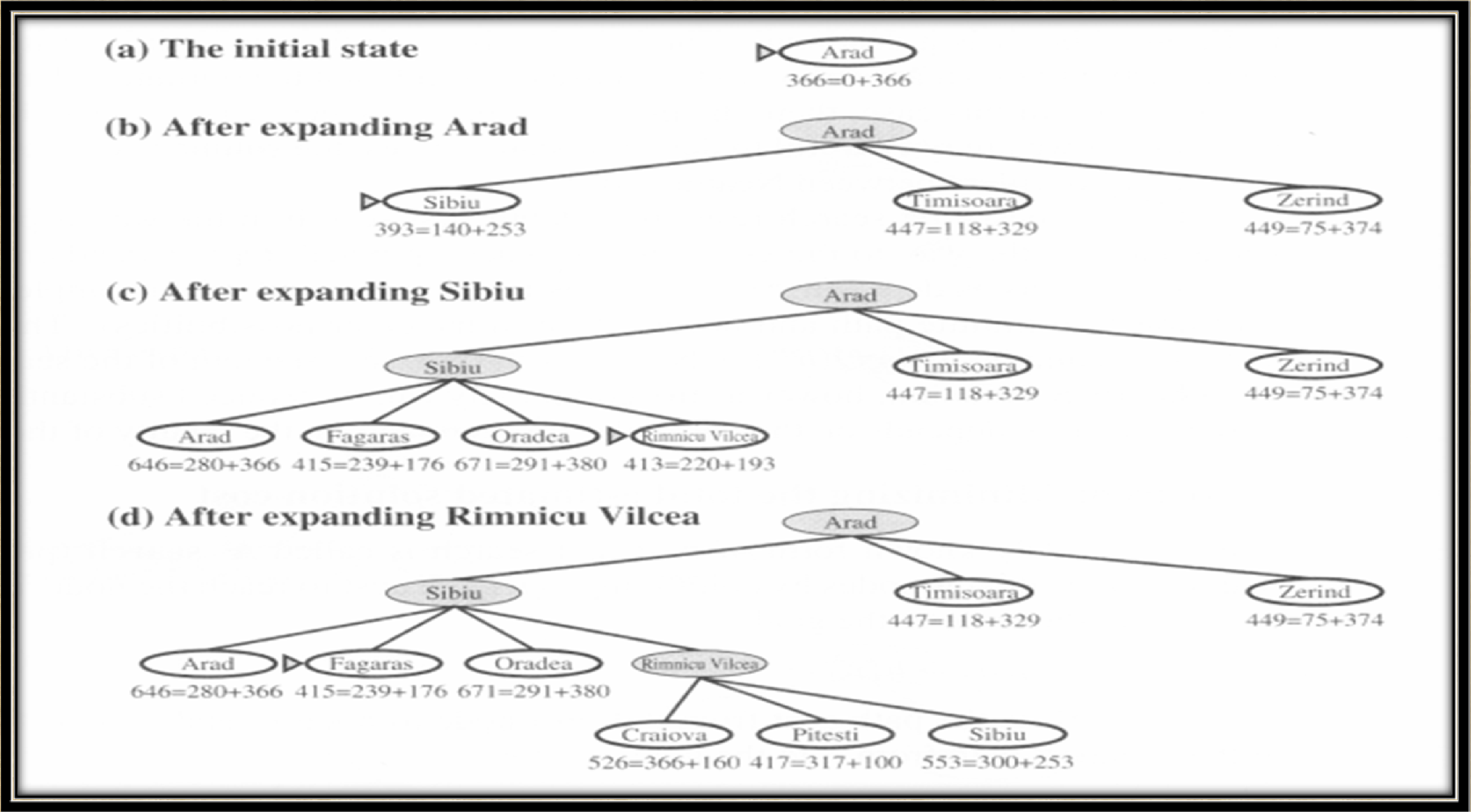
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Figure 4.2 Map of Romania with road distances in km, and straight-line distances to Bucharest.

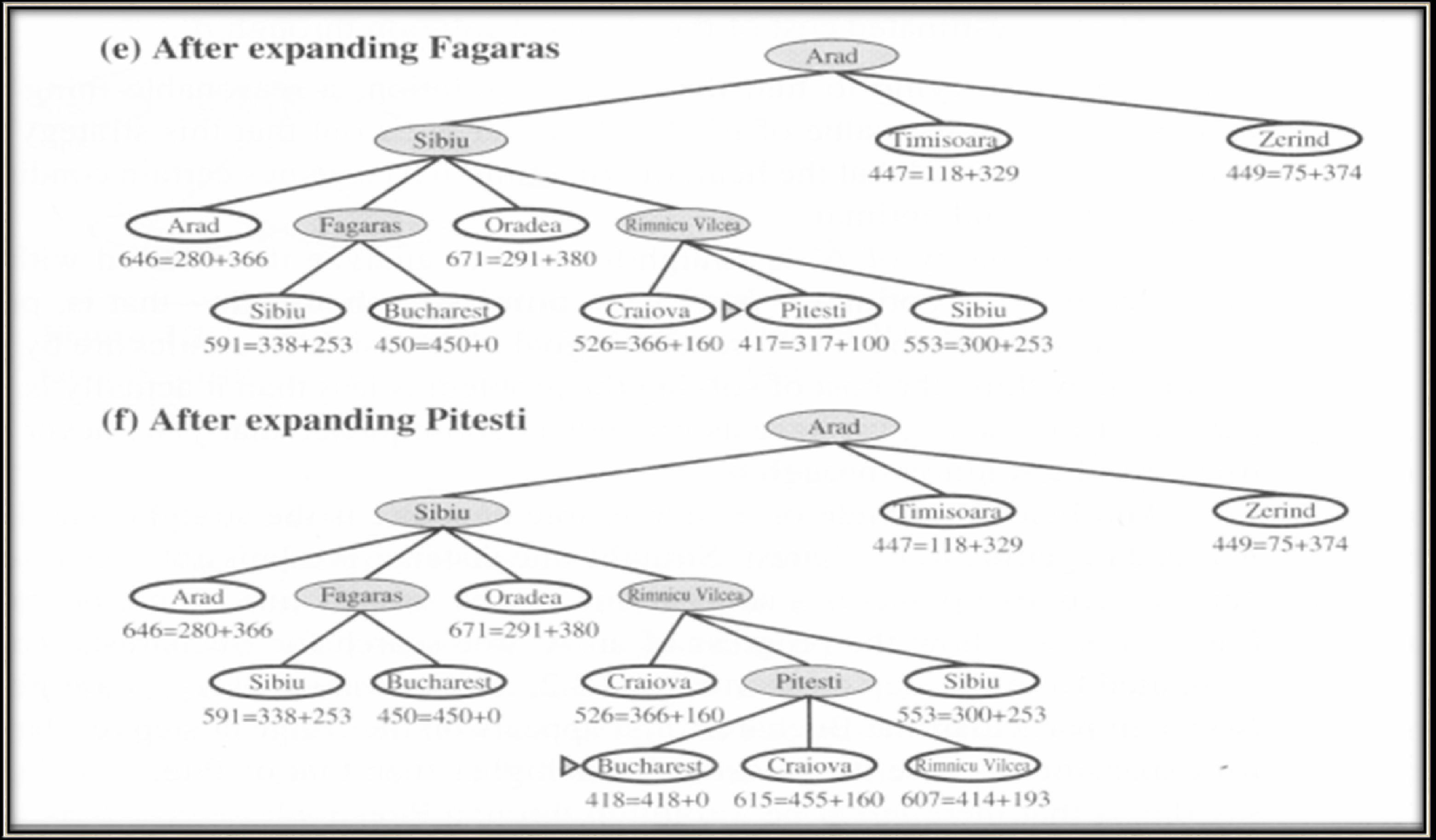
Method: Greedy Best First Search: Start from Source (Arad). At each possible outward node n from S, write the heuristic function h(n). Proceed further in the direction in which h(n) is minimum. Repeat the exercise till goal (destination- Bucharest) is achieved



Method: A* Search: Start from Source (Arad). At each possible outward n, node from S, calculate f(n)=g(n)+h(n), where the heuristic function is h(n) and the total distance travelled so far is g(n). Proceed further in the direction in which h(n)(is minimum. Repeat the exercise till goal (destination- Bucharest) is achieved



Method: A* Search: Start from Source (Arad). At each possible outward node from S, write the heuristic function h(n). Add the total distance travelled so far g(n). Proceed further in the direction in which h(n)(is minimum. Repeat the exercise till goal (destination- Bucharest) is achieved



Heuristics

- Where the exhaustive search is impractical, heuristic methods are used to speed up the process of finding a satisfactory solution via mental shortcuts to ease the cognitive load of making a decision. Examples of this method include using a rule of thumb, an educated guess, an intuitive judgment, stereotyping, or common sense.
- In more precise terms, heuristics are strategies using readily accessible, though loosely applicable, information to control problem solving in human beings and machines. Error and trial is simplest form of heuristics. We can fit some variables in an algebraic equation to solve it.

Local Search Algorithms

- Generate and Test

  1. Generate a possible solution.
  2. Test to see if this is actually a solution.
  3. Quit if a solution has been found. Otherwise, return to step 1.

Features:

1. Acceptable for simple problems.
2. Inefficient for problems with large space.

Local Search Algorithms

- Just operate on a single current state rather than multiple paths
- Generally move only to neighbors of that state
- The paths followed by the search are not retained hence the method is not systematic

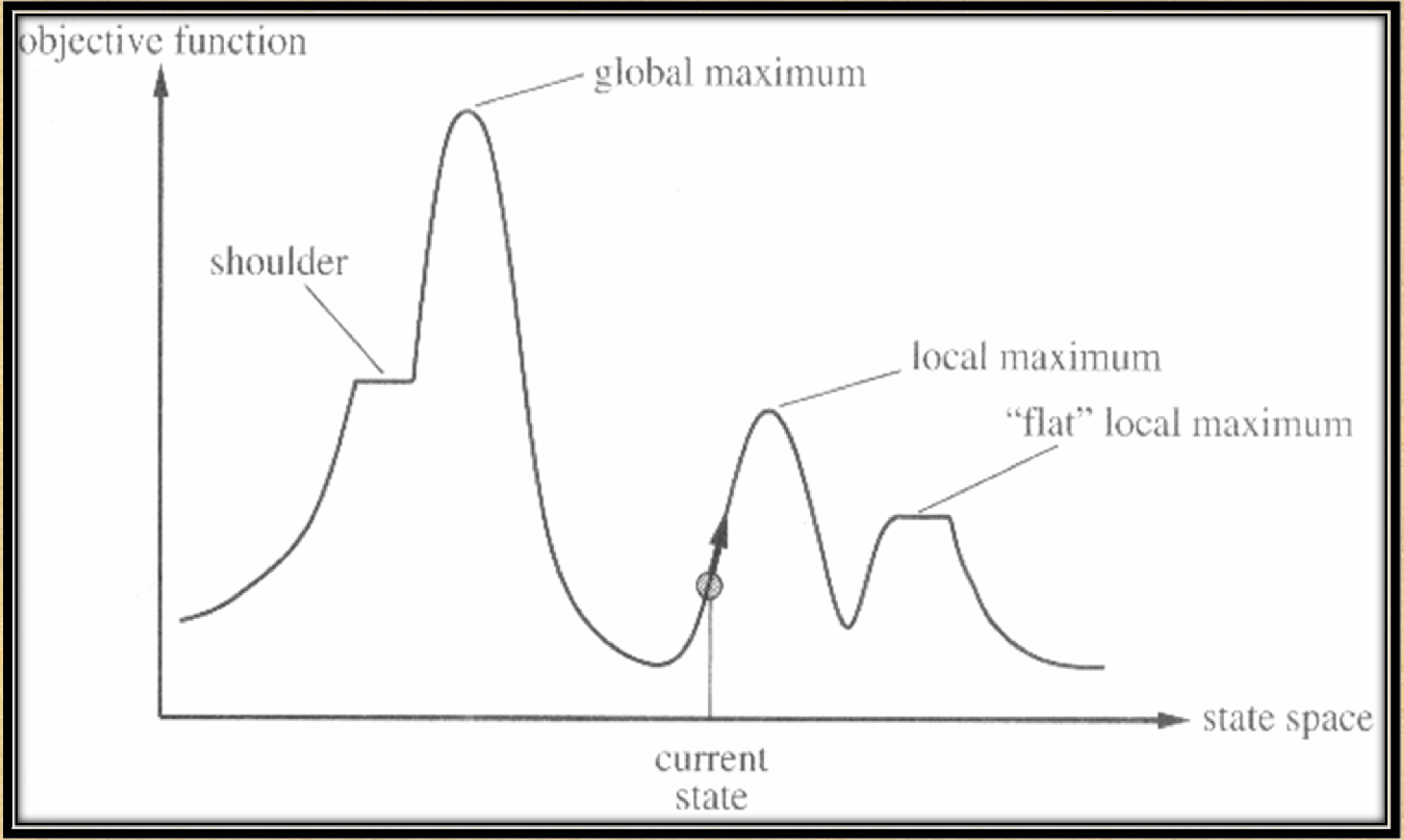
Benefits:

1. uses little memory – a constant amount for current state and some information
2. can find reasonable solutions in large or infinite (continuous) state spaces
   - where systematic algorithms are unsuitable

Local Search

- **State space landscape has two axes**
  - location (defined by states)
  - Elevation or height (defined by objective function or by the value of heuristic cost function)
  - In this figure, the cost refers to global minima and the objective function refers to global maxima(profit e.g.)

Local Search



Local Search: Hill Climbing

- Hill Climbing is an iterative algorithm that starts with an arbitrary solution to a problem, then attempts to find a better solution by incrementally changing a single element of the solution. If the change produces a better solution, an incremental change is made to the new solution, repeating until no further improvements can be found. In **simple hill climbing**, the first closer node is chosen, whereas in **steepest ascent hill climbing** all successors are compared and the closest to the solution is chosen. Both forms fail if there is no closer node, which may happen if there are local maxima in the search space which are not solutions. Steepest ascent hill climbing is similar to best-first search, which tries all possible extensions of the current path instead of only one.

- **Stochastic hill climbing** does not examine all neighbors before deciding how to move. Rather, it selects a neighbor at random, and decides (based on the amount of improvement in that neighbor) whether to move to that neighbor or to examine another.

Local Search: Hill Climbing

Pseudo Algorithm

```
1. Pick initial state s
2. Pick t in neighbors(s) with the largest f(t)
3. IF f(t) <= f(s) THEN stop, return s
4. s = t. GOTO 2.
```

Features:

Not the most sophisticated algorithm in the world.

- Very greedy.
- Easily stuck.

Simple Hill Climbing

1. Evaluate the initial state.
2. Loop until a solution is found or there are no new operators left to be applied:

– Select and apply a new operator

– Evaluate the new state:

  if goal then quit

  otherwise better than current state <- - new current state

Drawbacks: it not try all possible new states!

Gradient Descent

Considers all the moves from the current state.

Selects the best one as the next state.

Example of TSP

Unit-3

Knowledge Representation (KR) and Reasoning

Unit-3 Knowledge Representation (KR) and Reasoning

- In this unit, we discuss and illustrate how knowledge can be represented. As we saw in first unit that KR is a big issue in AI. Converting knowledge to formal sentences which could later be coded is the purpose of KR.
- We start with essential structure of KR. We will brief Natural Language Processing and its importance. We may utter a sentence in many ways and so can be done by our friends, others in the world, without altering the meaning. NLP attempts to read the sentences and provide the meaning reflected in the sentences which must be same in all. On many occasions, it does not matter whether the sentence was stated to represent past, future or present. The meaning matters.

There can be basically following three representations to handle linguistic framework. The branch or study that covers these three issues is known as Meaning–text theory (MTT) which is a linguistic framework for the construction of models of natural language. The theory provides a large and elaborate basis for linguistic description and, due to its formal character, the theory offers itself particularly well to various applications of computers, including machine translation, phraseology and lexicography. There can be three levels of representations: Semantic, Syntactic and morphological.

1. **Semantic Representation:** It is a web-like semantic structure (SemS) which combines with other semantic-level structures (most notably the Semantic-Communicative Structure). In simple words, A *semantic network* or *net* is a graphic notation for representing knowledge in patterns of interconnected nodes and arcs. The structure should represent some logical or valid meaning otherwise It will be rejected e.g. *Red colorless apples have no color* is semantically wrong and must be rejected. In common jargon if we say, *apple eats monkey* is semantically wrong and will be rejected.

2. Syntactic representations are implemented using dependency trees, which constitute the Syntactic Structure (SyntS). SyntS is accompanied by various other types of structure, most notably the syntactic communicative structure and the anaphoric structure. Alternatively linear sequence of words are transformed into structures that relates words to each other. If a word sequence violates the language's rules, it will be rejected. E.g. in English, sentence: *Savita going is school* will be rejected, as it does not conform to English grammar rules.

3. **Morphological representations:** These are implemented as strings of morphemes arranged in a fixed linear order reflecting the ordering of elements in the actual utterance. For morphological analysis, individual words of a sentence are placed into their components and punctuations, special symbols are separated from the words. E.g. Ram's house will be separated in Ram, house, and possessive suffix s (belongs to, of).

Semantic net will be addressed later in details.

Propositional logic and predicate logic

Propositional logic is a study of propositions. Each proposition has either a true or a false value but not both. Propositions can be represented by variables. Usually symbols P and Q represent propositions. Propositions are simply the sentences used alone or combined with other sentences. Hence there can be two types of propositions: simple proposition and compound proposition. A simple or atomic proposition does not contain any other proposition as it's part e.g. Phantom is a dog. The compound proposition contains more than one proposition e.g. Rajiv is a boy and he likes ice cream.

Propositional logic and predicate logic

There can be six types of compound propositions as follows. If p and q are two propositions (sentences) then

1. Negation $\neg p$ indicates the opposite of p.
2. Conjunction $(p \wedge q)$ indicates that p and q both and are enclosed in parenthesis. P and q are called conjuncts
3. Disjunction $(p \vee q)$ indicates that p or q either or both and are enclosed in parenthesis. P and q are called disjuncts.
4. An *implication* $(p \Rightarrow q)$ consists of a pair of sentences separated by the $\Rightarrow$ operator and enclosed in parentheses. The sentence to the left of the operator is called the *antecedent*, and the sentence to the right is called the *consequent*.
5. A *reduction* $(p \Leftarrow q)$ is the reverse of an implication. It consists of a pair of sentences separated by the $\Leftarrow$ operator and enclosed in parentheses. In this case, the sentence to the left of the operator is called the *consequent*, and the sentence to the right is called the *antecedent*.
6. An *equivalence* $(p \Leftrightarrow q)$ is a combination of an implication and a reduction.

Propositional logic and predicate logic

Propositions can not be vague, they have to be true or false but not both

e.g. following sentences are propositions

5+9 =14

11 + 9 = 17

Socrates was a man

It is raining

It is cold

But following sentences are not propositions

Come here

Stand up

X is less than 5

Tautology

In a **tautology** is a formula or proposition which is true in every possible interpretation.

e.g. $p \vee \neg p$ is a tautology (0 OR 1; T V F)

Men are mortal

Truth Tables: A truth table is a table that contains the propositions, well formed formulae (wffs) , sentences as symbols or variables such that each of them has a true or false value only. The table can contain the results due to combining these variables logically.

Take example sentences to explain following

P	Q	$p \wedge q$	$p \vee q$	$p \Rightarrow q$			
T	T	T	T	T			
T	F	F	T	F			
F	T	F	T	T			
F	F	F	F	T			

Strengths and weaknesses of propositional logic

- Propositional logic is easy to represent world knowledge which could be required by an AI system
- Propositional logic is simple to deal with and is declarative.
- Propositional logic permits conjunctive/disjunctive/partial/negative information.
- Real world facts can be written as well-formed formulas (wffs) e.g.

Socrates is a man	SOCRATESMAN
Ramesh is a man	RAMESHMAN
It is cold	COLD

- Propositional logic has very limited expressive power e.g.
  Search a candle in all locality shops has a clear meaning to search all shops in the locality for candle. But propositional logic will require separate statement for each shop
- Propositions could be deceptive or extremely difficult to draw a meaningful conclusion e.g. IRFANMAN and INZMAMMAN produce totally different assertion
- Propositional logic assumes world is all full of facts so constitute wffs accordingly
- Reasoning with propositional logic is difficult.

First Order Logic (FOL) or predicate logic

The limitations of propositional logic can be reduced to some extent by using FOL or predicate logic

A list of basic components / elements / terms used in predicate logic

- Constants: Ramesh, Irfan, 2, 2013, March
- Functions: brotherof, gt, lt,..
- Variables: x,y,z,..
- Predicates have values true or false
- A predicate can take arguments e.g. man(Ramesh), gt(3,2)
- A predicate with one argument shows property of the bracketed argument or object e.g. teacher(Mukesh)
- A predicate with two arguments relates the arguments with each other e.g. Brother(Mukesh, Suresh)
- A predicate without any argument is a proposition or zero order logic
- Quantifiers: Universal ∀ means for all; ∀x: gt (y,x) means for all values of x; y will be greater than x. Existential quantifier ∃ means there exists at least one x, e.g, ∃x: eq(y,x) means there exists at least one x for which y equals to x.

First Order Logic (FOL) or predicate logic

Representation of sentences into predicate logic

- Suppose we want to convert following sentences into Predicate logic

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.**

The sentence wise conversions are displayed as

First Order Logic (FOL) or predicate logic

Representation of sentences into predicate logic

1. Man(Marcus)
2. Pompeian(Marcus)
3. $\forall x$: Pompeian(x) $\rightarrow$ Roman(x)
4. ruler(Caesar)
5. $\forall x$: Roman(x) $\rightarrow$ loyalto(x, Caesar) $\vee$ hate(x, Caesar)
6. $\forall x$: $\exists y$: loyalto(x, y)
7. $\forall x$: $\forall y$: person(x) $\wedge$ ruler(y) $\wedge$ tryassassinate(x, y) $\rightarrow$ $\neg$loyalto(x, y)
8. tryassassinate(Marcus, Caesar)

Now think how will you prove that Marcus was not loyal to Caesar.

First Order Logic (FOL) or predicate logic

Representation of sentences into predicate logic

- Few more facts about Marcus

1. **Marcus was a man.**
2. **Marcus was a Pompeian.**
3. **Marcus was born in 40 AD.**
4. **All men are mortal**
5. **Volcano erupted in 79 AD.**
6. **All Pompeians died in 79 AD.**
7. **No mortal lives longer than 150 years.**
8. **It is 2013.**
9. **Alive means not dead.**
10. **If some one dies then he is dead at all later times.**

In class room, several examples are conducted, however conversions are on next page

First Order Logic (FOL) or predicate logic

Representation of sentences into predicate logic

1. man(Marcus)
2. Pompeian(Marcus)
3. born (Marcus, 40)
4. $\forall x$: man(x) $\rightarrow$ mortal(x)
5. erupted (volcano, 79)
6. $\forall x$: Pompeian(x) $\rightarrow$ died(x,79)
7. $\forall x$: $\forall t1$: $\forall t2$: mortal(x) $\wedge$ born(x,t1) $\wedge$ gt(t2-t1,150) $\rightarrow$ dead(x,t2)
8. Now=2013.
9. $\forall x$: $\forall t$:[alive(x,t) $\rightarrow \neg$dead(x,t)] $\wedge$ [$\neg$dead(x,t) $\rightarrow$ alive(x,t)]
10. $\forall x$: $\forall t1$: $\forall t2$:died(x,t1) $\wedge$ gt(t2,t1) $\rightarrow$ dead(x,t2)

First Order Logic (FOL) or predicate logic

Instance and isa relationship in predicate

in man(Marcus), class man is represented using unary predicate.

In instance(Marcus, man) predicate is binary such that first argument is an instance (element) of the second argument which is usually a class.

In isa(Pompeian,Roman) the subclass (Pompeian) and superclass (Romans) are used .

Isa relationships simplifies representation of wffs and combines statements to express knowledge in a shorter version.

e.g.

Dog(Phantom)

Instance(Phantom,dog)

Isa(dog, creatures)

Try more such sentences

First Order Logic (FOL) or predicate logic

Resolution

Resolution is used to produce proofs for facts represented in sentence forms. Resolution uses following two steps

a) Conversion of axioms to canonical or clause form
b) Using refutation which means to show that the negation of the statement produces a contradiction with the known statement (which is a fact)

Conversion from Conjuctive Norm Form CNF to clause form

Suppose we want to convert following wff to clause form (all Romans who know Marcus either hate Caesar or think that anyone who anyone is crazy)

$\forall x$:[Roman(x) $\wedge$ know(x,Marcus)] $\rightarrow$ [hate(x,Caesar)$\vee$($\forall y$: $\exists z$: hate(y,z) $\rightarrow$ thinkcrazy(x,y))]

1. In the process of clause form conversion, we will take one by one all rules.

Resolution Conversion to clause form

∀x:[Roman(x) ∧ know(x,Marcus)] → [hate(x,Caesar)∨(∀y: ∃z: hate(y,z) → thinkcrazy(x,y))]

1. Eliminate → (e.g. A → B can be replaced by ¬A ∨B)

∀x: ¬[Roman(x)∧know(x,Marcus)] ∨ [hate(x,Caesar)∨(∀y:¬(∃z: hate(y,z) ∨ thinkcrazy(x,y))]

Use property of negation : ¬(¬p) = p;

Apply deMorgan's laws as follows

¬(a ∧ b) = ¬a ∨ ¬b; ¬(a ∨ b) = ¬a ∧ ¬b

And the quantifiers negations as follows

¬ ∀x: p(x) is same as ∃x: ¬ p(x); and ¬∃x: p(x) same as ∀x: ¬ p(x)

∀x: [¬Roman(x)∨¬know(x,Marcus)] ∨ [hate(x,Caesar)∨(∀y: :∀z: ¬ hate(y,z) ∨ thinkcrazy(x,y))]

Allow each quantifier to bind unique variable (as variables are dummy names)

∀x :p(x) ∨ ∀x:q(x) can be converted to ∀x:p(x) ∨ ∀y: q(y)

Bring all quantifiers to the left of wff in relative order.

∀x: ∀y: ∀z: [¬Roman(x)∨¬know(x,Marcus)] ∨ [hate(x,Caesar)∨(¬hate(y,z) ∨ thinkcrazy(x,y))] This form is prenex normal form, a form in which we have prefix of quantifiers followed by a matrix (rectangular bracket, free of quantifiers]

Resolution Conversion to clause form

Skolem Functions

Existential quantifiers $\exists$ can be eliminated by assuming that rather than thinking one anonymous value to satisfy $\exists(y)$, we can have one function to replace $\exists(y)$. As an example $\exists(y)$: president(y) means for some value of y, President(y) holds true. We can transform this to President(S) to represent specifically S as President. Here S is a Skolem function without argument. Further, example: *everyone has a father* is formulated as, $\forall x: \exists y:$ father-of(y,x) can be written in Skolem function as $\forall x:$ father-of(S(x),x)) to relate S(x) as father of each x.

Everyone loves someone: $\forall x: \exists y:$ lovedby(y,x) in Skolem function as $\forall x:$lovedby(S(x),x))

Continuing our original sentence,

At this point all variables are universally quantified, drop each such quantifier from wff

$[\neg Roman(x) \vee \neg know(x,Marcus)] \vee [hate(x,Caesar) \vee (\neg hate(y,z) \vee thinkcrazy(x,y))]$

Apply associate law $a \vee (b \vee c) = (a \vee b) \vee c$ to get

$\neg Roman(x) \vee \neg know(x,Marcus) \vee hate(x,Caesar) \vee \neg hate(y,z) \vee thinkcrazy(x,y)$

Distributive property is given as follows (although not required in our example)

$(a \wedge b) \vee c = (a \vee c) \wedge (b \vee c)$

Exercises with simple, short sentences to be tried by students.

Unification Algorithm

We know that dog(Boxer) and ¬ dog(Boxer) is a contradiction as both can not be true at the same time. However dog(Boxer) and ¬dog(Jackie) is not a contradiction. To check a contradiction, there must be some procedure to match literals and possibility to make them identical. This recursive procedure is called unification algorithm.

We know that

classmates(Ram, Ramesh) and beats(Ram,Ramesh) can not be unified as both have different initial predicate symbols (classmates and beats which differ).If both predicate symbols match then only we can use unification procedure.

Simple examples

Unify Q(x) and P(x) → FAIL as literals are different and can not be unified

Unify Q(x) and Q(x) → Nil as literals are identical so no scope of unification

Unify P(x) and P(x,y) → FAIL as both literals have different number of arguments

Simple examples

Unify P(x,x) and P(y,z)

Here both initial predicate symbols are identical, P so we check number of arguments, which is also same

Now substitute y/x to get P(y,y) and P(y,z) then take z/y which produces P(z,z) thus (z/y)(y/) is the total substitution applied to unify the two literals

Avoid substitution like (x/y)(x/z) as they cause inconsistency

Unification Algorithm

```
Unify(L1,L2) // unifies two literals L1 and L2
1. If L1 or L2 is a variable or constant, then:
       a) If L1 and L2 are identical, then return NIL.
       b) Else if L1 is a variable, then if L1 occurs in L2 then return FAIL, else return {(L2/L1)}.
       c) Else if L2 is a variable, then if L2 occurs in L1 then return FAIL, else return {(L1/L2)}.
       d) Else return FAIL.

2. If the initial predicate symbols in L1 and L2 are not identical, then return FAIL.

3. If L1 and L2 have a different number of arguments, then return FAIL
4. Set SUBST to NIL.

5. For i ← 1 to number of arguments in L1:
       a) Call Unify with the ith argument of L1 and the ith argument of L2, putting result in S.
       b) If S = FAIL then return FAIL.
       c) If S is not equal to NIL then:
              i. Apply S to the remainder of both L1 and L2.
              ii. SUBST := APPEND(S, SUBST).

6. Return SUBST.
```

As another example hate(x,y) and hate (Marcus,z) can be unified using (Marcus/x,z/y) or (Marcus/x,y/z)

Resolution in propositional logic

Suppose a set of axioms(propositions) is given. Convert all propositions of this set to clause form. The resolution algorithm is given as follows

Algorithm for propositional resolution

1.Convert all propositions to clause form.

2.Negate P and convert the result to clause form. Add it to the set of clauses.

3.Repeat until either a contradiction is found or no progress is possible.

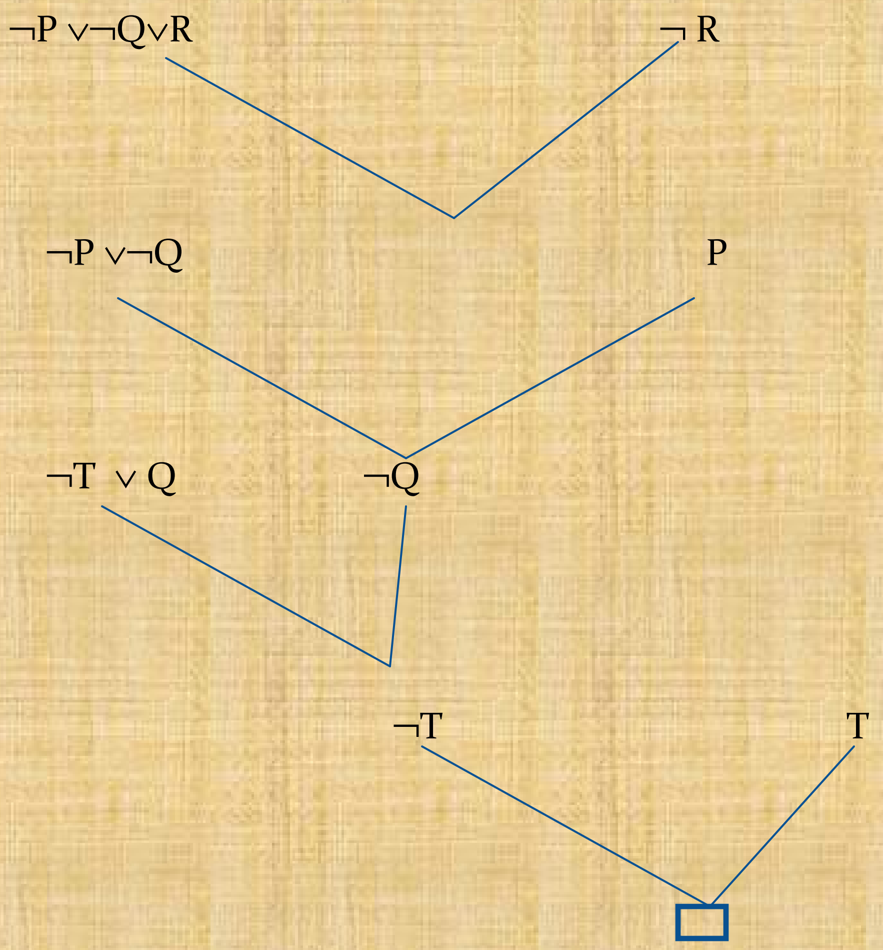
(a) Take any two clauses as parent clauses.

(b) Resolve these two clauses. The resulting clause is called resolvent.

(c) if the resolvent is empty clause then a contradiction is found. If not, then add resolvent to the set of clauses. Suppose following propositions are given and we have to prove R.

Given axioms (propositions)	Clause form	No.
P	P	(1)
$(P\wedge Q) \rightarrow R$	$\neg P \vee\neg Q\vee R$	(2)
$(S \vee T) \rightarrow Q$	$\neg S \vee Q$	(3)
Separate (3) in CF	$\neg T \vee Q$	(4)
T	T	(5)

Resolution in propositional logic (To prove R, start with ¬ R)



$\square$ means that a contradiction is reached, we started with ¬ R and combined all true propositions , thus our initial assumption was wrong and R is true

Resolution Proofs in Predicate Logic

In order to prove a fact following algorithm of resolution is applied

Resolution Algorithm (To prove P)

❖Convert all the propositions (axioms) of *F* to clause form.

❖Negate *P* and convert the result to clause form. Add it to the set of clauses obtained in 1.

❖Repeat until either a contradiction is found, no progress can be made, or a predetermined amount of effort has been expended.

a) Select two clauses. Call these the parent clauses.

b) Resolve them together. The resolvent will be the disjunction of all the literals of both parent clauses with appropriate substitutions performed and with the following exception: If there is one pair of literals T_1 and $\neg T_2$ such that one of the parent clauses contains T_1 and the other contains $\neg T_2$ and if T_1 and T_2 are unifiable, then neither T_1 nor $\neg T_2$ should appear in the resolvent. If there is more than one pair of complementary literals, only one pair should be omitted from the resolvent.

c) If the resolvent is the empty clause, then a contradiction has been found. If it is not, then add it to the set of clauses available to the procedure.

Resolution Proofs in Predicate Logic

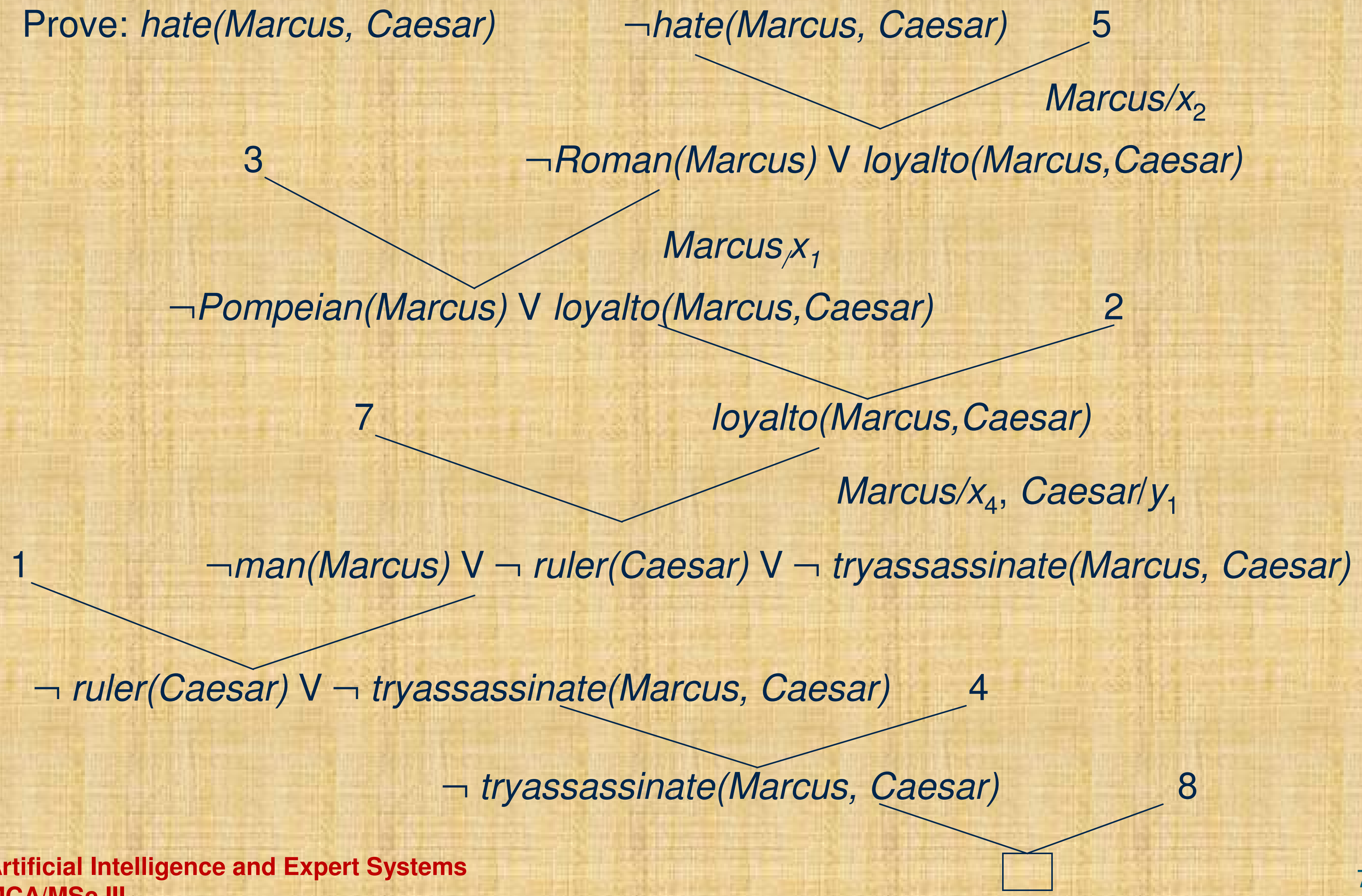
A simple example to prove that Marcus hated Caesar

Suppose following axioms in clause form are given

1. *man(Marcus)*
2. *Pompeian(Marcus)*
3. $\neg$ *Pompeian*(x_1) v Roman(x_1)
4. *Ruler*(*Caesar*)
5. $\neg$ *Roman*(x_2) v *loyalto*(x_2, *Caesar*) v *hate*(x_2, *Caesar*)
6. *loyalto*(x_3, *fi*(x_3))
7. $\neg$ *man*(x_4) v $\neg$ *ruler*(y_1) v $\neg$ *tryassassinate*(x_4, y_1) v $\neg$ *loyalto* (x_4, y_1)
8. *tryassassinate*(*Marcus*, *Caesar*)

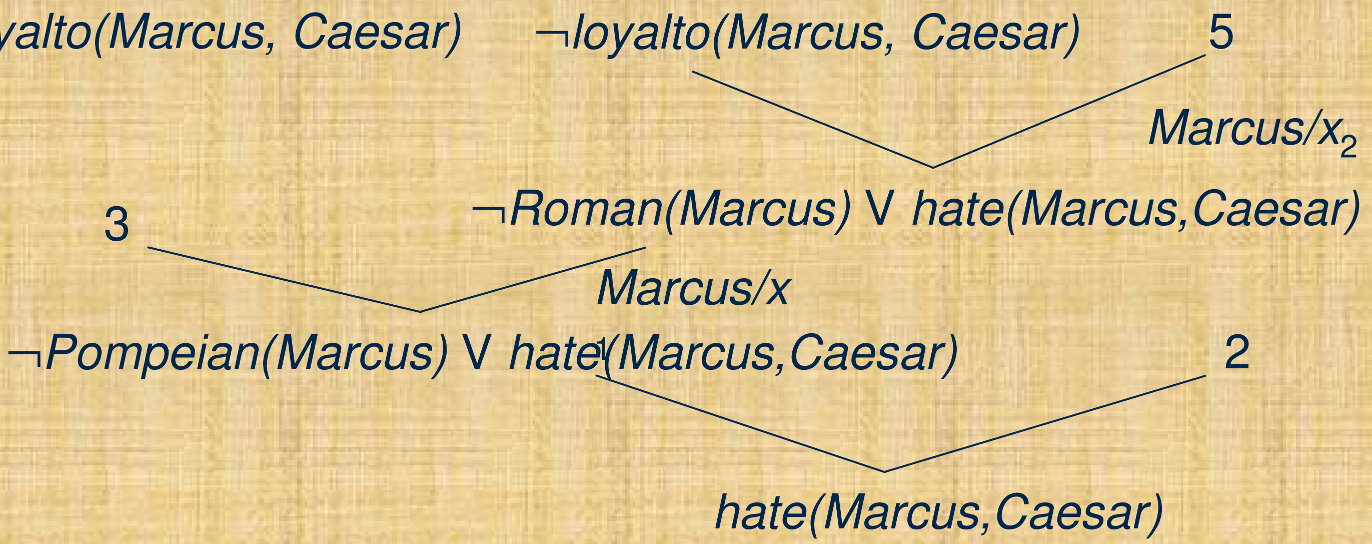
Variables in 3,5,6,7, (x_1, x_2, x_3, x_4 *y)* have been used to discriminate from each other

Prove: *hate(Marcus, Caesar)*

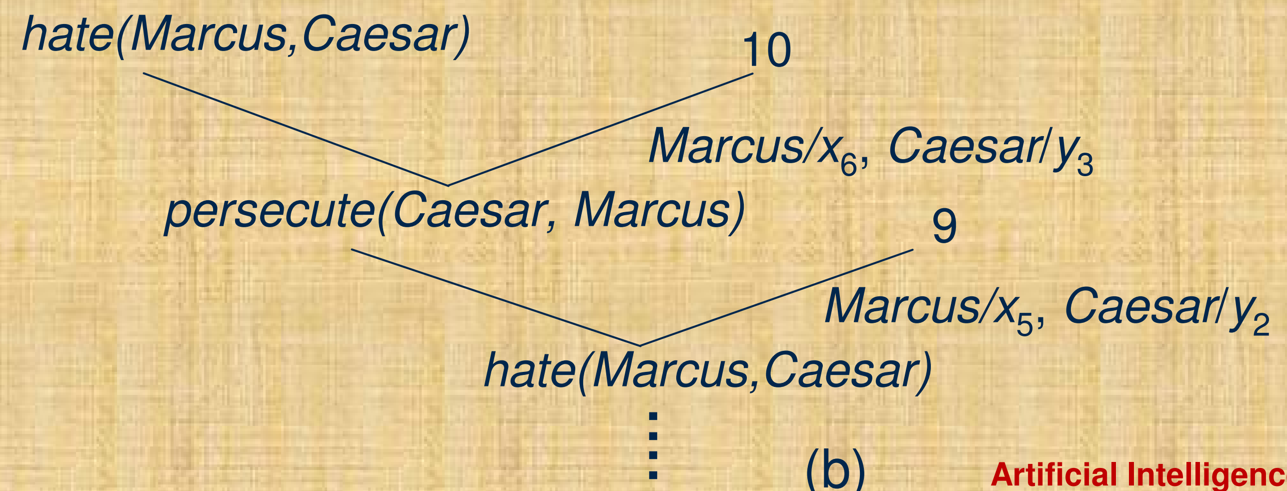

¬hate(Marcus, Caesar)
5
Marcus/x_2
3
¬Roman(Marcus) ∨ loyalto(Marcus,Caesar)
Marcus/x_1
¬Pompeian(Marcus) ∨ loyalto(Marcus,Caesar)
2
7
loyalto(Marcus,Caesar)
Marcus/x_4, Caesar/y_1
1
¬man(Marcus) ∨ ¬ ruler(Caesar) ∨ ¬ tryassassinate(Marcus, Caesar)
¬ ruler(Caesar) ∨ ¬ tryassassinate(Marcus, Caesar)
4
¬ tryassassinate(Marcus, Caesar)
8

Unsuccessful Attempts at Resolution to prove that Marcus is loyal to Caesar

Prove: *loyalto(Marcus, Caesar)*


$\neg loyalto(Marcus, Caesar)$
5
$Marcus/x_2$
$\neg Roman(Marcus) \lor hate(Marcus, Caesar)$
3
$Marcus/x_1$
$\neg Pompeian(Marcus) \lor hate(Marcus, Caesar)$
2
$hate(Marcus, Caesar)$

(a)



(b)

Unit-3 Knowledge Representation: Semantic Net(works)

There are four common approaches by which knowledge can be achieved

1.Simple relational knowledge: Using relational databases / tables, various queries can be answered. E.g

RN	Name	Age	height	Weight
1	XAR	11	4.5	75
2	BAS	27	6	75.2

It is easy to answer the queries like what is the age of XAR etc. But who is heaviest? (answer)

2. Inheritable knowledge : Here knowledge is derived from the inheritance properties through attributes (features) and associated values.

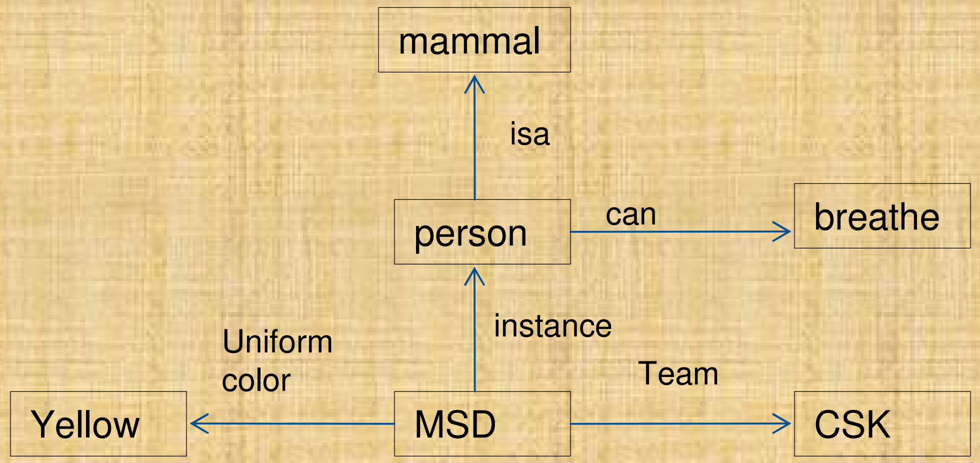
3. Inferential Knowledge: Knowledge comes from predicate wffs like

$$\forall x: \text{man}(x) \rightarrow \text{mortal}(x)$$

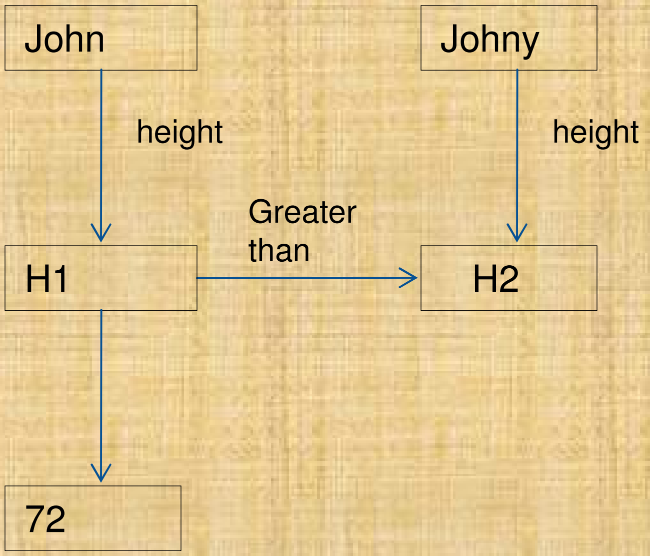
4. Procedural Knowledge : Using basic facts and small programs, knowledge can be expressed. E.g if Den is a bird then we can conclude Den can fly but it can turn out to be wrong if Den is an ostrich or a any bird which doesn't fly. However in most cases, it is true.

Unit-3 Knowledge Representation: Semantic Net(works)

Semantic Net: A semantic network is often used as a form of knowledge representation by connecting concepts together. It is a directed graph consisting of vertices which represent concepts and edges which represent semantic relations between the concepts. E.g see the following semantic net to answer the query 'can CPD breathe'? 'What is the uniform color of CPC'?



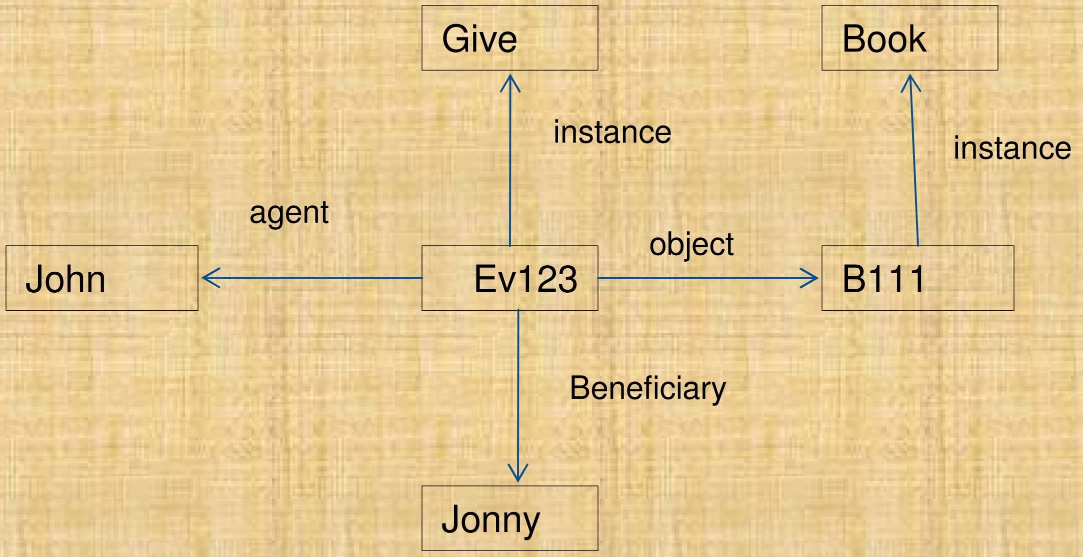
Semantic Net: Following sentences are converted to semantic nets


$$\begin{tikzcd}
\boxed{\text{John}} \arrow[d, "\text{height}"] & \boxed{\text{Johny}} \arrow[d, "\text{height}"] \\
\boxed{\text{H1}} \arrow[r, "\text{Greater than}"] \arrow[d] & \boxed{\text{H2}} \\
\boxed{72} &
\end{tikzcd}$$

(John is 72" tall and taller than Johnny)

Unit-3 Knowledge Representation: Semantic Net(works)

Semantic Net: Following sentences are converted to semantic nets



(John gave the book to Johnny)

Unit-3 Knowledge Representation: Partitioned Semantic Net(works)

In many cases, the sentences may not be as straight and simple as seen in previous examples. Especially when the scope of certain literals of the sentence is quantified by $\forall$ (for all) and $\exists$ (at least one). Under these situations, the sentences may be partly broken into segments such that each segment has a specific scope. Such networks where, the scope of the variables in a sentence is separately shown, are called partitioned semantic networks.

In the next four examples, following sentences are expressed as semantic nets

a)The dog bit the mail-carrier

b)Every dog has bitten a mail carrier

c)Every dog has bitten the constable

d)Every dog has bitten every mail carrier

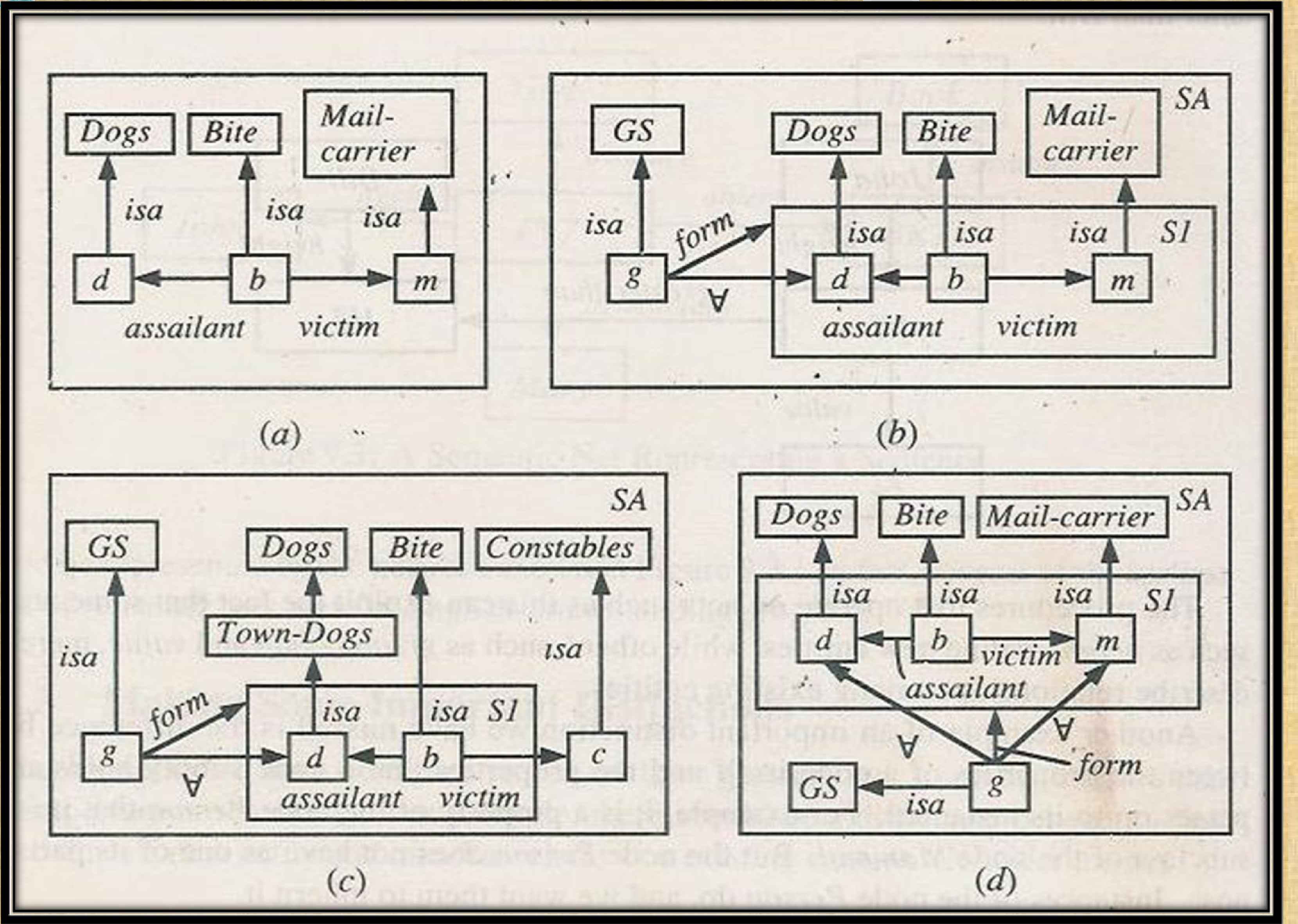
In (a), there is not more than one dog/mail carrier affected. So no need of partitioning, in (b), every means $\forall$ (thus $\forall$d: $\exists$ m:dog(d) $\wedge$ m(mail carrier $\rightarrow$ bit(d,m))

In (c), every dog bits the same constable, so sentence has a generic scope for dogs and not for constable (i.e. $\forall$d: dog(d) $\rightarrow$ bit(d, constable))

In (d), scope of the sentence is for dogs as well as for mail carrier so $\forall$quantifier is connected to both, dogs and mail carriers ($\forall$d: $\forall$ m:dog(d) $\wedge$ m(mail carrier $\rightarrow$ bit(d,m))

The sentences can have a form to envelop the scope of it and secondly the variables which are attached to quantifiers

Unit-3 Knowledge Representation: Partitioned Semantic Net(works)



Unit-4

Pattern Recognition

Unit-4 Pattern Recognition

Pattern: refers to an object, article, machine, face, living thing, character etc. In our context, pattern refers to those items which are related to humans in some way.

Pattern Recognition: It is the study of making a machine to recognize patterns. The examples include, face recognition, fruit identification, character recognition etc. It is a broad branch of AI. The study encloses various tools for pattern recognition such as neural networks, various classifiers, fuzzy sets, evolutionary computations. We will discuss in details.

Pattern classification

Pattern classification or simply classification is the back bone of Pattern Recognition. Most of the issues in Pattern Recognition use classification's techniques.

In general classification is the process of separating objects on the basis of classes (or similarity, to be discussed later) or classifying objects into different groups such that objects belonging to a group have something similar than in other groups.

Alternatively, **pattern recognition** is the assignment of a label to a given input value.

Unit-4 Pattern Recognition:

Pattern classification: Supervised or Unsupervised

There can be two situations while classifying objects.

1. Classes of the patterns are given
2. Classes of the patterns are NOT given

While classifying the patterns, case (1) is called supervised classification while case (2) is known as unsupervised classification.

P#	F1	F2	F3	Class
1	4.5	2.3	1.9	1
2	1.8	0.9	5.6	2
3	0.55	3.12	9.1	3

1. Supervised Classification
   (class is given)

P#	F1	F2	F3
1	4	2.3	8
2	8	5.6	12
3	6	2.3	3

2 Unsupervised Classification
(class is not given)

In both cases, we have a pattern set of 3 patterns, each consisting of 3 features. The left figure (1) contains a class while right figure (2) doesn't.

Pattern classification: Algorithms

The problems related to classification includes following cases

1. Given a set of patterns, called training data. We are asked to train ourselves for this data so that later on we would be able to tell the class of any pattern. After training, we may be presented a pattern from the training data and we are asked to tell the class of this pattern. This pattern is called test pattern. As this test pattern is from the given training data, we might answer correctly and the class answered by us would be compared with that of the training data. If the classes of both patterns (i.e training and testing) match, then we are correct otherwise wrong.
2. In the above case, if the test pattern is suppose not from the training data and still we tell the class of a test pattern which is never seen before during training. Then our prediction might be correct or wrong.
3. If no class is given at all during training data patterns, but they are grouped on the basis of some similarities within groups. Suppose we are asked to place the test pattern in one of the groups of the training data, then we may do it correctly such that test pattern is placed in the genuine group.

Cases 1,2 refer to supervised methods of classification while case 3 belongs to unsupervised classification.

We also noted that training data set is one which is used to get knowledge whereas test data is used to test our knowledge. Simple example is when you learn in class, it is training, but when you take exam, no assistance is available with you, then your performance is evaluated, what we call is test (exam).

Unit-4 Pattern Recognition:

- K-Nearest Neighbor Algorithm: The alternative names of this algorithm are
- K-NN
- Memory-Based Reasoning
- Example-Based Reasoning
- Instance-Based Learning
- Case-Based Reasoning
- Lazy Learning
- It is a supervised classification algorithm also called classifier

The idea of a k-nn learning is simple. In this algorithm, k is the number of nearest neighbors. When we want to find the class of an unknown (or unseen) test pattern, then decide its class by knowing the class of a pattern which is nearest to unknown test pattern (k=1). Sometimes due to certain discrepancies, the nearest neighbor may be a different class pattern, then our approach would lead to a wrong class of test pattern. In this case it is better to take more nearest neighbors (k>1) and know their classes from training data set. Take the majority of the classes known from nearest neighbors. For a clear determination of class, take k as odd (3,5,7,..) so that there is no tie while taking the majority of classes from nearest neighbors.

Unit-4 Pattern Recognition:

- **K-Nearest Neighbor Algorithm:**
- The nearest neighbors are determined by finding the Euclidean distance between training pattern and testing pattern. Suppose we have 100 patterns in training data set. To find the class of a test pattern, find out the Euclidean distance of test pattern from all 100 training patterns. Thus we have 100 values. If we take k=1, then take the training pattern with which test pattern has minimum value (distance). The class of this pattern (which is already given in the training data) will be class of the test pattern. If k=3, then take 3 minimum values and see the classes of concerned patterns. The class of the test pattern will be the class which is given by the majority of the 3 classes.
- If P_tr and P_tt are training and test patterns respectively with same number of features f1,f2,f3. P_tr is represented by P_tr (f1,f2,f3) and P_tt by P_tt(f1t,f2t,f3t) then the Euclidean distance between these pattern is given by
- E_d =
- Remember that $\sqrt{(f_1 - f_{1t})^2 + (f_2 - f_{2t})^2 + (f_3 - f_{3t})^2}$
- (1) The class is not considered while taking Euclidean distance (because you do not know the class of the test pattern)
- (2) The class of the nearest neighbor of the test pattern is checked to decide the class of the test pattern
- ***** For numeric examples, refer to class room lectures, books, notes, internet. For any clarity contact the teacher any time in department

- **Classifiers and classification accuracy**
- For the data sets where class of each pattern is given before training and later on testing, we apply supervised classification techniques (such as k-nn) and predict the class of the unknown testing pattern by finding the class of its k- nearest neighbors. The algorithm or technique which is applied for predicting the class is known as classifier. Out of 100 testing patterns for example, if a classifier predicts the classes of 80 patterns correctly (and 20 incorrectly) then the classifier is said to have a **classification accuracy** of 80%. The performance of a classifier is normally judged by its classification accuracy. No doubt there can be other factors also like time, cost, space etc to decide the suitability of a classifier, but for now we focus on classification accuracy only.
- **Cross Validation:** Suppose we want to develop a classifier, say k-nn for example. Given a data set, to develop a robust classifier to predict the class of a test pattern, it is important that the classifier should be able to predict the class of a testing pattern taken from any part of the data set . For that, we must shuffle the patterns of the data set randomly and ensuring that the classifier is going to be trained for every class. For this reason, cross validation technique is applied. In this, the data set is divided into a whole number of sections, say 5. Then usually we take four sections for training purpose and one section for testing. We shift training and testing sections frequently so that all patterns are trained for at least once . These sections are also called sometimes as folds. Thus in this example, we have five folds, out of which four are used for training and one for testing. We can also say that we have 80% training data and 20% testing data. When the unknown pattern is tested, the classifier would have either seen it during training period or at least classifier would have trained similar kind of a pattern if not exactly the same unknown pattern. This will improve the accuracy to some extent.

Unit 4 Pattern Recognition:

- **Classifiers and classification accuracy**
- For the data sets where class of each pattern is given before training and later on testing, we apply supervised classification techniques (such as k-nn) and predict the class of the unknown testing pattern by finding the class of its k- nearest neighbors. The algorithm or technique which is applied for predicting the class is known as classifier. Out of 100 testing patterns for example, if a classifier predicts the classes of 80 patterns correctly (and 20 incorrectly) then the classifier is said to have a **classification accuracy** of 80%. The performance of a classifier is normally judged by its classification accuracy. No doubt there can be other factors also like time, cost, space etc to decide the suitability of a classifier, but for now we focus on classification accuracy only.
- **Cross Validation:** Suppose we want to develop a classifier, say k-nn for example. Given a data set, to develop a robust classifier to predict the class of a test pattern, it is important that the classifier should be able to predict the class of a testing pattern taken from any part of the data set . For that, we must shuffle the patterns of the data set randomly and ensuring that the classifier is going to be trained for every class. For this reason, cross validation technique is applied. In this, the data set is divided into a whole number of sections, say 5. Then usually we take four sections for training purpose and one section for testing. We shift training and testing sections frequently so that all patterns are trained for at least once . These sections are also called sometimes as folds. Thus in this example, we have five folds, out of which four are used for training and one for testing. We can also say that we have 80% training data and 20% testing data. When the unknown pattern is tested, the classifier would have either seen it during training period or at least classifier would have trained similar kind of a pattern if not exactly the same unknown pattern. This will improve the accuracy to some extent.

Unit 4 Pattern Recognition:

- **Unsupervised Classification and Clustering**
- We saw, how, the class of an unknown pattern can be determined using the help of exiting patterns labeled with class. But there are situations when the class of none of the patterns is given. For examples, different objects are given to a child (which we have already done at KG or nursery class level), and he is asked to separate the objects into groups. The child does not know the names of those objects, neither there is any label attached to any object. The child will see the resemblance of an object will try to put it with another object which is quite similar to the former. In this manner, he can form some groups and in each group all objects are similar. Moreover an object in a group has maximum similarity with any other object of that group and more dissimilarity with any object of other groups. These groups are called clusters. Whenever the child is presented a new object, he will place that new object into an existing group where this new object has maximum similarity. This type of classification is called unsupervised classification. It is commonly known as clustering.
- As another example, if hundreds of students are enjoying the musical concert in an auditorium. There can be traditionally two clusters formed, in one cluster, only boys and in another cluster, the girls only. There can be boys of different classes in the cluster of boys (may be MCA, MSc, MA, Ph.D, BA etc), similarly there will be girls of different classes in the same cluster. We see that there can be many classes belonging to same cluster but it is meaning less or not known (as it will be difficult to find out the class of each student in dark). Contrarily there can be many clusters in a single class. E.g., in an MSc class, there can be two clusters, boys and girls. Or may be three clusters, one cluster having students wearing sandals, second cluster's students wearing shoes and the third cluster's students wearing slippers.

Unsupervised Classification and Clustering

- We saw, how, the class of an unknown pattern can be determined using the help of exiting patterns labeled with class. But there are situations when the class of none of the patterns is given. For examples, different objects are given to a child (which we have already done at KG or nursery class level), and he is asked to separate the objects into groups. The child does not know the names of those objects, neither there is any label attached to any object. The child will see the resemblance of an object will try to put it with another object which is quite similar to the former. In this manner, he can form some groups and in each group all objects are similar. Moreover an object in a group has maximum similarity with any other object of that group and more dissimilarity with any object of other groups. These groups are called clusters. Whenever the child is presented a new object, he will place that new object into an existing group where this new object has maximum similarity. This type of classification is called unsupervised classification. It is commonly known as clustering.
- As another example, if hundreds of students are enjoying the musical concert in an auditorium. There can be traditionally two clusters formed, in one cluster, only boys and in another cluster, the girls only. There can be boys of different classes in the cluster of boys (may be MCA, MSc, MA, Ph.D, BA etc), similarly there will be girls of different classes in the same cluster. We see that there can be many classes belonging to same cluster but it is meaning less or not known (as it will be difficult to find out the class of each student in dark). Contrarily there can be many clusters in a single class. E.g., in an MSc class, there can be two clusters, boys and girls. Or may be three clusters, one cluster having students wearing sandals, second cluster's students wearing shoes and the third cluster's students wearing slippers.

K-means Clustering

Pseudo code (Simple implementation)

1. Decide the number of clusters k
2. Initialize k- number of centroids (or can take any k number of patterns)
3. Find out the Euclidean distance of every pattern from each of these k-centroids. Thus we have k-distances of a pattern from these k-centroids.
4. A pattern will belong to a cluster with which its distance is minimum. In case of a tie, a pattern can reside in any one of the tied up clusters.
5. Find out the belongingness of each pattern. Thus all k-clusters will have some number of patterns.
6. Now modify each of the k-centroids by taking the means of the coordinates (feature values) of the patterns of a cluster.
7. If the modified centroid is same as previous centroid, stop, otherwise repeat steps 3-6.
8. Note the patterns belonging to each cluster.

Unit-5

Expert Systems

Unit-5: Expert Systems

Meaning: An expert system is a complex AI program which is used to solve problems which are solved by human expert. E.g. what an expert physician doctor does, same purpose must be achieved by an expert system.

In a conventional programming environment we see Algorithm + data structure = A computer program Whereas in an expert system Inference engine + knowledge = expert system In order to solve expert level problems, an expert system must have an access to huge knowledge base acquired from human experts. First expert system was DENDRAL, developed in 1970s at Stanford University

Advantages

Availability: ES are always available to users whenever called. They are un-affected by human like factors such as fatigue, emotional , unwillingness; can be used repetitively

Knowledge base to update any time and use: The rules added to knowledge base of an ES can be effectively used any time and updated whenever some rule is to be revised

Reliability and consistency: ES can offer reliable and trustworthy decisions. If the rules are clear and not ambiguous, ES can give clear advices. Moreover, whatever decision an ES gives to one user, it is given to all.

Pedagogy: The ES can deliver the logic or knowledge to its user why and how a decision has been taken. The user can learn from these explanations.

Preservation and updation of knowledge any time: The ES use the knowledge of human expert even after the death of the human expert. Moreover the knowledge of a particular human expert can be improved in the expert system if required.

Disadvantages or limitations: The ES are very difficult to understand by common human. Knowledge collection is difficult as well as costly. Knowledge is collected mostly manually so errors are likely. ES are expensive to develop. ES may still not be taken reliably on serious or sensitive issues like health, critical policies. ES may lead to legal or ethical issues. ES are difficult to maintain.

Despite their so many claims, ES have not been able to get a universal good acceptability in society.

Some of the areas of applications

ES can be used for following purposes

1. Interactive or conversational applications: Chatterbot
2. Fault or medical diagnosis: **Dxplain, CDSS**(Clinical Decision Support System)
3. Educational software
4. Knowledge management
5. Decision support for engineering, process control related areas
6. Accounting, loan, credit
7. Health care, hospital management, estate management
8. MYCIN, DENDRAL, CADUCEUS
9. PROSPECTOR, DESIGN ADVISOR

Components of an Expert System

ES can have following components, these may differ in different text/literature

1. Knowledge base
2. Inference Engines
3. Rule base mostly having if-then else rules
4. Fuzzy logic, neural networks, evolutionary algorithms, other tools to handle uncertainties and soft decisions.
5. Backward or forward chaining

Participants of Expert System

- Domain Experts
- Knowledge users
- Knowledge engineers

Expert System Shells

The early expert systems were built using LISP (LISt Processing) language. Later on to facilitate building of more and more ES, it was thought to form a platform which would contain most of the common procedures that an ES would require. Keeping this view in mind, shells were developed. An expert system shell is a software which contains user interface, a format for declarative knowledge in the knowledge base and an inference engine. As an example, EMYCIN was derived from MYCIN. Usually knowledge engineer will use these shells to develop an ES.

Knowledge Acquisition: A process which allows the experts to enter their knowledge or expertise into the expert system, and to refine it later as and when required. The knowledge acquisition process can usually comprise of three principal stages:

1. *Knowledge elicitation is the interaction between the expert and the knowledge* engineer/program to elicit the expert knowledge in some systematic way.
2. The knowledge thus obtained is usually stored in some form of human friendly *intermediate representation.*
3. The intermediate representation of the knowledge is then compiled into an *executable form (e.g. production rules) that the inference engine can process*

Final Words

Most part of the syllabi is covered in this presentation. At some places, highlights are given, whereas details are given in many places. The examples have been presented in class rooms. Moreover a session on LISP is also devoted in class rooms. The students can however contact me for further readings, handouts or any other difficulty regarding the course material and topics.

As the presentation will be used by students regularly and discussed frequently, sometimes, if required, the contents might be replaced as an improvement to existing material. The students are therefore advised to see the presentation regularly and/or meet me in the office for any such improvement.

Best of Luck for semester examinations!

Acknowledgements to known/unknown internet sites/readings/literature used for complex figures /symbols/explanations to make presentation more useful and simple for students.