BIT 2319 ARTIFICIAL INTELLIGENCE

Oyier Philip Apodo

January 22, 2012

Contents

1 COURSE OVERVIEW

1.1 Course Purpose

This module introduces students to the basics of Artificial Intelligence (AI). The objective is to give you an insightful introduction to AI. You will learn about the basic techniques, problem areas, major methods and current applications. Each student will participate through doing homework assignments, taking online quizzes, participate in online discussions with other students and ask questions of the facilitator.

1.2 Course Description

Fundamentals and types of KBS. Reasoning and knowledge engineering. Proposition and predicate logic. Strategies for space search such as data and goal driven, and heuristics. inference in logic, belief networks, decision theory, fuzzy logic and evolutionary techniques, neural networks, robotics.

1.3 Course Objectives

At the end of the module, the students should be able to:

- Understand how mature and commonly used AI technologies e.g. knowledge representation, reasoning, search, and related topics, can be applied to solve practical problems
- Appreciate the problems, major methods, current situation, and so on, without going into technical details.
- Use the Prolog language for various tasks.

1.4 Delivery Methodology

Online lectures, tutorials, online interactive sessions and guided self study.

1.5 Prerequisites

ICS 2405 Knowledge Based Systems

1.6 Instructional Materials

Mobile phones, tablets, laptops, computers, Prolog software.

1.7 Course Evaluation

Online quizzes, online cats (2), online assignments(2), online participation 20%
Written cats (10%) and a final Examination 70%

1.8 References

1. Artificial Intelligence: A modern Approach, 2nd Edition- Stuart Russell & Peter Norvig

2. Introduction to AI and Expert Systems: Dan W. Patterson, Prentice Hall

3. Prolog Programming for AI, 3rd Edition-Ivan Bratko, Pearson Education

2 ARTIFICIAL INTELLIGENCE INTRODUCTION

2.1 Learning Outcomes

Upon successful completion of this lesson, the student will be able to:

- Describe the differences between definitions of AI based on human-like behavior and rationality;
- Explain how artificial intelligence differs from human intelligence
- Describe the contributions and application of AI.
- Characterize the goals of AI, approaches to and progress toward those goals.

2.2 Intelligence: Dictionary definition

1. The ability to comprehend; to understand and profit from experience.
2. Capacity of mind, especially to understand principles, truths, facts or meanings, acquire knowledge, and apply it to practice.
3. The ability to apply knowledge to manipulate one's environment or to think abstractly as measured by objective criteria (as tests).

Intelligence is an umbrella term describing a property of the mind comprehending related abilities, such as the capacities for abstract thought, reasoning, planning and problem solving, the use of language, and to learn.

2.3 Types of intelligence (Multiple Intelligence Theory, Howard Gardner)

The theory of multiple intelligences was proposed by Howard Gardner in 1983 to explore and articulate various forms or expressions of intelligence available to cognition

1. **General intelligence**:
   (a) Abilities that allow us to be flexible and adaptive thinkers, not necessarily tied to acquired knowledge.
2. **Linguistic / Verbal intelligence:**
   (a) Use words and language in various forms
   (b) Ability to manipulate language to express oneself poetically
   (c) Careers that suit those with this intelligence include writers, lawyers, policemen, philosophers, journalists, politicians, poets, and teachers

3. **Logical / Mathematical intelligence:**

   (a) This area has to do with logic, abstractions, reasoning, and numbers

   (b) It correlates strongly with traditional concepts of "intelligence" or IQ.

   (c) Careers which suit those with this intelligence include scientists, physicists, mathematicians, logicians, engineers, doctors, economists and philosophers

4. **Musical intelligence**:

   (a) Recognize nonverbal sounds: pitch, rhythm, tonal patterns and music

   (b) Careers that suit those with this intelligence include instrumentalists, singers, conductors, disc-jockeys, orators, writers and composers.

5. **Spatial intelligence:**

   (a) Typically thinks in images and pictures, used in both arts and sciences

   (b) Careers which suit those with this type of intelligence include artists, designers and architects.

   (c) A spatial person is also good with puzzles.

6. **Intrapersonal intelligence:** -

   (a) Ability to understand oneself, including feelings and motivations / Can discipline themselves to accomplish a wide variety of tasks

   (b) People with intrapersonal intelligence are intuitive and typically introverted.

   (c) They are skillful at deciphering their own feelings and motivations.

7. **Interpersonal intelligence:** -

   (a) Ability to "read people"; discriminate among other individuals especially their moods, intentions, motivations;

   (b) In theory, people who have a high interpersonal intelligence tend to be extroverts, characterized by their sensitivity to others' moods, feelings, temperaments and motivations, and their ability to cooperate in order to work as part of a group.

   (c) Careers that suit those with this intelligence include sales, politicians, bankers, teachers, and social workers

8. **Naturalist intelligence: -**

   (a) Ability to recognize and classify living things like plants, animals

(b) This area has to do with nature, nurturing and relating information to one's natural surroundings.

(c) Careers which suit those with this intelligence include naturalists, farmers and gardeners.

9. **Bodily / Kinesthetic intelligence: -**

(a) Use one's mental abilities to coordinate one's own bodily movements

(b) Careers that suit those with this intelligence include: athletes, dancers, no common lactors, surgeons, doctors, builders, police officers, and soldiers.

Note: Understanding the various types of intelligence provides theoretical foundations for recognizing different talents and abilities in people.

2.4 The Nature of Intelligence

Characteristics of intelligent behavior include the ability to:

- Learn from experience and apply knowledge acquired from experience
- Handle complex situations
- Solve problems when important information is missing
- Determine what is important
- React quickly and correctly to a new situation

2.4.1 What's involved in Intelligence?

Ability to interact with the real world

- to perceive, understand, and act
  - e.g., speech recognition and understanding and synthesis
  - e.g., image understanding
  - e.g., ability to take actions, have an effect

Reasoning and Planning

- modeling the external world, given input
- solving new problems, planning, and making decisions
- ability to deal with unexpected problems, uncertainties

Learning and Adaptation

- we are continuously learning and adapting
- our internal models are always being "updated"
  - e.g., a baby learning to categorize and recognize animals

2.5 Natural and Artificial Intelligence

Ability to	Natural intelligence(Human)		AI(Machine)	
	Low	High	Low	High
Use sensors (eyes,ears,touch,smell)	Yes	No	√	
Be creative and imaginative		√	√	
Learn from experience		√	√	
Adapt to new situations		√	√	
Afford the cost of acquiring intelligence		√	√	
Acquire a large amount of external information		√		√
Use a variety of information sources		√		√
Make complex calculations		√		√
Transfer information	√			√
Make a series of calculations rapidly and adequately	√			√

2.6 Defining Artificial Intelligence (A I)

There is no agreed definition of the term artificial intelligence. However, there are various definitions that have been proposed. These are considered below:-

- AI is a study in which computer systems are made that think like human beings. Haugeland, 1985 & Bellman, 1978.
- AI is the study of computations that make it possible to perceive, reason and act. Winston, 1992
- AI is considered to be a study that seeks to explain and emulate intelligent behavior in terms of computational processes. Schalkeoff, 1990.
- AI is considered to be a branch of computer science that is concerned with the automation of intelligent behavior. Luger & Stubblefield, 1993

2.6.1 Artificial Intelligence Systems

The people, procedures, hardware, software, data, and knowledge needed to develop computer systems and machines that demonstrate the characteristics of intelligence in human behavior: perception, natural language processing, reasoning, planning and problem solving, learning and adaptation, etc.

2.6.2 Emphasized aspects

- Cognitive approaches
  - Emphasis on the way systems work or "think"
  - Requires insight into the internal representations and processes of the system
- Behavioral approaches
  - Only activities observed from the outside are taken into account
- Human-like systems
  - Try to emulate human intelligence
- Rational systems
  - Systems that do the "right thing"
  - Idealized concept of intelligence

2.6.3 Goals of Artificial Intelligence (AI):

Its scientific goal is to understand intelligence by building computer programs that exhibit intelligent behavior. It is concerned with:

- The concepts and methods of symbolic inference, or reasoning, by a computer, and
- How the knowledge used to make those inferences will be represented inside the machine

2.7 Foundations of Artificial Intelligence /Academic Disciplines relevant to AI

Many disciplines contribute to goal of creating/modeling intelligent entities: Philosophy

- Logic, methods of reasoning, mind as physical system, foundations of learning, language, rationality, nature of belief, theories of language

Mathematics

- Formal representation and proof, algorithms, computation, logic, probability

Probability/Statistics

- Modeling uncertainty, learning from data

Economics

- Utility, decision theory, rational economic agents

Neuroscience/ Human Biology (how brain works)

- Neurons as information processing units.

Psychology / Cognitive Science

- How do people behave, perceive, process cognitive information, represent knowledge, reason.

Computer engineering

- Building fast computers, tools for testing theories, programmability, speed, storage, actions Control theory

Linguistics

- Knowledge representation, grammar, structure and meaning of language, natural language processing

Note: Subject draws on ideas from each discipline

2.7.1 History of Artificial Intelligence

- En43 McCulloch & Pitts: Boolean circuit model of brain
- 1950 Turing's "Computing Machinery and Intelligence"
- 1950s Early AI programs, including Samuel's checkers (draughts) program
- Newell & Simon's Logic Theorist, Gelernter's Geometry Engine
- 1956 Dartmouth meeting: "Artificial Intelligence" adopted
- 1966–74 AI discovers computational complexity, Neural network research almost disappears
- 1969–79 Early development of knowledge-based systems
- 1980–88 Expert systems industry booms
- 1988–93 Expert systems industry busts: "AI Winter"
- 1985–95 Neural networks return to popularity (Return of ANN)
- 1988– Resurgence of probabilistic and decision-theoretic methods Rapid increase in technical depth of mainstream AI, "Nouvelle AI": ALife, GAs, soft computing

2.8 Main branches of AI

1. **Machine vision:** This is area dealing with visual recognition of objects i.e. capture, store, and manipulate visual images and pictures.

2. **Speech synthesis and recognition:** This is an area that attempt to find how to make computers recognize voice inputs and respond vocally.

3. **Machine learning:** The investigation is focused on making computers acquire knowledge, skills and be adaptive.

4. **Robotics:** Mechanical and computer devices that perform tedious tasks with high precision. The investigation is focused on movement and positioning of arms and other parts.

5. **Natural language and understanding:** Computers understand and react to statements and commands made in a "natural" language, such as English. Here investigations consider grammars and semantics of languages.

6. **Problem solving:** Concepts and methods for building programs that reason about problems rather than calculate a solution.

7. **Pattern recognition:** When a program makes observations of some kind, it is often programmed to compare what it sees with a pattern.

8. **Planning programs:** Start with general facts about the world (especially facts about the effects of actions), facts about the particular situation and a statement of a goal. From these, they generate a strategy for achieving the goal. In the most common cases, the strategy is just a sequence of actions.

9. **Ontology:** The study of the kinds of things that exist.

10. **Neural network:** Computer system that can act like or simulate the functioning of the human brain

11. **Expert Systems:** Hardware and software that stores knowledge and makes inferences, similar to a human expert

2.9 Some Applications of Artificial Intelligence

There are currently many applications of artificial intelligence including: Military, medicine, industry, business, education, agriculture, engineering, entertainment etc.

2.9.1 Some Real Accomplishments of AI/ Success Stories

- DARPA Grand Challenge – 123 miles through the desert
- Deep Space 1 – Remote Agent Experiment
- Deep Blue defeated the reigning world chess champion Garry Kasparov in 1997
- Proved a mathematical conjecture (Robbins conjecture) unsolved for decades
- Logistics Planning for 1991 Gulf War
- Computer Algebra Systems
- Credit Evaluation
- Fraud Detection
- NASA's on-board autonomous planning program controlled the scheduling of operations for a spacecraft
- Proverb solves crossword puzzles better than most humans
- Robot driving: DARPA grand challenge 2003-2007
- 2006: Face recognition software available in consumer cameras

2.9.2 Approaches to AI

There are two primary approaches to AI are:

- symbol processing
- artificial neural networks

These two approaches seem to be complementary.

2.9.3 The Symbol Processing Approach

This course focuses on the symbol processing approach to AI, which has two major components:

1. The study of how information can be stored as patterns/structures of symbols. (Knowledge representation)
2. The study of the processes by which new information can be generated from old information. (Inference/reasoning)

2.9.4 The Standard Architecture

The standard AI system architecture consists of:

1. A knowledge base or database consisting of (representations of) the knowledge/beliefs of the system.
2. An inference engine which applies rules of inference to the knowledge base in response to queries or new information.
3. An interface between the system and the world through which the system receives new information and queries, and returns answers and other responses.

2.10 Acting Humanly: Turing Test For Intelligence

Proposed by Alan Turing in 1950 to provide an operational definition of intelligent behavior. Turing defined intelligent behavior as the ability to achieve human-level performance in all cognitive tasks, sufficient to fool an interrogator

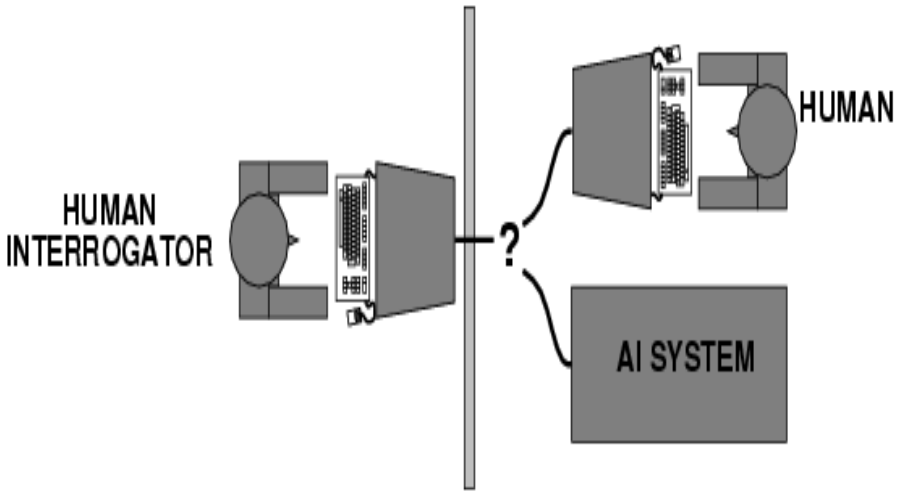


Figure 1: Turing Test

- The computer is interrogated by a human via a teletype
- It passes the test if the interrogator cannot identify the answerer as computer or human
- Suggested major components required for AI:
  - knowledge representation
  - reasoning,
  - language/image understanding,
  - learning

- To pass Turing Test, the computer must:
  - Process natural language (Communicate with the interrogator)
  - Represent knowledge (Store information)
  - Automated Reasoning (Answer questions, draw conclusions)
  - Learn and adapt to the new situations (Adapt behavior, detect patterns)
- Total Turing test included vision and robotics.

Question:

1. Is it important that an intelligent system act like a human?
2. Compare the Turing test and Chinese room experiment.

2.11 Limits of AI Today

Today's successful AI systems operate in well-defined domains and employ narrow, specialized knowledge. Common sense knowledge is needed to function in complex, open-ended worlds. Such a system also needs to understand unconstrained natural language. However these capabilities are not yet fully present in today's intelligent systems.

2.12 What can AI Systems do

Today's AI systems have been able to achieve limited success in some of these tasks.

- In Computer vision, the systems are capable of face recognition
- In Robotics, we have been able to make vehicles that are mostly autonomous.
- In Natural language processing, we have systems that are capable of simple machine translation.
- Today's Expert systems can carry out medical diagnosis in a narrow domain
- Speech understanding systems are capable of recognizing several thousand words continuous speech
- Planning and scheduling systems had been employed in scheduling experiments with the Hubble Telescope.
- The Learning systems are capable of doing text categorization into about a 1000 topics
- In Games, AI systems can play at the Grand Master level in chess (world champion), checkers, etc.

2.13 What can AI systems NOT do yet?

- Understand natural language robustly (e.g., read and understand articles in a newspaper)
- Surf the web
- Interpret an arbitrary visual scene
- Learn a natural language
- Construct plans in dynamic real-time domains
- Exhibit true autonomy and intelligence

3 ARTIFICIAL INTELLIGENCE AS THE DESIGN OF AGENTS

3.1 Learning Outcomes

At the end of this study theme, the student will have knowledge and understanding of:

- The difference between an agent and other categories of intelligent systems.
- Characterizing and contrasting the standard agent architectures.
- The relationship between agents and the environment in which they operate
- The important elements that constitute an intelligent agent.
- The use of the state-of-the-art intelligent agent technologies in important application domains

3.2 Definitions

3.2.1 Agent:

- One that acts or has the power or authority to act.
- One empowered to act for or represent another: an author's agent; an insurance agent.

3.2.2 Intelligent Agent:

Anything that can be viewed as perceiving its environment through sensors and acting upon that environment through its effectors to maximize progress towards its goals. An agent perceives its environment through sensors. The complete set of inputs at a given time is called a *percept*. The current percept or a sequence of *percepts* can influence the actions of an agent. The agent can change the environment through actuators or effectors. An operation involving an effector is called an *action*. Actions can be grouped into *action sequences*. The agent can have goals which it tries to achieve. Hence, an agent gets percepts one at a time, and maps this percept sequence to actions.

Hence, an agent gets percepts one at a time, and maps this percept sequence to actions.

- **PAGE** (Percepts, Actions, Goals, Environment)

3.3 Agent Performance

An agent function implements a mapping from perception history to action. The behavior and performance of intelligent agents have to be evaluated in terms of the agent function. The **ideal mapping** specifies which actions an agent ought to take at any point in time. The **performance measure** is a subjective measure to characterize how successful an agent is. The success can be measured in various ways. It can be measured in terms of speed or efficiency of the agent. It can be measured by the accuracy or the quality of the solutions achieved by the agent. It can also be measured by power usage, money, etc.

3.4 Examples of Agents

1. Human agent
   (a) They have eyes, ears, taste buds and other organs for sensors;
   (b) They have hands, legs, mouth, fingers and other body parts for actuators
2. Robotic agent
   (a) Robots may have cameras, sonar, bumper and infrared range finders for sensors;
   (b) They can have grippers, wheels, lights, speakers, various motors etc. for actuators
   (c) Some examples of robots are Xavier from CMU, COG from MIT, AIBO entertainment robot from SONY etc.
3. Software agents or softbots
   (a) They have some functions as sensors and some functions as actuators.
4. Expert systems like the Cardiologist.
5. Autonomous spacecrafts.
6. Intelligent buildings.

3.4.1 Example: Automated Taxi Driving System

- Percepts:
  - Video, sonar, speedometer, engine sensors, keyboard input, microphone, GPS,...
- Actions:

  - Steer, accelerate, brake, horn, speak/display, ...

- Goals:
  - Maintain safety, reach destination, maximize profits (fuel, tire wear), obey laws, provide passenger comfort, ...

- Environment:
  - Kenya urban streets, traffic, pedestrians, weather, customers, ...

Note: Different aspects of driving may require different types of agent programs!

More Examples of Agents

Agent Type	Percepts	Actions	Goals	Environment
Bin-Picking Robot	Images	Grasp objects; Sort into bins	Parts in correct bins	Conveyor belt
Medical Diagnosis	Patient symptoms, tests	Tests and treatments	Healthy patient	Patient & hospital
Excite's Jango product finder	Web pages	Navigate web, gather relevant products	Find best price for a product	Internet
Webcrawler Softbot	Web pages	Follow links, pattern matching	Collect info on a subject	Internet
Financial forecasting software	Financial data	Gather data on companies	Pick stocks to buy & sell	Stock market, company reports

Figure 2: Examples of agents

3.5 Agent Faculties

The fundamental faculties of intelligence are

- Acting
- Sensing
- Understanding, reasoning, learning

Blind action is not a characterization of intelligence. In order to act intelligently, one must sense. Understanding is essential to interpret the sensory percepts and decide on an action. Many robotic agents stress sensing and acting, and do not have understanding.

3.6 Intelligent Agents and Artificial Intelligence

Artificial Intelligence is the study of rational agents. A rational agent carries out an action with the best outcome after considering past and current percepts. A rational agent acts to achieve goals, given beliefs. An Intelligent Agent must sense, must act, must be autonomous (to some extent). It also must be rational. Artificial Intelligence is about building rational agents.

3.7 Characteristics / Properties of Agents

1. Autonomy
   (a) The agent can act without direct intervention by humans or other agents and that it has control over its own actions and internal state.
2. Situatedness
   (a) The agent receives some form of sensory input from its environment, and it performs some action that changes its environment in some way.
3. Adaptivity
   (a) Intelligent agent is capable of
      i. reacting flexibly to changes in its environment (i.e. reactive to the environment);
      ii. taking goal-directed initiative (i.e., is pro-active), when appropriate;
      iii. learning from its own experience, its environment, and interactions with others.
4. Sociability
   (a) The agent is capable of interacting in a peer-to-peer manner with other agents or humans.

3.8 How is an Agent different from other software?

- Agents are autonomous, that is, they act on behalf of the user
- Agents contain some level of intelligence, from fixed rules to learning engines that allow them to adapt to changes in the environment
- Agents don't only act reactively, but sometimes also proactively
- Agents have social ability, that is, they communicate with the user, the system, and other agents as required

- Agents may also cooperate with other agents to carry out more complex tasks than they themselves can handle
- Agents may migrate from one system to another to access remote resources or even to meet other agents

3.9 Agent Environment

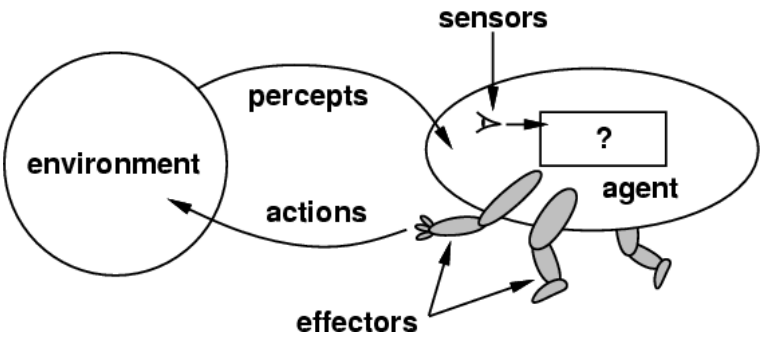


Figure 3: Agent and its environment

- The agent function maps from percept histories to actions: [f: P* $\rightarrow$ A]
- The agent program runs on the physical architecture to produce f
- agent = architecture + program

3.9.1 Specifying the Task Environment

- To design a rational agent, we must specify the task environment
- Environments in which agents operate can be defined in different ways.

3.9.2 PEAS

- Performance Measure:
  - Captures agent 's aspiration
- Environment :
  - Context , restrictions
- Actuators:
  - Indicates what the agent can carry out
- Sensors:
  - Indicates what the agent can perceive

3.9.3 Properties of Environments

Accessible/Inaccessible.

If an agent's sensors give it access to the complete state of the environment needed to choose an action, the environment is accessible. Such environments are convenient, since the agent is freed from the task of keeping track of the changes in the environment.

Deterministic/Non-deterministic.

An environment is deterministic if the next state of the environment is completely determined by the current state of the environment and the action of the agent. In an accessible and deterministic environment, the agent need not deal with uncertainty.

Episodic/Non-episodic.

An episodic environment means that subsequent episodes do not depend on what actions occurred in previous episodes. Such environments do not require the agent to plan ahead. In a sequential environment, the agent engages in a series of connected episodes

Static/Dynamic.

A static environment does not change while the agent is thinking. A Dynamic Environment changes over time independent of the actions of the agent and thus if an agent does not respond in a timely manner, this counts as a choice to do nothing.

Discrete/Continuous.

If the number of distinct percepts and actions is limited, the environment is discrete, otherwise it is continuous. Example: Chess vs. driving.

Note: The environment types largely determine the agent design

3.10 Agent Architectures

3.10.1 Table-driven agents

The action is looked up from a table based on information about the agent's percepts. A table is simple way to specify a mapping from percepts to actions. The mapping is implicitly defined by a program. The mapping may be implemented by a rule based system, by a neural network or by a procedure. There are several disadvantages to a table based system. The tables may become very large. Learning a table may take a very long time, especially if the table is large. Such systems usually have little autonomy, as all actions are pre-determined.

3.10.2 Simple reflex agents or State-based agent

State based agents differ from percept based agents in that such agents maintain some sort of state based on the percept sequence received so far. The state is updated regularly based on what the agent senses, and the agent's actions. Keeping track of the state requires that the agent has knowledge about how the world evolves, and how the agent's actions affect the world. Thus a state based agent works as follows:

- information comes from sensors - percepts
- based on this, the agent changes the current state of the world
- based on state of the world and knowledge (memory), it triggers actions through the effectors

3.10.3 Goal-based agent / Agents with memory

The goal based agent has some goal which forms a basis of its actions. Such agents work as follows:

- information comes from sensors - percepts
- changes the agents current state of the world
- based on state of the world and knowledge (memory) and goals/intentions, it chooses actions and does them through the effectors.

Goal formulation based on the current situation is a way of solving many problems and search is a universal problem solving mechanism in AI. The sequence of steps required to solve a problem is not known a priori and must be determined by a systematic exploration of the alternatives.

3.10.4 Utility-based agents

Utility based agents provides a more general agent framework. In case that the agent has multiple goals, this framework can accommodate different preferences for the different goals. Such systems are characterized by a utility function that maps a state or a sequence of states to a real valued utility. The agent acts so as to maximize expected utility. They allow decisions comparing choice between conflicting goals, and choice between likelihood of success and importance of goal (if achievement is uncertain)

3.10.5 Learning agent

Learning allows an agent to operate in initially unknown environments. The learning element modifies the performance element. Learning is required for true autonomy.

3.11 Mobile Agents

- Programs that can migrate from one machine to another.
- Execute in a platform-independent execution environment.
- Require agent execution environment (places).
- Mobility not necessary a sufficient condition for agent hood.
- Practical but non-functional advantages:
  - Reduced communication cost (eg, from PDA)
  - Asynchronous computing (when you are not connected)
- Two types:
  - One-hop mobile agents (migrate to one other place)
  - Multi-hop mobile agents (roam the network from place to place)
- Applications:
  - Distributed information retrieval.
  - Telecommunication network routing.

3.11.1 A mail agent

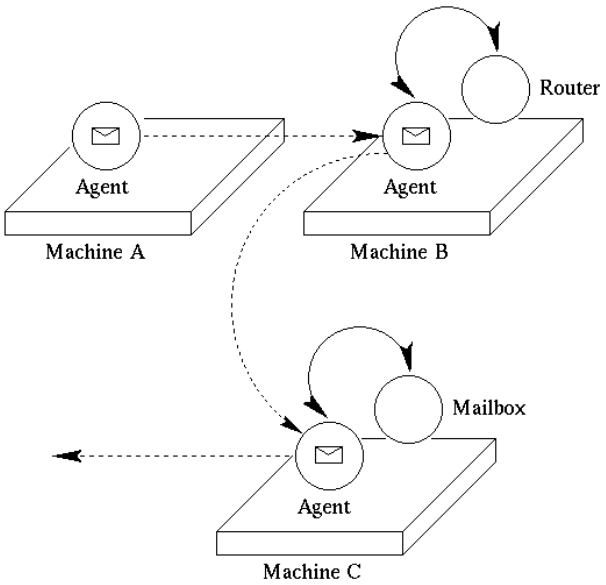


Figure 4 : Mail agent

3.11.2 Information Agents

- Manage the explosive growth of information.

- Manipulate or collate information from many distributed sources.
- Information agents can be mobile or static.
- Example:
  - BargainFinder comparison shops among Internet stores for CDs
- Internet Softbot infers which internet facilities (finger, ftp, gopher) to use and when from high-level search requests.

4 PROBLEM SOLVING AND SEARCH

If Artificial Intelligence can inform the other sciences about anything, it is about problem solving and, in particular, how to search for solutions to problems. Much of AI research can be explained in terms of specifying a problem, defining a search space which should contain a solution to the problem, choosing a search strategy and getting an agent to use the strategy to find a solution.

4.1 Learning Outcomes

Upon successful completion of this lesson, the student will be able to:

- Describe the role of search in AI.
- Explain and give examples of the basic types of search algorithms.
- Discuss the computational complexity of search algorithms.

4.2 Problem

Collection of information that the agent uses to decide what to do.

4.3 Problem Solving Techniques in Artificial Intelligence

Problems are tackled in AI using two main broad approaches namely:

1. Search techniques e.g in games
2. Modeling natural phenomena
   (a) Using Knowledge Base Systems (KBS)
   (b) Using Machine Learning techniques e.g. Artificial Neural Networks, Decision Trees, Case-base reasoning, Genetic algorithms etc

4.4 Specifying Search Problems (Single-State Problem Formulation)

There are three initial considerations in problem solving (as described in Russell and Norvig):

1. **Initial State** Firstly, the agent needs to be told exactly what the initial state is before it starts its search, so that it can keep track of the state as it searches.

2. **Operators** An operator is a function taking one state to another via an action undertaken by the agent.Plainexample, in chess, an operator takes one arrangement of pieces on the board to another arrangement by the action of the agent moving a piece.

3. **Goal** Test It is essential when designing a problem solving agent to know when the problem has been solved, i.e., to have a well defined goal test.

In summary, a problem is defined by four items:

- Initial state
- Successor function S(x) = set of action–state pairs
- Goal test, can be explicit
- Path cost (additive) e.g., sum of distances, number of actions executed, etc. Usually given as c(x, a, y), the step cost from x to y by action a.

A solution is a sequence of actions leading from the initial state to a goal state

4.5 Formulating Problem as a Graph

A search problem is represented using a directed graph. The states are represented as nodes while the allowed steps or actions are represented as arcs. In the graph, each node represents a possible state;

- a node is designated as the initial state;
- One or more nodes represent goal states, states in which the agent's goal is considered accomplished.
- each edge represents a state transition caused by a specific agent action;
- associated to each edge is the cost of performing that transition.

4.5.1 Process of Searching

Searching proceeds as follows:

1. Check the current state;
2. Execute allowable actions to move to the next state;
3. Check if the new state is the solution state; if it is not then the new state becomes the current state and the process is repeated until a solution is found or the state space is exhausted.

4.5.2 Example of a Search Problem

The figure below contains a representation of a map. The nodes represent cities, and the links represent direct road connections between cities. The number associated to a link represents the length of the corresponding road.

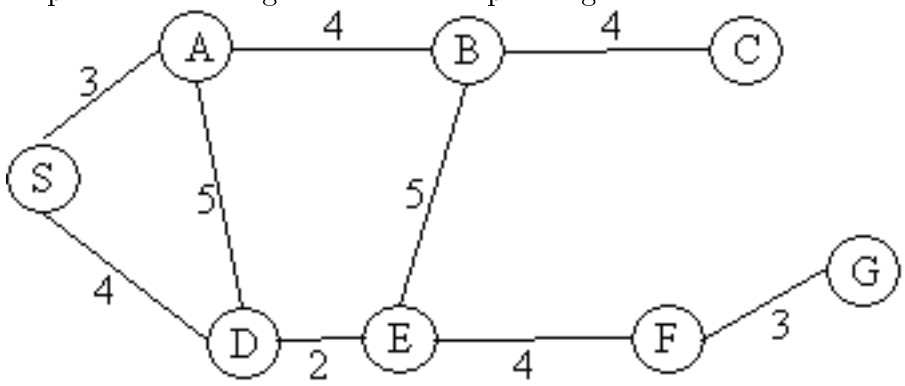


Figure 5: The search problem is to find a path from a city S to a city G

4.5.3 Evaluating Search Strategies

Different search strategies are evaluated in terms of three criteria:

1. Completeness - In a search problem, there may be any number of solutions, and the problem specification may involve finding just one, finding some, or finding all the solutions. If our search strategy is guaranteed to find all the solutions eventually, then we say that it is complete.

2. Time and Space Tradeoffs - What is the search cost associated with the time and memory required to find a solution?

   (a) Time complexity: Time taken (number of nodes expanded) (worst or average case) to find a solution.

   (b) Space complexity: Space used by the algorithm (Memory required to perform search)

3. Optimality - Finding of the highest-quality solution when a number exist.

4.6 Search Methods

There are two broad classes of search methods:

1. Uninformed (or blind) search methods:

Systematically explore all options available to the present state. They have no information about the number of steps or the cost from the Standart state to the goal.

Searching may yield a solution or the state space may be exhausted without a solution. There are two main types of blind search:

a) Depth-first Search (DFS)

b) Breadth-first Search (BFS)

These strategies differ in the order in which nodes are expanded. Blind Search quickly leads to search spaces that are too large.

2. Heuristically informed search methods.

The knowledge about the problem domain is used to guide the search mechanism thus enabling it to avoid wrong options. They considers the problem characteristics such as the cost of the path taken, the closeness of each state to the goal, etc. Heuristics are rules that apply most of the time but not all the time. Heuristic Function (h(n)), is a function that calculates the cost of reaching the goal state from the node n. There are two Basic Approaches:

a) Greedy Search

- Minimizes the estimated cost to reach the goal.

b) A* Search

- Minimizes total path cost.

4.7 Uninformed search

4.7.1 Breadth-first search

Looks for the goal node among all the nodes at a given level before using the children of those nodes to push on. The nodes of the same level are visited first.

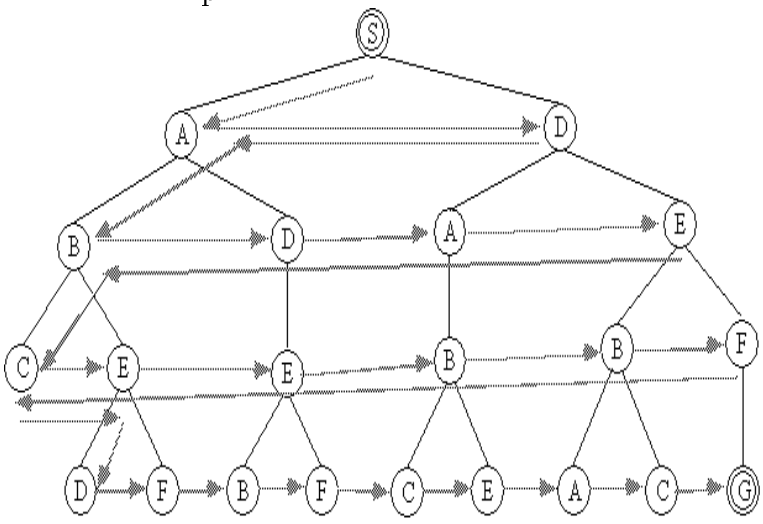


Figure 6: Breadth-first search of a path from node S to node G.

Finds the shallowest goal state. Uses a lot of memory because the entire search tree must be stored.

Properties of Breadth-First Search

- Time and space complexity is a problem
- Complete.
- Optimal if cost is directly related to depth.
- If there is a solution, breadth-first is guaranteed to find it.
- Modifications:
  - Uniform Cost Search

Advantages of Breadth First Search

- Finds the path of minimal length to the goal.

Disadvantages of Breadth First Search

- Requires the generation and storage of a tree whose size is exponential the the depth of the shallowest goal node.

4.7.2 Depth-first search

Looks for the goal node among all the children of the current node before using the sibling of this node to push on. Expands nodes at the deepest level of the search tree. Stores only one path from root to leaf.

1. Space complexity not a problem
2. Time could be wasted going down the wrong branch of the tree.
3. Modifications:
   (a) Depth Limited Search
   (b) Iterative Deepening Search

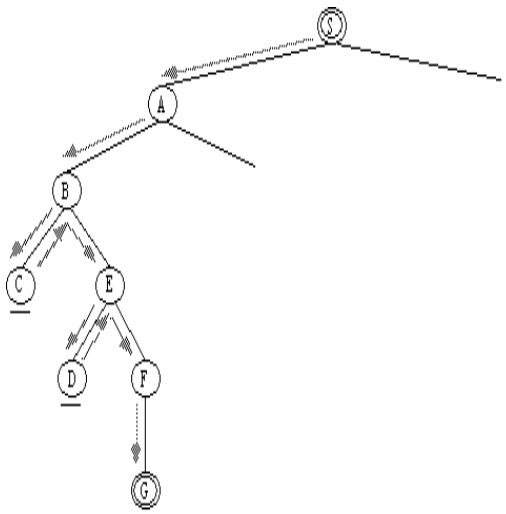


Figure 7: Depth Limited Search

4.7.3 Depth Limited Search

Set a limit on the depth of the search. Addresses the problem of wasting time going down the wrong branch. The difficulty with this approach is choosing a suitable limit.

4.7.4 Iterative Deepening Search

Combines DFS and BFS. Iterative depth limited search. Overcomes the problem of choosing a suitable limit. Start with a very low limit and increase gradually until a solution is found.

4.7.5 Uniform-cost Search

Looks for the cheapest path from the start node to a goal node. Expands the lowest cost node, rather than the lowest depth node. Cost of an arc from node i to node j is denoted by C(i,j). For instance, C(S,A) = 3. Cost of a path from node x to node y is the sum of the costs of the component arcs

C(S,F) = C(S,D) + C(D,E) + C(E,F) = 10

This method always extends the cheapest path found so far. In this way it is guaranteed to find the cheapest path, if one exists.

Uniform-cost search of a path from city S to city G.

Step 1: Build all the paths from S to its neighboring cities. These paths are (S A) and (S D). Attach to the end node of each path the cost of the path.

Step 2: The cheapest (shortest) path found so far is (S A) the cost (length) of which is 3. Extend this path to reach the neighbors of A (except S, which is already on the path). The paths built in this way are (S A B) and (S A D). Attach the cost of the path to the end node of each path. Notice, however, that D can be reached on a shorter path (S D), of cost 4. Therefore we are not going to further extend the path (S A D) whose cost is 8.

Step 3: The cheapest extendable path found so far is (S D) the cost of which is 4. Extend this path to reach the neighbors of D (except S). The paths built in this way are (S D A) and (S D E). Attach the cost of the path to the end node of each path. Notice, however, that A can be reached on a shorter path (S A), of cost 3. Therefore we are not going to further extend the path (S D A) whose cost is 9.

Step 4: The cheapest extendable path found so far is (S D E) the cost of which is 6. Extend this path to reach the neighbors of E (except those already contained in the path). The paths built in this way are (S D E B) and (S D E F). Attach the cost of the path to the end node of each path. Notice, however, that B can

be reached on a shorter path (S A B), of cost 7. Therefore we are not going to further extend the path (S D E B) whose cost is 11.

Step 5: The cheapest extendable path found so far is (S A B) the cost of which is 7. Extend this path to reach the neighbors of B (except those already contained in the path). The paths built in this way are (S A B C) and (S A B E). Attach the cost of the path to the end node of each path. Notice, however, that E can be reached on a shorter path (S D E), of cost 6. Therefore we are not going to further extend the path (S A B E) whose cost is 12.

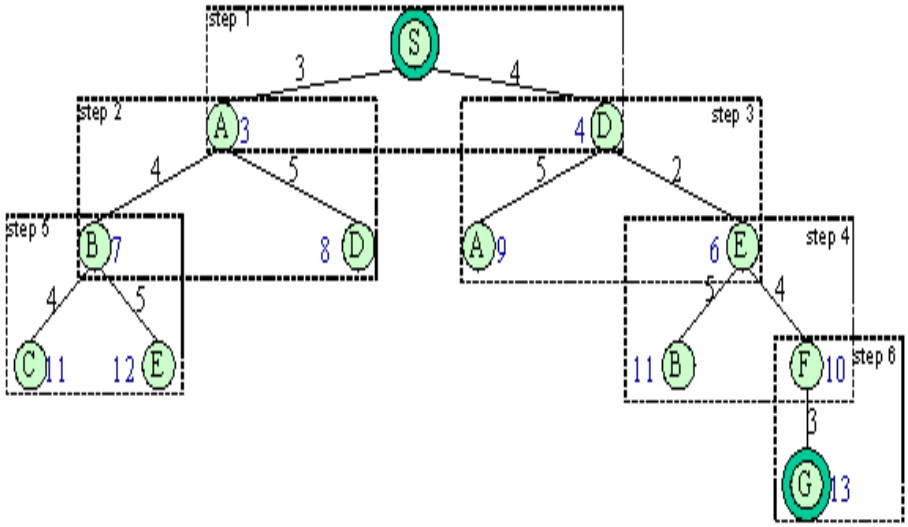


Figure 8: Uniform cost search

Step 6: The cheapest extendable path found so far is (S D E F) the cost of which is 10. Extend this path to reach the neighbors of F (except those already contained in the path). The path built in this way is (S D E F G). Because G is the destination, a path from S to G has been found. The cost of this path is 13. However, if there are other extendable paths with a cost less than 13, then these paths need to be extended, because they may lead to a shorter path to the goal.

Step 7: If there are other extendable paths with a cost less than 13, then these paths need to be extended, because they may lead to a shorter path to the goal. The only node that the system may try to extend is C. However C has no neighbor that is not already on the path to C. Moreover, no other leaf node in the tree is extendable. Therefore, at this point it can be concluded that the searching system has found that the cheapest path from city S to city G, which is (S D E F G), of cost 13. All the other possible paths from S to G will cost more than 13.

Properties of Uniform Cost Search Algorithm

- Complete

- Optimal/Admissible
- Exponential time and space complexity

4.8 Limitations of Exhaustive Search Methods

- Exhaustive search has the advantage of guaranteeing solutions if they exist.
- However, exhaustive search has some problems associated with the need to visit all states at times.
  - One problem is about combinatorial problems associated with some search cases. At times the number of possibilities rise and demands on storage and processing cannot be met.
  - The other problem is that the state space can be extremely large such that complete search is time consuming. Sometimes the state space can be inexhaustible, in that it is difficult to enumerate.

4.9 Informed / Heuristic search

Uses domain-dependent (heuristic) information in order to search the space more efficiently. Heuristic Function (h(n)), a function that calculates the cost of reaching the goal state from the node n.

Ways of using heuristic information

- Deciding which node to expand next, instead of doing the expansion in a strictly breadth-first or depth-first order;
- In the course of expanding a node, deciding which successor or successors to generate, instead of blindly generating all possible successors at one time;
- Deciding that certain nodes should be discarded, or pruned, from the search space.

4.9.1 Hill climbing

Hill climbing is depth-first search with a heuristic measurement that orders choices as nodes are expanded. It always selects the most promising successor of the node last expanded. Expand a node, sort children according to heuristic evaluation function then choose best action. For instance, one may consider that the most promising successor of a node is the one that has the shortest straight-line distance to the goal city G.

In figure 9, the straight line distances between each city and G is indicated in square brackets.

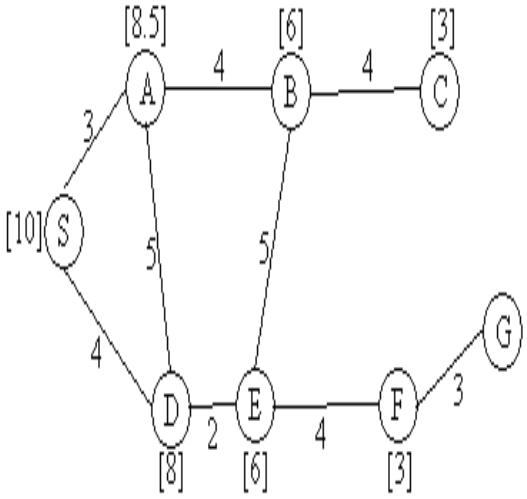


Figure 9: The map indicating also the straight line distances between each city and G.

The hill climbing search from S to G proceeds as follows:

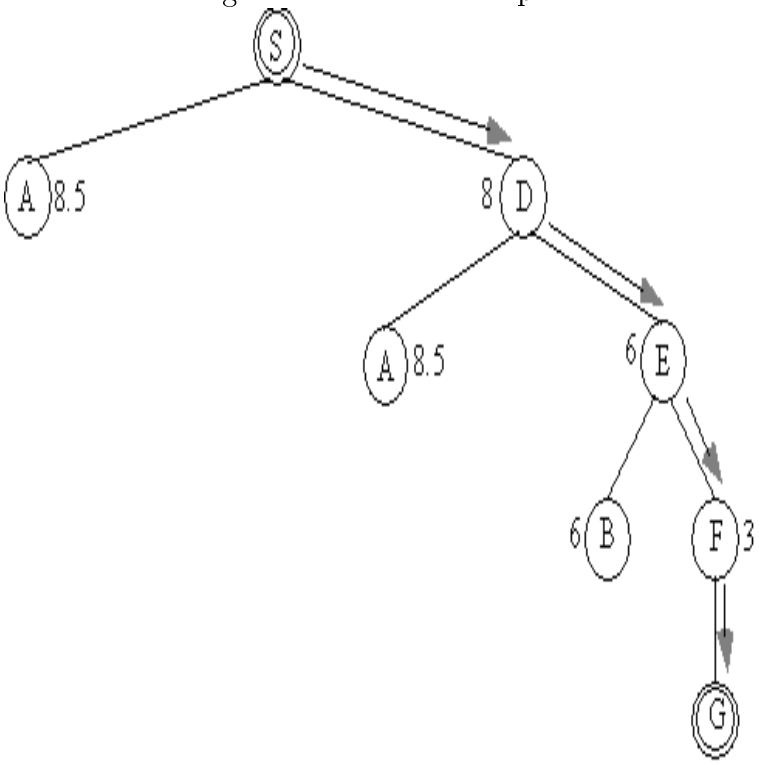


Figure 10: Hill climbing search of a path from node S to node G

Exercise 1. Exercise

Apply the hill climbing algorithm to find a path from S to G, in the map from **figure 9** above considering that the most promising successor of a node is its closest neighbor. Compare the search with the one from **figure 10.** What do you notice?

Apply now the same algorithm to the following (slightly modified) network:

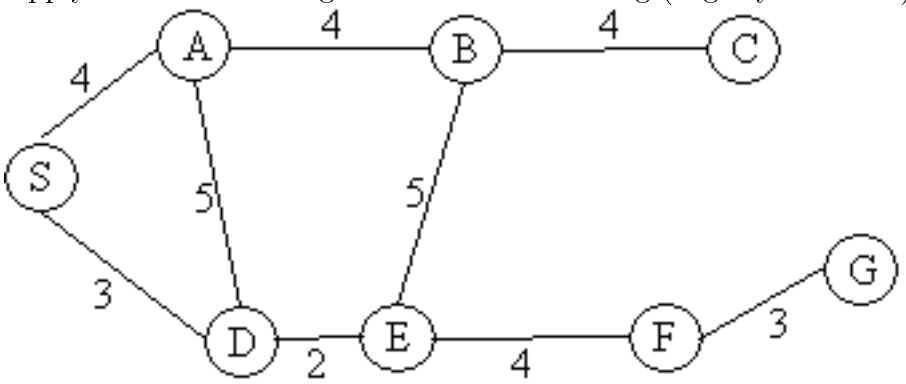


Figure 11: A modification of the map from Figure 9

4.9.2 Greedy best-first search

Greedy best-first search is a generalization of the previous search methods. The idea is to always expand the leaf node (of the current search tree) that seems "most promising" to lead to a goal node. Minimizes the **estimated cost to reach the goal.** The node closest to the goal state, as determined by h(n), is expanded first. The promise of a node may be estimated by using an evaluation function f* which is defined so that the more promising a node is, the smaller is the value of f*. The node selected for expansion is the one at which f* is minimum. Let us notice that the blind search methods correspond to a best-first search with a particular functions f*:

- Breadth-first search: f*(i) = the depth of node i (the distance from the root to i)
- Depth-first search: f*(i) = the depth of node i
- Uniform-cost search: f*(i) = the cost of the path from the start node to i

This shows that the definition of f* is essential for the usefulness of the best-search method. The difference between the hill climbing search method and the best first search method is the following one:

- the best first search method selects for expansion the most promising leaf node of the current search tree;
- the hill climbing search method selects for expansion the most promising successor of the node last expanded.

4.9.3 The search algorithm A*

The A* algorithm is a refinement of the best-first search method. The goal of the A* algorithm is to find a minimal cost-path joining the start node and a goal node. It

minimizes total path cost. It uses the following evaluation function to estimate the promise of a node n:

f*(n) = g*(n) + h*(n) where:

- **g*(n)** estimates the minimum cost of a path from the start node to node n;
- **h*(n)** estimates the minimum cost from node n to a goal.

The node chosen for expansion is the one for which f* is minimum. f*, g*, and h* are estimations of the real minimum costs denoted by f, g, and h, respectively. As one could notice, A* is very similar to the uniform-cost search, except that it uses a different function (which is a heuristic function) to estimate the promise of a node. In fact, A* reduces to uniform-cost search if h*(n) is always considered to be 0.

Example: Find a path from city S to city G by using the following functions:

g*(X) = the actual distance from S to X, along the cheapest path found so far by the algorithm.

h*(X) = the straight-line distance between X and G (this is known from the map).

h(X) = the real shortest distance between X and G (this is unknown).

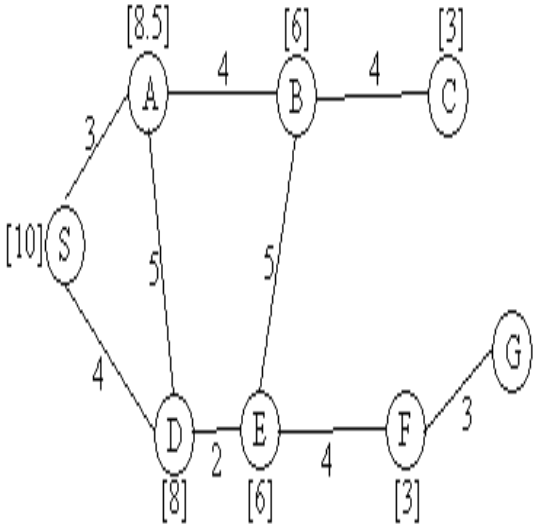


Figure 12:The map indicating also the straight line distances between each city and G

In the above figure, the value of h* for each node is represented in square brackets. That is, h*(D) = 8.

The following figure shows the steps of the A* algorithm for the considered problem:

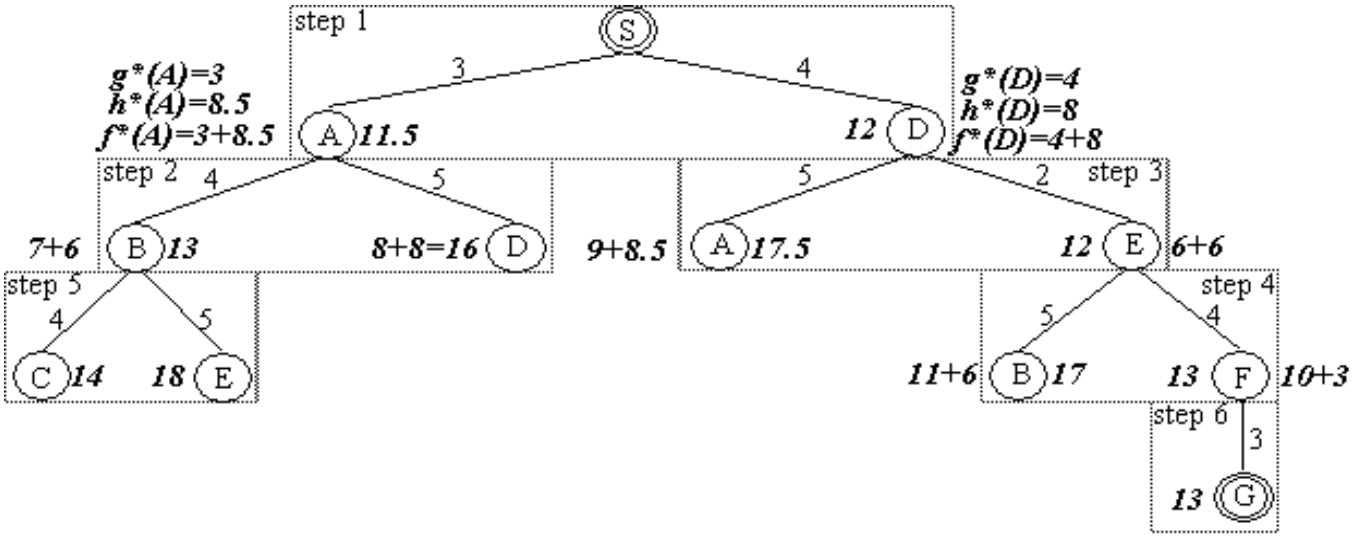


Figure 13: A* Search

4.10 Other applications of search methods

1. Route Finding

   (a) Routing in computer networks;

   (b) Automated travel advisory systems;

   (c) Airline travel planning systems.

2. Touring and Traveling Salesperson Problems

   Planning trips for traveling salespersons;

   (a) Planning movements for automatic circuit board drills.

3. Very Large Scale Integrated Circuit (VLSI) Layout

   (a) Cell layout on the chip;

   (b) Channel routing between the cells.

4. Robot Navigation

   (a) In a two dimensional space (moving on a surface);

   (b) In a multidimensional space (when the robot has arms and legs that must be controlled).

5. Assembly Sequencing

   (a) Checking the feasibility of the steps in a sequence (a geometrical search problem).

Exercise

1. Investigate how to program models related to agents, neural networks and genetic algorithms.
2. How are problems solved in artificial intelligence?
3. What is searching?
4. Discuss how to handle problems using searching technique.
5. Discuss exhaustive search and heuristic search.
6. Discuss relevance of the search technique and other examples where the search technique may be applied.

5 KNOWLEDGE-BASED AGENTS AND LOGICAL PROBLEM SOLVING

5.1 Learning Outcomes

Upon successful completion of this lesson, the student will be able to:

- Explain the use of knowledge by agents
- Discuss different methods of representing knowledge (Logic, semantic networks, frames, production rules)
- Discuss the advantages and shortcomings of different reasoning methods

5.2 Review of AI Techniques

There are various techniques that have evolved that can be applied to a variety of AI tasks. These techniques are concerned with how we represent, manipulate and reason with knowledge in order to solve problems. This includes:

1. Knowledge Representation
2. Search

5.2.1 Knowledge Representation

Definitions:

1. Knowledge
   (a) A theoretical or practical understanding of a subject or domain.
   (b) Knowledge is also the sum of what is currently known.
   (c) A set of representations of facts about the world. Each representation is a sentence.
2. Knowledge Base
   (a) Forms the system's intelligence source
   (b) Inference mechanism uses it to reason and draw conclusions
3. Knowledge representation

It is the way that knowledge is stored in a program. This implies:

- There is a systematic way to store the information
- The knowledge is coded into the program

5.3 Review of Knowledge Based Systems

5.3.1 Definition

A software system capable of supporting the explicit representation of knowledge in some specific competence domain and of exploiting it through appropriate reasoning mechanism in order to provide high-level problem-solving performance

5.3.2 Types of Knowledge Based Systems

In general, KBS are classified according to the human behavior they attempt to mimic.

Expert Systems They model the higher order cognitive functions of the human mind. They are used to mimic the decision making process of the human mind.

Neural Networks They model the brain at the biological level. They are adept at pattern recognition and introduce the concept of learning into computing.

Case Based Reasoning Models the human ability to learn from past experience. They borrow from the legal system where past cases are used as a basis for making decisions in the present cases.

5.3.3 Data -> Information -> Knowledge

- DATA- measurements or records about events. Can be numerical, alphabetical, images, sounds, etc.
- INFORMATION – analyzed and organized data such that we know its characteristics
- KNOWLEDGE – information put into a specific context

5.3.4 Main Components of Knowledge Based Systems

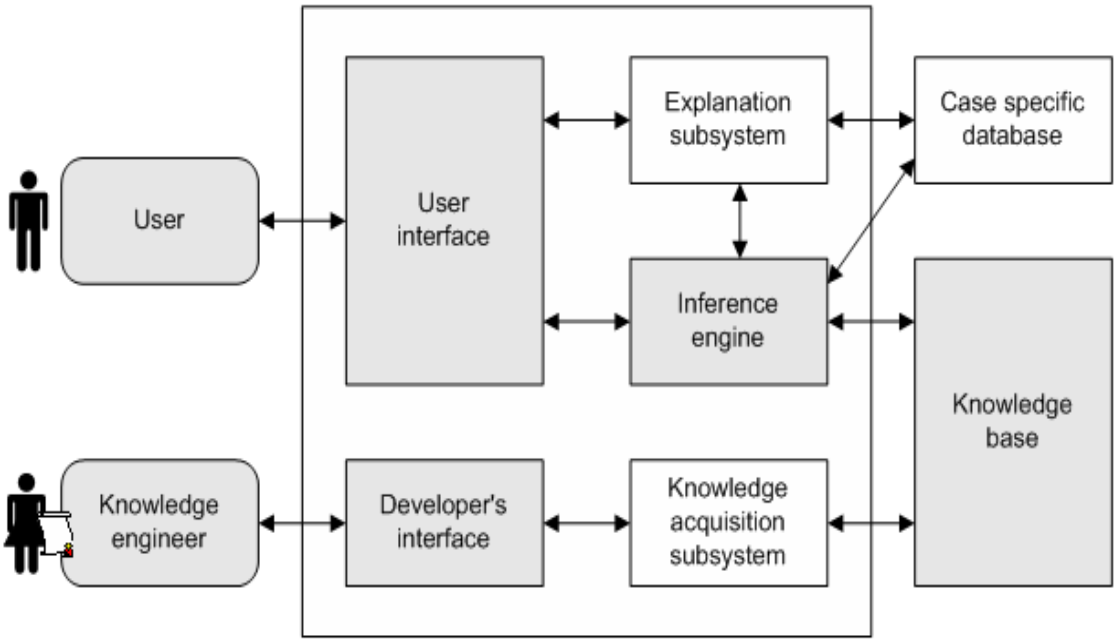


Figure 16: Components of Knowledge Based Systems

5.3.5 Expert Systems

Definition

1. An Expert System (ES) is computer-based systems (mainly software) that uses knowledge and facts, and apply an appropriate reasoning technique (inferencing) to solve problems in a given field (domain) that normally require the services of human experts.

2. A computer application that performs a task that would otherwise be performed by a human expert.

3. A model and associated procedure that exhibits, within a specific domain, a degree of expertise in problem solving that is comparable to that of a human expert. – Ignizio

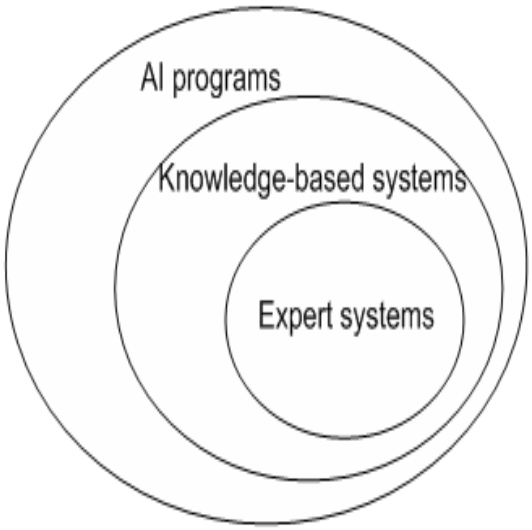
4. An expert system is a computer system which emulates the decision-making ability of a human expert.-Giarratono

Brief History Examples of the early and famous expert systems

- DENDRAL - Stanford Univ. (1965)
  - Analysis of chemical compounds

  - Rule-based system

- CADACEUS - Univ. of Pittsburgh (1970)
  - Diagnosis of human internal diseases
- MYCYSMA - MIT (1971)
  - Symbolic mathematical analysis
- MYCIN - Stanford Univ. (1972)
  - Diagnosis and treatment of infectious blood diseases o Rule-based system
- EMYCIN - Stanford Univ. (1978)
  - Evolved from (Empty) Mycin
  - First ES Shell
- XCON - DEC (1980)
  - configures VAX computer system
  - First ES used commercially
- 1943 Post, E.L proved that any computable problem can be solved using simple IF-THEN rules
- 1962 General Problem Solver (GPS) by A.Newell and H.Simon



Structure and characteristics

Figure 16: Structure of ES

- AI programs:
  - Intelligent problem solving tools
- KBSs
  - AI programs with special program structure separated knowledge base
- ESs
  - KBSs applied in a specific narrow field

Types of ES

- Ruled Based Expert System
  - Represented as a series of rules
- Frame-Based System
  - Representation of the object-oriented programming approach
- Hybrid System
  - Include several knowledge representation approach
- Model-Based System
  - Structured around the model that stimulates the structure and function of the system under study
- Ready-Made (Off-the-shelf) System
  - Custom-made, similar to application package such as an accounting general ledger or project management in operation mgmt.

Classic problem areas addressed by Expert Systems Expert systems have been used in several typical problem areas. Several application systems have been demonstrated in these areas. These areas are given below.

Category	Problem area addressed
Interpretation	Infer situation descriptions from observations
Prediction	Infer consequences of given situations
Diagnosis	Infer malfunctions from observations
Design	Configure objects under constraints
Planning	Develop plans to achieve goals
Monitoring	Compare observations and plans, flag exceptions
Debugging	Prescribe remedies to malfunctions
Repair	Execute plans to administer prescribed remedy
Instruction	Diagnose, debug, and correct student performance
Control	Interpret, predict, repair and monitor system behaviour

5.3.6 Designing ES

The process of building ES is called Knowledge Engineering, consist of three stages :

1. **Knowledge acquisition** : the process of getting the knowledge from experts
2. **Knowledge representation** : selecting the most appropriate structures to represent the knowledge
3. **Knowledge validation** :testing that the knowledge of ES is correct and complete

Software for Building Expert Systems Expert System tools: Software that is used for constructing expert systems. The tools range from programs used for building expert systems to programs that can aid the knowledge acquisition process. The main software tools for developing expert system fall into the following categories.

- Development software-programming languages (AI languages and general purpose languages); shells and AI toolkits
- Development support tools - I/O facilities; debugging aids; explanation facilities; Editors
- Systems building tools e.g. Inule induction engines

5.4 Knowledge-based Agent

Central component of a knowledge-based agent is its knowledge-base (KB). A KB is a set of representations of facts about the world. Each individual representation is called a sentence. The sentences are expressed in a language called a knowledge representation language. A knowledge-based agent should be able to infer.

5.5 Desirable features of any knowledge representation scheme

Completeness Should support the acquisition of all aspects of the knowledge

Conciseness This should allow efficient acquisition so that knowledge is stored compactly and is easily retrieved

Computational Efficiency It should be possible to use the knowledge rapidly and without the need for excessive computation

Transparency It should be possible to understand its behavior and how it arrives at conclusions

Explicity The important things should be explicit, but the details suppressed but available in case it is required in future

5.6 Major Knowledge Representation Schemes:

1. Logic
   (a) Propositional Logic
   (b) First Order Logic
2. Production Rules
3. Structured Objects
   (a) Semantics Networks
   (b) Frames

5.7 Logic

Logic, almost by definition, has a well defined syntax and semantics, and is concerned with truth preserving inference. It is a collection of rules used when doing logical reasoning. Its main purpose of is *the soundness or unsoundness of arguments*

Types of logic:

1. Propositional logic (logic of sentences)
2. Logic of objects (predicate logic)
3. Logic involving uncertainties
4. Logic dealing with fuzziness
5. Temporal logic etc.

5.7.1 Propositional Logic

Proposition:

It involves sentences that are either true=T (or 1) or false =F (or 0) (binary logic)

Examples

i) 'John is a happy man' -yes

ii) 'All cats are good pets- yes

iii) 'Mary's pet'- no

iv) 'Oh deer me!'- no

A proposition in general contains a number of variables. Example of propositional variables P, Q, R, S, . . . A table of all possible relationships between a proposition and its variables is called a truth table

Truth Tables

Logical Operators (connectives): Unary, binary

Unary

Negation, 'not.' Symbol: ¬

Example 1. Consider the following propositions

P: I am going to town

¬P: I am not going to town

I ain't goin'. It is not the case that I am going to town

Truth Table:

p	¬P
F	T
T	F

Binary Operators

Conjunction 'and' Symbol: ∧

Disjunction, 'or' Symbol: ∨

Exclusive OR Symbol: ⊕

P - 'I am going to town'

Q - 'It is going to rain'

P ∧ Q 'I am going to town and it is going to rain'

P∨ Q 'I am going to town or it is going to rain'

P⊕ Q 'Either I am going to town or it is going to rain

P	Q	P ∧ Q	P ∨ Q	P ⊕ Q
F	F	F	F	F
F	T	F	T	T
T	F	F	T	T
T	T	T	T	F

Implication
'if...then...' Symbol: $\rightarrow$
P - 'I am going to town'
Q - 'It is going to rain'
P $\rightarrow$ Q : 'If I am going to town then it is going to rain'

P	Q	P$\rightarrow$ Q
F	F	T
F	T	T
T	F	F
T	T	T

Note: The implication is false only when P is true and Q is false

Modus ponens

If (P $\rightarrow$Q) and P is true, then Q is true, written also as (((P $\rightarrow$ Q) and P) |– Q) i.e. if proposition P is true, and the rule of inference P $\rightarrow$Q is true, then Q will also be true.

Validity

A sentence of the form: Premises $\rightarrow$Conclusion is valid if all the rows o$\rightarrow$ are all true.

This is what enables machine to check premises to determine if the conclusion is true. A truth table can be used to check if a sentence is valid.

Consider the following inference:

- A: If Keith is a happy man
- B: Keith is a teacher

This could be expressed in propositional logic as:

- if A then B

Logic notation: A $\rightarrow$ B (Meaning proposition A implies proposition B)

Biconditional

'if and only if ', 'iff' Symbol: $\leftrightarrow$

P - 'I am going to town' Q - 'It is going to rain'

P Q : 'I am going to town if and only if it is going to rain.'

P	Q	P$\leftrightarrow$ Q
F	F	T
F	T	F
T	F	F
T	T	T

Note:

Both P and Q must have the same value.

Other binary operators NAND (|) NOR (↓)

Example 2. Consider the folowing propositions

'If I go to Harry's or go to the country then I will not go shopping.'
P: I go to Harry's
Q: I go to country
R: I will go shopping
$(P \vee Q) \rightarrow \neg R$

Note: A truth table with n propositional variables has - 2^n rows

P	**Q**	**R**	**¬R**	**(P ∨ Q) → ¬R**
F	**F**	**F**	**T**	**T**
F	**F**	**T**	**F**	**T**
F	**T**	**F**	**T**	**T**
F	**T**	**T**	**F**	**F**
T	**F**	**F**	**T**	**T**
T	**F**	**T**	**F**	**F**
T	**T**	**F**	**T**	**T**
T	**T**	**T**	**F**	**F**

Propositional Equivalence

A tautology is a proposition which is always true

Example : $P \vee \neg P$

A contradiction is a proposition which is always false

Example : $P \wedge \neg P$

A contingency is a proposition which is neither a tautology nor a contradiction.

Example: $(P \vee Q) \rightarrow \neg R$

Compound propositions with the same truth values in all possible cases are called **logically equivalent**

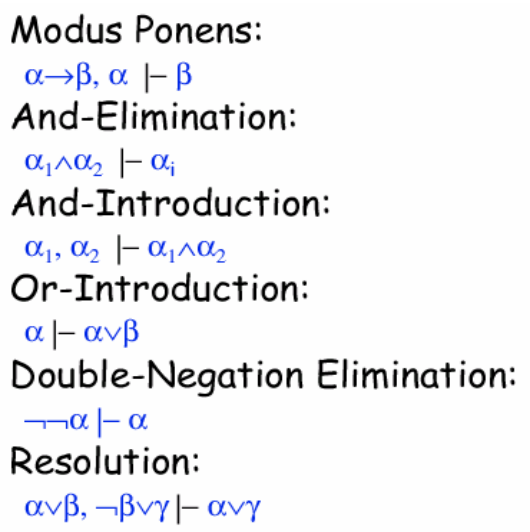
Logical equivalence

Example: Show that the proposition $\neg P \vee Q$ and $P \rightarrow Q$ are logically equivalent

P	Q	¬P ∨ Q	P → Q
F	F	T	T
F	T	T	T
T	F	F	F
T	T	T	T

P and Q are **logically equivalent** if $P \leftrightarrow Q$ is a tautology. We write $P \Leftrightarrow Q$

Propositional Logic: Rules of Inference



- Modus Ponens/Implication-Elimination:
  - From an implication and the premise of the implication, you can infer the conclusion
- And-Elimination:
  - From a conjunction you can infer any of the conjuncts.
- And-Introduction:
  - From a list you can infer their conjunction
- Or-Introduction:
  - From a sentence, you can infer its disjunction with anything else at all.
- Double-Negation Elimination:
  - From a doubly negated sentence, you can infer a positive sentence
- Unit Resolution:
  - From a disjunction, if one of the disjuncts is false, then you can infer the other one is true.
- Resolution:

– This is the most difficult. Because b cannot be both true and false, one of the other disjuncts must be true in one of the premises. Or equivalently, implication is transitive.

Advantage Can reason about the world; based on proven theory.

Disadvantage Components cannot be individually examined.

5.7.2 Predicate Logic (First Order Logic)

Is an extension of propositional calculus. Predicates are used of the form function (arguments), where function is any object or relationship.

Review of Propositional Logic

Propositional Logic is limited. The world only consists of facts. It is not possible to express general statements in e.g. cannot express the following:

- All men are mortal
- Socrates is a man
- Therefore, Socrates is mortal

Predicate Logic: Syntax

Predicate logic makes such general statements possible. Sentences in predicate calculus are built up from *atomic sentences*. Atomic sentences consist of a predicate name followed by a number of arguments. These arguments may be any *term*.

Terms may be:

- Constant symbols such as "Reagan".
- Variable symbols such as "X". For consistency with Prolog we'll use capital letters to denote variables.

It allows a proposition to be broken down into two components

- Arguments
- Predicates

It allows the use of variables, in addition to supporting the rules of inference derived from propositional logic (i.e. Modus ponens, etc)

Function Expressions

Function expressions consist of a functor followed by a number of arguments, which can be arbitrary terms, such as "father (Reagan)".

Atomic sentences in predicate logic include the following:

- friends(Alison, Richard)
- friends(father(Fred), father(Joe))
- likes(X, Richard)

Sentences in predicate logic are constructed (much as in propositional logic) by combining atomic sentences with logical connectives, so the following are all sentences in predicate calculus:

- friends(Alison, Richard) $\rightarrow$ likes(Alison, Richard)
- likes(Alison, Richard) $\vee$ likes(Alison, Waffles)
- ((likes(Alison, Richard) $\vee$ likes(Alison, Waffles)) $\wedge\neg$ likes(Alison, Waffles)) $\rightarrow$ likes(Alison, Richard)

Sentences can also be formed using *quantifiers* to indicate how any variables in the sentence are to be treated. The two quantifiers in predicate logic are and , so the following are valid sentences:

- $\exists$X bird(X) $\wedge\neg$ flies(X) i.e., there exists some bird that doesn't fly.
- $\forall$X (person(X) $\rightarrow \exists$Y loves(X,Y)) i.e., every person has something that they love.

A sentence should have all its variables quantified.

Quantifiers

Predicate logic allows the use of quantifiers, allowing the language to be extended to propositions that refer to a range of a variable.

$\forall$: The universal quantifier since it refers to all objects in the population

$\exists$: The existential quantifier since it refers to at least one object in the population

Representing Things in Predicate Logic

This section will just give a list of logical expressions paired with English descriptions, then some unpaired logical or English expressions - you should try and work out for yourself how to represent the English expressions in Logic, and what the Logic expressions mean in English.

- $\exists$X table(X) $\wedge$C$\neg$numberoflegs(X,4) i.e.There is some table that doesn't have 4 legs

- $\forall$X(macintosh(X) $\rightarrow \neg$realcomputer(X)) i.e. No macintosh is a real computer or If something is a macintosh then its not a real computer
- $\forall$X kenyan(X) $\rightarrow$ (supports(X,maunited) $\vee$ supports(X, arsenal)) i.e. All Kenyans support either Maunited or Arsenal
- $\exists$X small(X) $\wedge$ on(X,table) i.e. There is something small on the table

Example

- $\forall$x, Man(x)$\rightarrow\exists$y such that Woman(y) $\wedge$ Loves(x,y) i.e. For any object x in the world if x is a Man, then there exists an object y, such that y is a woman and x Loves y
- $\forall$x, Kenyan(x) $\exists$ Man(x) i.e. For any object x in the world if x is a Kenyan, then x is a man

From the two facts, it can concluded using the rules of inference, that the following facts must be true:

$\forall$ x, Kenyan(x) $\rightarrow$ y, such that Woman(y) $\wedge$Loves(x,y) i.e. Every Kenyan loves a woman

Advantages

- It has well defined rules for manipulation/ meaning;
- It is expressive.

Disadvantages

- Cannot handle uncertainty;
- Uses small primitives for descriptions whose numbers can be many.

Exercise: Translate English statements into predicate logic

1. All elephants are grey
2. Every apple is either green or yellow
3. There is some student who is intelligent

5.8 Structured Objects

The idea of structured objects is to represent knowledge as a collection of objects and relations, the most important relations being the subclass and instance relations. The subclass relation says that one class is a subclass of another, while the instance relation says that some individual belongs to some class.

5.8.1 Semantic Nets

A semantic network represents knowledge in the form of a graph, in which the nodes represent objects, situations, or events, and the arcs represent the relationships between them. The most important relations between concepts are *subclass* relations between classes and subclasses, and *instance* relations between particular objects and their parent class. However, any other relations are allowed, such as *has-part, is-a,* etc. Links represent relationships, which contain the structural information of the knowledge to be represented, and the label indicates the type of the relationship.

Example 3. : Representing knowledge about animals

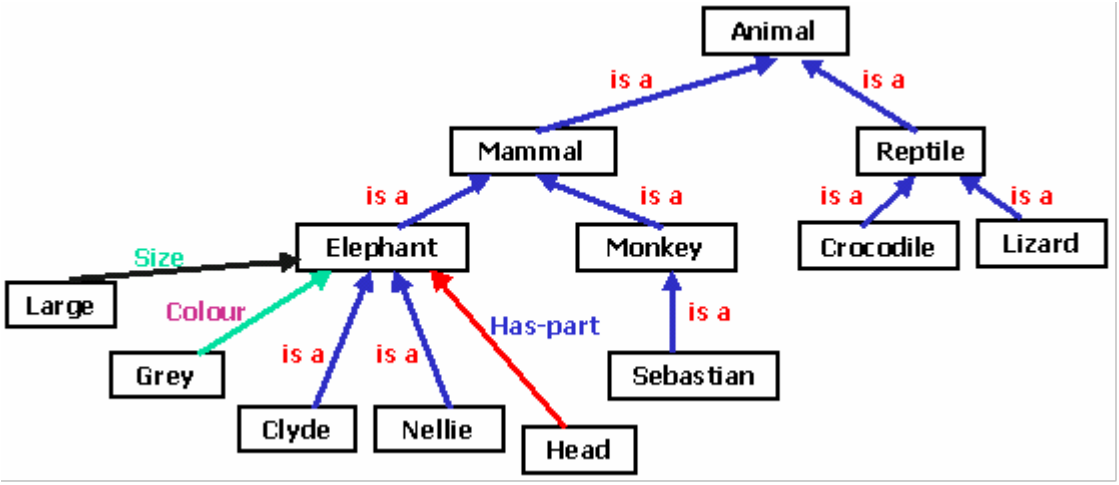


Figure 14: Semantic net representing animal relationships

This network represents the fact that mammals and reptiles are animals, that mammals have heads, an elephant is a large grey mammal, Clyde and Nellie are both elephants, and that Nellie likes apples. The subclass relations define a *class hierarchy.*

The subclass and instance relations may be used to derive new information which is not explicitly represented. We should be able to conclude that Clyde and Nellie both have a head, and are large and grey. They *inherit* information from their parent classes. Semantic networks normally allow efficient inheritance-based inferences using special purpose algorithms.

General features of the semantic networks

- Representational adequacy: High representational adequacy for binary relationships
- Inferential adequacy: Good inferential adequacy for certain types of inferential procedures.
- Inferential efficiency: The fundamental property of the semantic networks is that the structure used for representing knowledge is also a guide for the retrieval of the knowledge. Consequently, the processes of information retrieval are very efficient.

- Acquisitional efficiency: The acquisitional efficiency is low i.e. the knowledge added or deleted affects the rest of the knowledge.

Advantage

- Easy to translate to predicate calculus.

Disadvantages.

- Cannot handle quantifiers;
- Nodes may have confusing roles or meanings;
- Searching may lead to combinatorial explosion;
- Cannot express standard logical connectives;
- Can represent only binary or unary predicates.

5.8.2 Frames

In a frame, all the information relevant to a particular concept is stored in a single complex entity, called a frame. Superficially, frames look pretty much like record data structures. However frames, at the very least, support inheritance. They are often used to capture knowledge about *typical* objects or events, such as a typical bird, or a typical restaurant meal.

We could represent some knowledge about elephants in frames as follows:

Mammal	
Subclass:	Animal
Warm blooded:	yes

Elephant	
subclass	mammal
colour	grey
size	large

Clyde	
Instance	Elephant
Colour	Pink
Owner	Mutua

Nellie	
Instance	Elephant
Size	Small

A particular frame (such as Elephant) has a number of *attributes* or *slots* such as colour and size where these slots may be filled with particular values, such as grey. We have used a "*" to indicate those attributes that are only true of a typical member of the

class, and not necessarily every member. Most frame systems will let you distinguish between *typical attribute* values and definite values that must be true.

In the above frame system we would be able to infer that Nellie was small, grey and warm blooded. Clyde is large, pink and warm blooded and owned by Fred. Objects and classes inherit the properties of their parent classes unless they have an individual property value that conflicts with the inherited one.

Inheritance is simple where each object/class has a single parent class, and where slots take single values. If slots may take more than one value it is less clear whether to block inheritance when you have more specific information. For example, if you know that a mammal *has part* head, and that an elephant *has part* trunk you may still want to infer that an elephant has a head. It is therefore useful to label slots according to whether they take *single values* or *multiple values*.

If objects/classes have several parent classes (e.g., Clyde is both an elephant and a circus-animal), then you may have to decide which parent to inherit from.

In general, both slots and slot values may themselves be frames. Allowing slots of be frames means that we can specify various attributes of a slot.

Advantages:

- Can cope with missing values - close matches are presented.

Disadvantages:

- Has been hard to implement, especially inheritance.

5.9 Production Systems

Production systems consist of a set of if-then rules, and a working memory. The working memory represents the facts that are currently believed to hold, while the if-then rules typically state that if certain conditions hold (e.g., certain facts are in the working memory), then some action should be taken (e.g., other facts should be added or deleted). If the only action allowed is to add a fact to working memory then rules may be essentially logical implications, but generally greater flexibility is allowed. Production rules capture (relatively) procedural knowledge in a simple, modular manner.

The architecture of a production system

A production system consists of three parts:

1. A short-term memory of facts (Working memory).

2. A rule base, which stores permanent (problem-independent) knowledge. It is composed of a set of production rules of the form "antecedent –> consequent".

3. An inference mechanism (interpreter) that represents a mechanism to examine the short-term memory and to determine which rules to fire.

4. Explanation Facility, the rule trace that can be used to generate explanations.

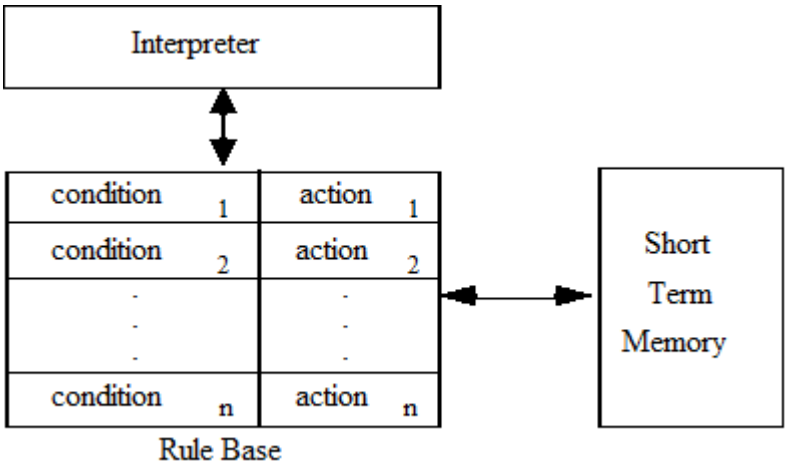


Figure 15: Architecture of a production system

Example of a production system

Short term memory: C5, C1, C3

Production rules:

C1 & C2 –> A1

C3 –> A2

C1 & C3 –> A3

C4 –> A4

C5 –> A5

Interpreter: Choose one rule from the applicable ones, according to some strategy.

The execution of a production system can be defined as a series of recognize-act cycles. It involves the following

1. Match: determines the rules that can be fired C3 –> A2, C1 & C3 –> A3, C5 –> A5. The set of rules that can be fired is called the *conflict set*.

2. Conflict resolution: select one rule from the conflict set C3 –> A2

3. Apply the rule: A2 executed Since the effect of executing the actions may change the short-term memory, different rules may fire on successive cycles.

Conflict Resolution Strategies

1. Rule Order i.e. Fire the first encountered rule that matches short term memory (STM)

2. Recency i.e. Fire rule that refers to element most recently added to STM

3. Specificity i.e. Fire most specific rule i.e. the one with most detailed part that matches current STM.

4. Priority i.e. Fire highest priority rule (assuming that each rule has an associated priority).

Advantages:

- Easy to use;
- Explanations are possible;
- Capture heuristics;
- Can handle uncertainties to some extent.

Disadvantages:

- Cannot cope with complex associated knowledge;
- They can grow to unmanageable size.

5.10 Inference

Inference is the process of drawing a conclusion from given evidence. It may also be seen as reaching a decision through reasoning. The program for inference is typically called the inference engine or control program. In rule based systems it is also called a rule interpreter.

5.10.1 Some inference strategies in Artificial Intelligence Applications

- Reasoning by analogy: the proponents claim that this is a natural way humans handle problems. The problematic situation is examined, then it is linked with other similar situations that are then used to build a solution.
- Formal reasoning: use rules, facts; predicate calculus; mathematical logic.
- Generalization and abstraction: use sets; induction and deduction.
- Procedural reasoning: use formulae and algorithms to solve problems.
- Meta-level reasoning: use knowledge about what is known to solve problems.
- Fuzzy logic: this is less precise and less logical method of reasoning.
  - Fuzzy sets; sets whose membership is probabilistic;
  - Fuzzy phenomena include: rain; heap of sand; harvest; height; size; beauty.

5.10.2 Reasoning Methods

Inference methods fall into the general categories

Deductive reasoning

- The method used in propositional and predicate calculus.
- The inference engine combines the rules or predicates to arrive at the final answer.
- It starts with a set of axioms – commonly accepted premises that one cannot derive from a system itself (basic facts in a domain used to prove theorems).
- Using axioms and proven formulae, deductive reasoning can deduce new conclusions.
- You thus build an axiomatic system, which can be used to infer new facts or prove that a certain fact is true

Process of Deductive Reasoning

- The inference process consists of two parts.
  - Single inference; The process of applying inference rules to combine two pieces of knowledge to derive new premises.
  - Multiple Inference; The sequence or order of applying the single inference process to the entire KB in order to derive final conclusions.
- Uses the inference rules stated earlier, which include modus pones, modus tollens, chair rule (Hypothetical syllogism).
- These must be a tautology to be true.

Multiple Inference in Deductive reasoning.

- Involves testing rules or predicates to find the one which must fire next.
- Most commonly used methods are:
  - Graphs, Trees and the And/or Graph e.g. in search
  - Backward chairing.
  - Forward chairing

Inductive reasoning

- Inductive reasoning is the process of going from specific cases to general rules e.g. the use of quantitative methods e.g. in statistics.
  - Most inference rules in A.I. have their theoretical basis in deductive reasoning while most learning is through experience, which is an inductive process.

5.10.3 Rule-Based Inference Control

There are two broad kinds of rule system: *forward chaining* systems, and *backward chaining* systems. In a forward chaining system you start with the initial facts, and keep using the rules to draw new conclusions (or take certain actions) given those facts. In a backward chaining system you start with some hypothesis (or goal) you are trying to prove, and keep looking for rules that would allow you to conclude that hypothesis, perhaps setting new sub-goals to prove as you go. Forward chaining systems are primarily data-driven, while backward chaining systems are goal-driven.

Forward Chaining Systems In a forward chaining system the facts in the system are represented in a *working memory* which is continually updated. Items in the system represent possible actions to take when specified conditions hold on items in the working memory - they are sometimes called condition-action rules. The conditions are usually *patterns* that must *match* items in the working memory, while the actions usually involve *adding* or *deleting* items from the working memory.

The interpreter controls the application of the rules, given the working memory, thus controlling the system's activity. It is based on a cycle of activity sometimes known as a *recognise-act* cycle. The system first checks to find all the rules whose conditions hold, given the current state of working memory. It then selects one and performs the actions in the action part of the rule. (The selection of a rule to fire is based on fixed strategies, known as *conflict resolution* strategies.) The actions will result in a new working memory, and the cycle begins again. This cycle will be repeated until either no rules fire, or some specified goal state is satisfied. Rule-based systems vary greatly in their details and syntax.

Forward Chaining Algorithm

1. Enter near data
2. Fire forward chaining rules
3. Rule actions infer new data values
4. Go to step 2
5. Repeat until no new data can be inferred.

6. If no solution, rule base is insufficient

Example

R1: IF (lecturing X) AND (marking-practicals X) THEN ADD (overworked X)
R2: IF (month february) THEN ADD (lecturing mwalimu)
R3: IF (month february) THEN ADD (marking-practicals mwalimu)
R4: IF (overworked X) OR (slept-badly X)THEN ADD (bad-mood X)
R5: IF (bad-mood X) THEN DELETE (happy X)
R6: IF (lecturing X) THEN DELETE (researching X)
Here we use capital letters to indicate variables.
Let us assume that initially we have a working memory with the following elements:
(month February)
(happy Mwalimu)
(researching Mwalimu)

Our system will first go through all the rules checking which ones apply given the current working memory. Rules 2 and 3 both apply, so the system has to choose between them, using its conflict resolution strategies. Let us say that rule 2 is chosen. So, (lecturing Mwalimu) is added to the working memory, which is now:
(lecturing Mwalimu)
(month February)
(happy Mwalimu)
(researching Mwalimu)

Now the cycle begins again. This time rule 3 and rule 6 have their preconditions satisfied. Lets say rule 3 is chosen and fires, so (marking-practicals Mwalimu) is added to the working memory. On the third cycle rule 1 fires, so, with X bound to Mwalimu, (overworked Mwalimu) is added to working memory which is now:
(overworked Mwalimu)
(marking-practicals Mwalimu)
(lecturing Mwalimu)
(month February)
(happy Mwalimu)
(researching Alison)

Now rules 4 and 6 can apply. Suppose rule 4 fires, and (bad-mood Mwalimu) is added to the working memory. And in the next cycle rule 5 is chosen and fires, with (happy Mwalimu) removed from the working memory. Finally, rule 6 will fire, and (researching Mwalimu) will be removed from working memory, to leave:
(bad-mood Mwalimu)
(overworked Mwalimu)
(marking-practicals Mwalimu)
(lecturing Mwalimu)
(month February)

The order that rules fire may be crucial, especially when rules may result in items being deleted from working memory (*nonmonotonic*) .

Anyway, suppose we have the following further rule in the rule set:

7. IF (happy X) THEN (gives-high-marks X)

If this rule fires BEFORE (happy Mwalimu) is removed from working memory then the system will conclude that I'll give high marks. However, if rule 5 fires first then rule 7 will no longer apply. Of course, if we fire rule 7 and then later remove its preconditions, then it would be nice if its conclusions could then be automatically removed from working memory. Special systems called *truth maintenance systems* have been developed to allow this.

A number of conflict resolution strategies are typically used to decide which rule to fire. These include:

1. Don't fire a rule twice on the same data. (We don't want to keep on adding (lecturing Mwalimu) to working memory).

2. Fire rules on more recent working memory elements before older ones. This allows the system to follow through a single chain of reasoning, rather than keeping on drawing new conclusions from old data.

3. Fire rules with more specific preconditions before ones with more general preconditions. This allows us to deal with non-standard cases. If, for example, we have a rule "IF (bird X) THEN ADD (flies X)" and another rule "IF (bird X) AND (penguin X) THEN ADD (swims X)" and a penguin called tweety, then we would fire the second rule first and start to draw conclusions from the fact that tweety swims.

These strategies may help in getting reasonable behavior from a forward chaining system, but the most important thing is how we write the rules. They should be carefully constructed, with the preconditions specifying as precisely as possible when different rules should fire. Otherwise we will have little idea or control of what will happen. Sometimes special working memory elements are used to help to control the behavior of the system.

Backward Chaining Systems It uses IF THEN rules to repetitively break a goal into smaller sub-goals, which are easier to prove. Given a goal state to try and prove (e.g., (*bad-mood Alison*)) the system will first check to see if the goal matches the initial facts given. If it does, then that goal succeeds. If it doesn't the system will look for rules whose conclusions (previously referred to as *actions*) match the goal. One such rule will be chosen, and the system will then try to prove any facts in the preconditions of the rule using the same procedure, setting these as new goals to prove. Note that a backward chaining system does ***NOT*** need to update a working memory. Instead it needs to keep track of what goals it needs to prove to prove its main hypothesis.

In backward chaining we are concerned with matching the conclusion of a rule against some goal that we are trying to prove. So the 'then' part of the rule is usually not expressed as an action to take (e.g., add/delete), but as a state which will be true if the premises are true.

Backward Chaining Algorithm.

1. State a specific goal (question)
2. Find rules which resolve the goal.
3. At mother, answer questions to satisfy the antecedent of the rules as required.
4. Obtain a result (goal resolved or not)

Example: Consider the following rules:
R1: IF (lecturing X) AND (marking-practicals X) THEN (overworked X)
R2: IF (month february) THEN (lecturing mwalimu)
R3: IF (month february) THEN (marking-practicals mwalimu)
R4: IF (overworked X) THEN (bad-mood X)
R5: IF (slept-badly X) THEN (bad-mood X)
R6: IF (month february)THEN (weather cold)
R7: IF (year 1993) THEN (economy bad)
and initial facts:
(month february) (year 1993)
and we're trying to prove:
(bad-mood mwalimu)

First we check whether the goal state is in the initial facts. As it isn't there, we try matching it against the conclusions of the rules. It matches rules 4 and 5. Let us assume that rule 4 is chosen first - it will try to prove (overworked mwalimu). Rule 1 can be used, and the system will try to prove (lecturing Mwalimu) and (marking practicals Mwalimu). Trying to prove the first goal, it will match rule 2 and try to prove (month February). This is in the set of initial facts.

We still have to prove (marking-practicals mwalimu).

Rule 3 can be used, and we have proved the original goal (bad-mood Mwalimu).

One way of implementing this basic mechanism is to use a stack of goals still to satisfy. You should repeatedly pop a goal of the stack, and try and prove it. If its in the set of initial facts then its proved. If it matches a rule which has a set of preconditions then the goals in the precondition are pushed onto the stack.

5.11 Knowledge Acquisition

Knowledge acquisition is the process of extracting knowledge (facts, procedures, rules) from human experts, books, documents, sensors or computer files and converting it into a form that can be stored and manipulated by the computer for purposes of problem solving. It occurs throughout the entire development process.

5.11.1 The Knowledge Acquisition Process

1. Identification: Identify the problem including data, criteria for solutions to meet, available resources, etc.
2. Conceptualization: Determine the key concepts and relationships by characterizing the data, flow of information, the domain structure, etc.
3. Formalization: Understand the underlying search space, uncertainty issues, etc.
4. Implementation: Translate acquired knowledge into the program.
5. Testing: Validate and verify.

Knowledge Elicitation It implies that knowledge acquisition is accomplished from a human expert.

5.11.2 Methods of Knowledge Elicitation

1. **Face to face interview with experts** – The experts are interviewed by knowledge engineers.
2. **Protocol analysis** – This is a documentation of how the expert behaves and processes information during problem solving. Usually the experts think aloud.
3. **Observation** – The experts are observed at work.
4. **Questionnaires** –This is where questions are sent to experts for responses.
5. **Analysis of documented knowledge** – It involves extraction of knowledge from sources such as books, journals, act, and constraints mass media materials.
6. **Rule induction (computer aided knowledge acquisition)** – rule induction can be viewed as a system that accepts examples and develops classification rules.

5.11.3 Issues With Knowledge Acquisition

1. Machine representation is lower in form than human usage of knowledge;
2. Many participants are involved and they have varied backgrounds causing communication challenges (Domain Experts, System designers, Users, etc.).
3. Experts may not express their knowledge;
4. Mismatch between the way experts hold their knowledge and the way computers represent knowledge.
5. What is knowledge acquisition?

6. Describe the steps in knowledge acquisition.
7. Explain how you may acquire knowledge to build a system.
8. Discuss ways of acquiring knowledge.
9. Discuss problems associated with knowledge acquisition.

5.12 Logic Programming

PROLOG (PROgramming in LOGic)

- A declarative programming language (Imperative languages like Basic, Pascal, in
- Based on the predicate calculus (first-order logic)
- Developed in about 1970 by Alain Colmerauer
- Uses resolution, a general rule of inference
- Brings logic into computer programs
- Express specifications for problem solving in formal logic
  - relations
  - logical variables

Declarative Programming Style

- Problems expressed in terms of high level descriptions, and not as a set of instructions for performing an algorithm.
- Emphasis on "what is true", "what needs to be done" rather than "how to do".

5.12.1 The Logic Paradigm

- A logic program comprises
  - collection of axioms (facts and inference rules)
  - one or more goal statements

5.12.2 Horn Clauses

- Prolog uses Horn clauses for explicit definition (facts) and for rules
- A program is a database of (Horn) clauses
- Rules are usually in the form of Horn clauses
  - :- is $\Rightarrow$ reversed
- Axioms, rules are written is standard form Horn clauses
  - a consequent (head H) and a body (terms Bi)

  H $\leftarrow$ B1, B2,..., Bn

  - when all Bi are true, we can deduce that H is true
- Horn clauses can capture most first-order predicate calculus statements but not all
- Backtracking in Prolog is known in FOL terms as *backward chaining*
- Prolog programs begins with a goal (the query), then recursively builds a set of substitutions that satisfy the goals necessary to conclude the goal
- All variables in Prolog are assumed to be *universally quantified*

5.12.3 Logic Programs

- A logic program is a set of predicates:
  - a predicate is made of facts and rules
  - The program is used to answer user queries.

5.12.4 Programming In Prolog

- Creates logical models that describe the world in which a problem exists
- Involves declaring facts and rules about objects and their relationships (predicates) by placing them in a computer file (the knowledge base)
- Then asking questions of the knowledge base (i.e. asking PROLOG to satisfy goals)
- PROLOG uses its own built in reasoning mechanism to do this (Backward chaining rule-based systems)
- Programming involves two phases: declaration and interpretation

Step 1: Declaration

- Creating a knowledge base (KB) of sentences describing the world
  - Declaring facts
    - fun(ai). (AI is fun!)
    - likes(reagan,bacon). (Reagan likes bacon)
- Defining rules
  - heartbroken(X) :- loves(X,Y), not(loves(Y,X)). (X is heartbroken if X loves Y and Y does not love X)
- Sentences end with a period

Example 1: A simple Knowledge Base

```
orbits(mercury, sun).
orbits(venus, sun).
orbits(earth, sun).
orbits(mars, sun).
orbits(moon, earth).
orbits(phobos, mars).
orbits(deimos, mars).
planet(P) :- orbits(P,sun).
satellite(S) :- orbits(S,P), planet(P).
```

Step 2: Interpretation

- Deriving new sentences from the sentences in the KB by making queries using a Prolog interpreter
- SWI-Prolog, GNU Prolog, etc.
  - Knowledge bases must first be loaded/consulted using consult('filename.pl').

Some simple queries

- ?- consult('solar.pl').
- % Is the moon a satellite?
  - ?- satellite(moon).
- % Is the sun a planet?

  - ?- planet(sun).
- % Is Uranus a planet?
  - ?- planet(uranus).
- % What are the planets?
  - ?- planet(Planet).
- % What objects orbit Mars?
  - ?- orbits(X, mars).F
- % Is the moon made of cheese?
  - ?- made_of_cheese(moon).

5.12.5 What Prolog is good for

- Knowledge representation
- Natural language processing
- State-space searching
- Logic problems
- Theorem provers
- Expert systems, deductive databases
- Agents

5.12.6 Language Elements

- Terms
- Facts and Rules
  - they use terms as data structures
- Predicates
  - made up of facts and rules
- The (Logic) Program
  - made up of predicates

Terms Three kinds of terms:

- Constants: integers, real numbers, atoms
- Variables
- Compound terms

Constants Constants can either be:

- Numbers
- Symbolic (non-numeric) constants/ Atoms:

Note: An atom is not a variable; it is not bound to anything, never equal to anything else

Variables

- Can be used to stand for any object
- Any name beginning with an uppercase letter or an underscore, followed by any number of additional letters, digits or underscores: X, Child, Fred, _, _123
- Prolog variables are logic variables, not containers to store values in.
- Use the underscore to indicate an anonymous variable
  - female(X) :- mother(X,_). (All mothers are female, regardless of who the child is.)

Naming tips

- Use real English when naming predicates, constants, and variables.
  - e.g. "John wants to help Somebody."
  - Could be: wants(john,to_help,Somebody).
- Use a Verb Subject Object structure: wants(john,to_help).

Compound Terms

- An atom followed by a parenthesized, comma-separated list of one or more terms: x(y,z), +(1,2), .(1,[]), parent(adam,seth), x(Y,x(Y,Z))
- Think of them as structured data

Clauses

- Predicate definitions consist of clauses.
- An individual definition (whether it be a fact or rule).
- Facts and rules are expressed as clauses in Prolog.
  - plays(john,football).
- The interpretation is defined by the writer
  - e.g. mother(jane,alan). = Fact
  - parent(P1,P2):- mother(P1,P2). = Rule
- A clause consists of a head and sometimes a body.
  - Facts don't have a body because they are always true.
- Rules have a head and a body e.g head:-body.
- If the body can be shown to be true, the head is true.

5.12.7 Principle of Resolution

- Prolog execution is based on the principle of resolution
  - If C1 and C2 are Horn clauses and the head of C1 matches one of the terms in the body of C2, then we can replace the term in C2 with the body of C1
- When two terms match we say that they unify.
  - Their structures and arguments are compatible.

For example,

C1: likes(sam,Food) :- indian(Food), mild(Food).
C2: indian(dahl).
C3: mild(dahl).

- We can replace the first and the second terms in C1 by C2 and C3 using the principle of resolution (after instantiating variable Food to dahl)
- Therefore, likes(sam, dahl) can be proved

5.12.8 Unification

- Prolog associates (binds) variables and values using a process known as unification
- Pattern-matching using Prolog terms
- A substitution is an assignment of variables to values.
- Two terms unify if there is some way of binding (substituting) their variables that makes them identical
- For instance, parent(adam,Child) and parent(adam,seth) unify by binding the variable Child to the atom seth

5.12.9 The Prolog Database

- A Prolog language system maintains a collection of facts and rules of inference
- It is like an internal database
- A Prolog program is just a set of data for this database
- The simplest kind of thing in the database is a fact: a term followed by a period

Example 2: Consider the following Prolog Database parent(kim,holly).
parent(margaret,kim).
parent(margaret,kent).
parent(esther,margaret).
parent(herbert,margaret).
parent(herbert,jean).

- Save as Relations.pl
- A Prolog program of six facts, defining a predicate parent of arity 2
- We would naturally interpret these as facts about families: Kim is the parent of Holly and so on

SWI-Prolog

- We will use SWI-Prolog
- Prompting for a query with ?-
- Normally interactive: get query, print result, repeat

The consult Predicate **?-** consult(relations).

% relations compiled 0.00 sec, 0 bytes

Yes ?-

- Predefined predicate to read a program from a file into the database
- Knowledge must be first be loaded/consulted using consult('filename.pl') e.g. file relations (or relations.pl) contains our parent facts

Simple Queries

- Describe problems to be solved
- Consist of goals to be satisfied
  - father(X) :- male(X), parent(X,_).
  - Goals are male(X) and parent (X,_).
  - Goals are checked against facts in the KB
  - Rules are satisfied if all the goals in the "if" part (in the body, separated by commas) are satisfied
  - Variables can take on any value
- We can ask about facts directly or we can define rules that prove if a property or relationship holds given the facts currently in the database.

  ?- parent(margaret,kent).

Yes

?- parent(fred,pebbles).

No

?-

- A query asks the language system to prove something
- It consist of goals to be satisfied, which are checked against facts in the KB
- The answer will be Yes or No

Note:

- Press semi-colon to look for further matches
- Uses backtracking to satisfy goals in all possible ways ("brute-forcing" it)

Final Period ?- parent(margaret,kent)
| .
Yes
?-

- Queries can take multiple lines
- If you forget the final period, Prolog prompts for more input with |

Queries with variables ?- parent(P,jean).
P = herbert
Yes
?- parent(P,esther).
No

- Any term can appear as a query, including a term with variables
- The Prolog system shows the bindings necessary to prove the query

Flexibility

- Normally, variables can appear in any or all positions in a query:
  - parent(Parent,jean)
  - parent(esther,Child)
  - parent(Parent,Child)
  - parent(Person,Person)

Conjunctions ?- parent(margaret,X), parent(X,holly).S
X = kim
Yes

- S conjunctive query has a list of query terms separated by commas
- The Prolog system tries prove them all (using a single set of bindings)

Multiple Solutions ?- parent(margaret,Child).
Child = kim ;
Child = kent ;
No

- There might be more than one way to prove the query
- By typing ; rather than Enter, you ask the Prolog system to find more

?- parent(Parent,kim), parent(Grandparent,Parent).
Parent = margaret Grandparent = esther ;
Parent = margaret Grandparent = herbert ;
No
?- parent(esther,Child),
| parent(Child,Grandchild),
| parent(Grandchild,GreatGrandchild).
Child = margaret
Grandchild = kim
GreatGrandchild = holly
Yes

Example 4. Path Searching

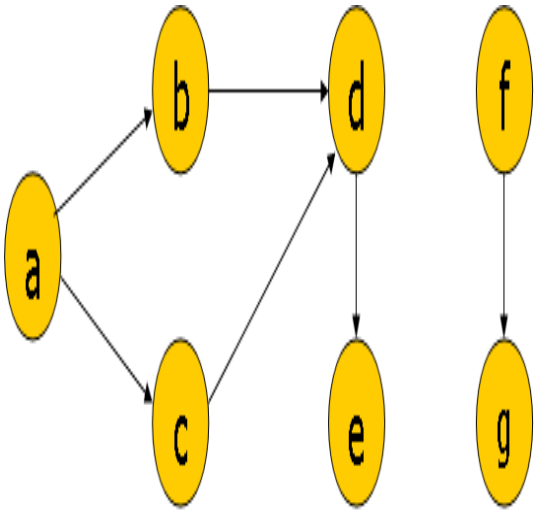


Figure 17: Path Searching

edge(a,b).
edge(a,c).
edge(b,d).
edge(c,d).
edge(d,e).
edge(f,g).
path(Node,Node).
path(Node1,Node3) :- edge(Node1,Node2), path(Node2,Node3).
Some Queries

- ?- path(a, e). – is there a path from a to e?
- ?- path(a, Y). – what node can be reached from a?
- ?- path(X, d). – what node has a path to d?

- ?- path(a,Y), write(Y), write("\n").
- ?- path(X,d), write(X), write("\n").

5.13 Reasoning under Uncertainty

So far, when we have assumed that if the preconditions of a rule hold, then the conclusion will certainly hold. In fact, most of our rules have looked pretty much like logical implications, and the ideas of forward and backward reasoning also apply to logic-based approaches to knowledge representation and inference.

Of course, in practice you rarely conclude things with absolute certainty. Usually we want to say things like "If Alison is tired then there's quite a good chance that she'll be in a bad mood". To allow for this sort of reasoning in rule-based systems we often add certainty values to a rule, and attach certainties to any new conclusions. We might conclude that Mwalimu is probably in a bad mood (maybe with certainty 0.6).

5.13.1 Sources of Uncertainty

- Occurs in various situations where the relevant information is deficient in some way
- Possible sources:
  - Partial information, information not reliable, representation language unreliable, information from multiple sources is conflicting, error, lack of confidence, imprecision, unreliability, variability, vagueness, ignorance, ambiguity.

5.13.2 Handling Uncertainty

There are various method of dealing with uncertain reasoning. These include:

- Uncertain answers
  - Allow Yes / No / Unknown answers
  - 'Unknown' is processed by triggering extra inference to determine answer when user cannot.
- Confidence factors
  - Confidence expressed as a number between 0 and 1
  - Allows uncertainty to be expressed in information and in reasoning
- Probabilities

  - Bayes' theorem allows the calculation of the probability of a hypothesis given any combination of symptoms (or events)
  - The raw data is easy to obtain though the symptoms must be independent of each other
  - Represents uncertainty by describing a model of the application domain as a set of possible outcomes known as the Hypothesis

- Fuzzy Logic

6 MACHINE LEARNING

6.1 Learning Outcomes

Upon successful completion of this unit, the student will be able to:

- Define machine learning, the basic concepts and methods used in machine learning.
- Explain the differences among the three main styles of learning (supervised, reinforcement, and unsupervised) and indicate when each is relevant
- Identify the various applications of machine learning.
- Identify the benefits and drawbacks of each type of machine learning.

6.2 Learning

- "Learning denotes changes in a system that enables a system to do the same task more efficiently the next time." –Herbert Simon
- "Learning is constructing or modifying representations of what is being experienced." Ryszard Michalski
- "Learning is making useful changes in our minds." Marvin Minsky

6.2.1 Why learn?

- Understand and improve efficiency of human learning
  - Use to improve methods for teaching and tutoring people (e.g., better computer-aided instruction)
- Discover new things or structure that were previously unknown to humans
  - Examples: data mining, scientific discovery
- Fill in skeletal or incomplete specifications about a domain
  - Large, complex AI systems cannot be completely derived by hand and require dynamic updating to incorporate new information.
  - Learning new characteristics expands the domain or expertise and lessens the "brittleness" of the system
- Build software agents that can adapt to their users or to other software agents
- Learning is used when:

- Human expertise does not exist (navigating on Mars),
- Humans are unable to explain their expertise (speech recognition)
- Solution changes in time (routing on a computer network)
- Solution needs to be adapted to particular cases (user biometrics)

6.2.2 Learning and Adaptation

- "Modification of a behavioral tendency by expertise." (Webster 1984)
- "A learning machine, broadly defined is any device whose actions are influenced by past experiences." (Nilsson 1965)
- "Any change in a system that allows it to perform better the second time on repetition of the same task or on another task drawn from the same population." (Simon 1983)
- "An improvement in information processing ability that results from information processing activity." (Tanimoto 1990)

6.3 Machine Learning

- It involves automatic procedures that learn a task from a series of examples
- Methods that teach machines to solve problems or to support problem solving, by applying historical cases
- Most convenient source of examples is data

Definition: A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience.

Examples:
Learn to play checkers

- T: play checkers & win
- P: % games won in tournament
- E: play against self Character recognition
- T: recognize & classify hand-written characters
- P: % of characters correctly classified
- E: a database of hand-written characters with given classifications

6.3.1 Learning as Related to AI

- Learning systems demonstrate interesting learning behaviors
- No claims about learning as well as humans or in the same way
- Learning systems are not defined very formally; implications are not well understood
- Learning in AI involves the manipulation of symbols (not numeric information)

A general model of learning agents

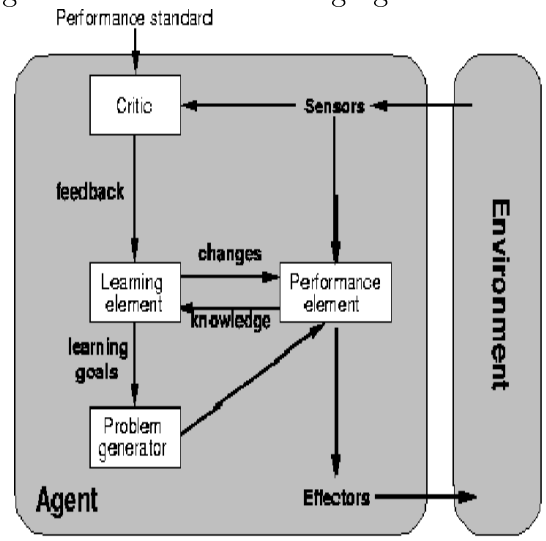


Figure 18:Model of learning agents

- Learning Element makes changes to the system based on how it's doing
- Performance Element is the agent itself that acts in the world
- Critic tells the Learning Element how it is doing (e.g., success or failure) by comparing with a fixed standard of performance
- Problem Generator suggests "problems" or actions that will generate new examples or experiences that will aid in training the system further

6.3.2 Evaluating Performance

- Several possible criteria for evaluating a learning algorithm:
  - Predictive accuracy of classifier
  - Speed of learner
  - Speed of classifier
  - Space requirements

Most common criterion is predictive accuracy

6.4 Designing a Learning System

- Choosing the training experience:
  - Direct or indirect feedback
  - Degree of learner's control
  - Representative distribution of examples
- Choosing the target function:
  - Type of knowledge to be learned
  - Function approximation
- Choosing a representation for the target function:
  - Expressive representation for a close function approximation
  - Simple representation for simple training data and learning algorithms
  - Choosing a function approximation algorithm (learning algorithm)

Example: Chess game:

- Task T: playing chess games
- Performance measure P: percent of games won against opponents
- Training experience E: playing practice games against itself
- Target function: V: Board $\rightarrow$R

Designing a Learning System: Chess game

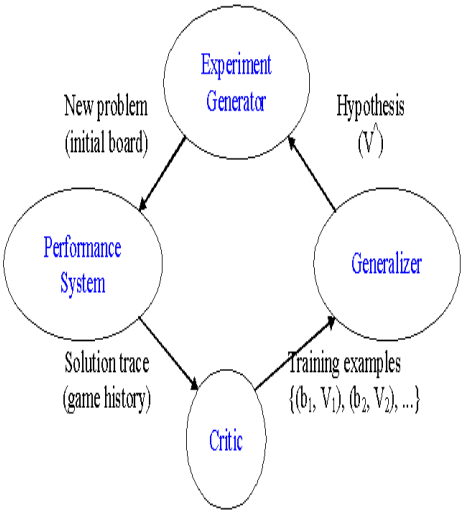


Figure 19: Designing a Learning System

6.5 Major paradigms of machine learning

- Rote learning
  - One-to-one mapping from inputs to stored representation.
  - "Learning by memorization." Association-based storage and retrieval.
- Induction
  - Use specific examples to reach general conclusions
- Clustering
- Unsupervised identification of natural groups in data
- Analogy
  - Determine correspondence between two different representations
- Discovery
  - Unsupervised, specific goal not given
- Genetic algorithms
  - "Evolutionary" search techniques, based on an analogy to "survival of the fittest"
- Reinforcement
  - Feedback (positive or negative reward) given at the end of a sequence of steps

6.5.1 Issues in Machine Learning

- What learning algorithms to be used?
- How much training data is sufficient?
- When and how prior knowledge can guide the learning process?
- What is the best strategy for choosing a next training experience?
- What is the best way to reduce the learning task to one or more function approximation problems?
- How can the learner automatically alter its representation to improve its learning ability?

6.6 Applications

1. Association
2. Supervised Learning
   (a) Classification
   (b) Regression
3. Unsupervised Learning
4. Reinforcement Learning

6.6.1 Learning Associations

- Basket analysis:
  - $P(Y|X)$probability that somebody who buys X also buys Y where X and Y are products/services.

  Example: P (chips | beer) = 0.7

6.6.2 The inductive learning problem

- Extrapolate from a given set of examples to make accurate predictions about future examples
- Supervised versus unsupervised learning
  - Learn an unknown function f(X) = Y, where X is an input example and Y is the desired output.
  - Supervised learning implies we are given a training set of (X, Y) pairs by a "teacher"
  - Unsupervised learning means we are only given the Xs and some (ultimate) feedback function on our performance.
- Concept learning or classification
  - Given a set of examples of some concept/class/category, determine if a given example is an instance of the concept or not
  - If it is an instance, we call it a positive example
  - If it is not, it is called a negative example
  - Or we can make a probabilistic prediction (e.g., using a Bayes net)

6.6.3 Supervised concept learning

- Given a training set of positive and negative examples of a concept
- Construct a description that will accurately classify whether future examples are positive or negative
- That is, learn some good estimate of function f given a training set

  $\{(x1, y1), (x2, y2), ..., (xn, yn)\}$where each yi is either + (positive) or - (negative), or a probability distribution over +/-

Supervised Learning Applications

Given: Training examples (x; f(x)) for some unknown function f, find a good approximation to f.

- Handwriting Recognition
  - x: Data from pen motion.
  - f(x): Letter of the alphabet.
- Disease diagnosis
  - x: Properties of patient (symptoms, lab tests)
  - f(x): Disease (or maybe, recommended therapy)
- Face recognition
  - x: Bitmap picture of person's face
  - f(x): Name of the person.
- Spam Detection
  - x: Email message
  - f(x): Spam or not spam.

6.6.4 Classification

- Example: Credit scoring
- Differentiating between low-risk and high-risk customers from their income and savings

Applications of classification

- Face recognition: Pose, lighting, occlusion (glasses, beard), make-up, hair style
- Character recognition: Different handwriting styles.
- Speech recognition: Temporal dependency.
  - Use of a dictionary or the syntax of the language.
  - Sensor fusion: Combine multiple modalities; eg, visual (lip image) and acoustic for speech
- Medical diagnosis: From symptoms to illnesses

6.6.5 Unsupervised Learning

We are interested in capturing inherent organization in the data

- clustering,
  - density estimation
- No feedback goal is to group data into similar groups
- No output
- Unsupervised learning means we are only given the Xs and some (ultimate) feedback function on our performance.

Example applications

- Customer segmentation in Customer Relationship Management (CRM)
- Image compression: Color quantization
- Bioinformatics: Learning motifs

6.6.6 Reinforcement Learning

- We only get feedback in the form of how well we are doing (not what we should be doing)
- Feedback (positive or negative reward) given at the end of a sequence of steps/actions/decisions
- No supervised output but delayed reward

Example Applications

- credit risk assessment
  - x: properties of customer and proposed purchase
  - f(x): approve purchase or not
- Forensic hair comparison
  - x: Bitmap of hair image
  - f(x): match or not
- Stock market prediction
  - x: economic indicators (S&P, Dow Jones, interest rates, . . .)
  - f(x): market will go up or down
- Steering a vehicle
  - x: bitmap of road surface in front of car
  - f(x): degrees to turn steering wheel
- Others
  - Game playing
  - Robot in a maze
  - Multiple agents, partial observability, ...

6.7 Machine Learning Methods

- Artificial Neural Networks
- Decision Trees
- Instance Based Methods (Case Based Reasoning (CBR), K-Nearest Neighbor (k-NN))
- Bayesian Networks
- Evolutionary Strategies
- Support Vector Machines

6.8 Some issues in Machine Learning

- What hypothesis spaces work well?
- For these hypothesis spaces, what algorithms work well?
- How can we optimize accuracy for unseen data points?
- How does the number of training examples influence accuracy?
- What affect does noise in data have on performance?
- What are theoretical limits of learnability?
- How can prior knowledge help?
- How can systems alter their own representations?
- How can we apply machine learning in practice?

7 ROBOTICS

7.1 Learning Outcomes

Upon successful completion of this unit, the student will be able to:

- Give an overview of the history of robotics.
- Identify the challenges, scope of the problems, and progress in robotics.
- Describe some of the AI approaches, as well as approaches from other disciplines, used in robotics applications.

7.2 Definition

A robot is a programmable multifunction manipulator designed to move material parts, tools, or specific devices through variable programmable motions for the performance of a variety of tasks. Autonomous robots are those robots that make decisions on their own, guided by feedback from physical sensors.

7.3 Suitable uses of robots

1. Manufacturing and handling materials: Robots can be used for repetitive manufacturing tasks.
2. Moving items: Mobile robots can be used to distribute mails; or be used as moving vehicles such as underwater vehicles:
3. Hazardous environments: Robots can be used in dangerous situations such as nuclear disaster zones, lunar exploration, nuclear plant maintenance, toxic waste clean up, deep sea exploration.
4. Tele-presence and virtual reality: Robots can be used to monitor things going on in distant places or simulate reality through computer controls (virtual reality).
5. Augment human abilities: Robots can be used to duplicate lost limbs; they may also be used as blind guides.

7.4 Components of Robots

Robots consist of several parts, which are discussed below:

1. Effectors: These are tools for action usually by which the robot affects the environment under its control.

2. Actuator: This is a part of an effector that converts software commands to physical motion such as motors, hydraulic or pneumatic cylinders. They will determine the degree of freedom.

3. Locomotion: Locomotion involves the change of position of a robot within its environment using effectors.

4. Manipulation: Manipulation involves moving other objects in the environment using effectors.