(Copia modificata per una migliore consultazione on-line)

Artificial Intelligence Applied to Design of Intelligent Systems (a Soft Computing Approach)

Litvintseva L.V. and Ulyanov S.V.
(Scientific Supervisor: Ph.Doctor in Mechanics and Control Engineering, State Doctor in Physics and Mathematics, Professor S.V. Ulyanov)
Universita Degli Studi di Milano,
Polo Didattico e di Ricerca di Crema, Via Bramante, 65-26013 CREMA ASS013

Content

Part 1: Main Capabilities, General Structure and Properties of Intelligent Systems.
Knowledge Engineering and Expert Systems
Lecture 1. What is Artificial Intelligence?
Lecture 2. Expert Systems and Knowledge Engineering Problems
Lecture 3. Knowledge Processing. Application Examples
Part 2: Soft Computing as Flexible Tools for Knowledge Representation, Processing and Acquisition
2.1 *Human-like Decision Making based on Fuzzy Logic*
Lecture 4. Human Decision Making and Fuzzy Sets. Application Examples
Lecture 5 Fuzzy Numbers, Fuzzy Arithmetic and Fuzzy Relations. Application Examples
Lecture 6. Fuzzy Logic. Fuzzy Reasoning. Application Examples
Lecture 7. From Fuzzy Logic to Fuzzy System
Lecture 8. From Fuzzy Systems to Fuzzy Control. Examples of Applications
2.2 Genetic Algorithm as Nature-like Optimization Tool
Lecture 9. Genetic Algorithms: Theoretical Background
Lecture 10. GA Application to Intelligent Control Systems Design
2.3 *High parallel Implementation of Fuzzy Control based on Neural Networks*
Lecture 11. Artificial Neural Networks: Background and Application to Intelligent Control
Lecture 12. Learning in ANN. Fuzzy Neural Networks and GA-based FNN Tuning
2.4 *Soft Computing Application Benchmarks*
Lecture 13. Example: Intelligent Robust Control of Extension-Cableless Robotic Unicycle
References
Part 3: Symbolic AI for High-level Intelligent Systems Design
3.1 External World Modeling
Lecture 14. Temporal Knowledge Representation in Intelligent Systems
Lecture 15. Spatial Knowledge Representation in Intelligent Systems
Lecture 16. Actions Representation Models
3.2 Goal-oriented Behavior Modeling
Lecture 17. Goal-oriented Behavior and Planning Problems
Example: Behavior Simulation of Intelligent Service Robot
3.3 Natural Human-Computer Interaction Modeling
Lecture 18. Intelligent Human-Computer Interaction. Natural Language Processing
Lecture 19. Virtual Reality Systems and Intelligent Interaction Agents – New Forms of Human-Computer Interaction
Lecture 20. Problems of Computer's Creation Skills Simulation
Postscriptum
References

Introduction

In these Lectures Notes we will study Artificial Intelligence (AI) applications to design of intelligent systems. In particular, we will study AI methods for intelligent capabilities design of control systems based on soft computing and fuzzy logic. Fuzzy logic is 35 years old and has been developed largerly outside mainstream AI. As a consequence, fuzzy logic methods are not always well understood in AI, although an important part of fuzzy logic research has concentrated on issues such as knowledge representation, approximate reasoning and reasoning under uncertainty, which are central in AI. These Lectures Notes intend to bridge the gap between mainstream AI, fuzzy set and soft computing tools in general form, and to provide an organized view of works by gathering representative applications of soft computing theory-based methods to AI.

AI methods are traditionally concentrated in areas such as :

- Reasoning under uncertainty;
- Nonmonotonic reasoning and inconsistency handling;
- Case-based reasoning;
- Qualitative reasoning;
- Diagnostic;
- Soft constraint satisfication problems;
- Preference modelling and decision;
- Planning;
- Learning;
- Data fusion;
- Advanced information systems;
- Computer vision;
- Multiple agents systems, etc.

Many of these problems are discussed in these Lectures Notes. Here we would like to described briefly particularities of these Lectures Notes.

In first part we introduce two levels of intelligence in an intelligent behavior control system. Modeling, Reasoning, Planing, Decision-Making, Learning, and Language-based Communication capabilities represent the high level of intelligent behavior, while Perception, Adaptation,Task execution and Motion Control represent the low level of intelligent behavior.

Low level intelligence-based behavior appears in solving of different sensor-motor tasks (including intelligent control of motion, recognition, and adaptation). For modeling low level intelligent behavior we will apply soft computing methodology. This methodology and its application to intelligent control will be considered in the second part of our Lectures Notes.

High level intelligence-based behavior appears in solving of communication tasks, in thinking (including decision-making, reasoning and generalization), in explanation, learning, cognition and memorizing.

We believe that namely the usage of language distinguishes the high level of intelligence from another levels of intelligent behavior, and we affirm that a high level intelligent behavior is based on a language. Therefore for modeling high level intelligent behavior we will apply different approaches of symbolic paradigm of AI. Symbolic AI methodology and its application to intelligent control will be considered in the first and third parts of the Lectures Notes.

There are two main approaches to design intelligent systems. First approach is manual or knowledge-based approach, when intelligent systems are directly

programmed to perform given tasks. In a second approach, which is a learning-based approach, a system is evolved over time controlled by a task-specific learning program. Both two approaches we will consider and discuss.
These lectures are the result of authors research and teaching experience accumulated during giving lectures in Moscow University of Radio-Electronic and Automatics (Russia), in the University of Electro-Communications (Tokyo, Japan), in Montana State University (USA) and in Milan State University (Polo Didattico e di Ricerca di Crema, Italy).

P a r t 1 Main Capabilities, General Structure and Properties of Intelligent Systems. Knowledge Engineering and Expert Systems

Lecture N 1 What is Artificial Intelligence?

The progress in computers led to a new branch of computer science envisioned by
Alan Turing in his paper " Computing Machinery and Intelligence" (1950) [1]. This new
branch was called later as Artificial Intelligence (AI) .

The field of AI consists of models, methods and technologies for the following *scientific and practical aims*:

- constructing intelligent machines, whether or not these operate in the same way as people do;
- formalizing knowledge and reasoning, both common sense and refined expertise, in all areas of human activity;
- using computational models for understanding the psychology and behavior of people, animals, and artificial agents ;
- making a work with computers as easy and helpful as working with skilled and expert people.

The aim of AI reflects ancient dreams of people to create artificial beings like ourselves.
In past, pursuit of these dreams gave rise to mathematical automata, formal theories of reasoning, logic. Then investigation of intellectual abilities of human (for example, playing good chess, driving a car, or creation of new music, poems, and so on) are started, and Artificial Intelligence research was initiated.

AI researches goal is to understand the principles and mechanisms that account for intelligent action (behavior).

AI engineering goal is to design intelligent agents that can survive and operate in the physical (or artificial) world and solve problems of considerable scientific difficulty at high levels of competence.

AI was founded on the premise that all cognitive activity can be explained in terms of computation.

AI is primarily an empirical science, in which researches use the classical hypothesis-and-test research paradigm to validate models and mechanisms of intelligent action.
The computer is the laboratory where AI experiments are conducted.

A few definitions of AI

The term "artificial intelligence" has generated strong emotions among some researchers.
The following two questions are often debated:
1) May or not may computers have an intelligence?
2) Whether or not computer-based intelligence equal human intelligence?

If we started with an *operational definition* of artificial intelligence given in the Oxford dictionary of computing (1991) [2] computers have already exhibited intelligent behavior.
This definition is following:

AI is a discipline concerned with building of computer programs that perform tasks requiring intelligence when they done by humans.

So, a system that plays pretty good chess, keeps car on the rood, or diagnoses symptoms of a disease is exhibiting intelligent behavior under this definition.
Confusion arises when we attempt to equate human intelligence with artificial intelligence.

In fact, some properties of human intelligence may be exhibited in an AI system, and some properties might be impossible for AI system. Some skills may be performed better by human beings, another skills may be performed better by AI system.
Ultimately, *what AI system can do depend from the needs of society. In any case, it is clear that AI can enhance humans and can be a powerful tool for research and development.*
Many AI researches define AI in light of their own philosophy. Some representatives of AI definitions are listed below.
One of most widely accepted AI definition is as follows: *AI is a discipline concerned with making computers smart.*
Another definition, also widely accepted: *AI is a discipline concerned with making computer models of human intelligence*.
A third definition is: *AI is a discipline concerned with building machines that simulates human intelligence behavior.*
These three definitions of AI correspond to the three main approaches in AI:
1. *behavior-oriented approach* which attempts to program computers to behave in an intelligent, or «smart» way;
2. *cognitive approach* which tries to model human thought (thinking) processes in order to understand the human mind better;
3) *robotic approach* which attempts to build the machines like robots which can duplicate human intelligent behavior.

Why AI is needed?

Let us discuss important aspects of AI application significance.
1. A lot of modern industries in many countries (such as Japanese, USA, Germany, France, Italian industries,etc.) have achieved very high standards and quality in their products. *How to improve the quality of their products?*
In the near future will be only one alternative to do this - the use of AI technologies. Therefore many industial companies are open now to the use of AI technology in their manufacturing activities.
2. In XXI century a lot of countries are entering postindustrial information societies *where intelligent information system and intelligent robotic systems will play very important role.*
These systems will accommodate the demands of all of the people, and therefore they will be «people-friendly». How to do it?
They will need AI approaches to achieve a quality that satisfies all human demands.
3. AI applications will create new products that will also create new jobs.
This phenomenon would be very serious for the society of the future. We need to think how to raise the educational levels of people (perhaps, beginning from their young age) so that they can adopt intelligent systems for their individual purposes and life.

(Copia modificata per una migliore consultazione on-line)

All mentioned above are the reasons why we begin to study main ideas and techniques of AI.

Artificial Intelligence History

Historically AI originated from cybernetics – the theory of information and control in human beings and machines.
The term Artificial Intelligence was introduced in 1956 at a Darthmouth College conference by John McCarthy, Marvin Minsky, Allen Newell, and Herbert Simon. From this time AI is considered as a new discipline.
The history of AI and adjacent research areas are shown in Table 1-1.

Table 1-1.

	Conventional AI	Neural Networks	Fuzzy Systems	Other methodologies
1940s	1947 Cybernetics	1943 McCuloch-Pitts Neuron Model		
1950s	1956 Artificial Intelligence	1957 Perceptron		
1960s	1960 LISP programming language	1960 Adaline Madaline Neural Architectures	1965 Fuzzy Sets	
1970s	mid-1970s Knowledge Engineering (Expert Systems)	1974 Back Propagation Algorithm	1974 Fuzzy Controllers	1970 Genetic Algorithm
1980s	Expert Systems boom	1980 Hopfield Network 1983 Boltzman Machine 1986 Back-propagation algorithm boom	1985 Fuzzy Modeling (TSK Model)	mid-1980s Artificial Life Modeling Immune Modeling
1990s	Soft Computing boom Virtual reality Systems	Different Learning Algorithms Development	Neuro-Fuzzy Modeling	Genetic programming
2000s	Intelligent Agents Internet-based AI technologies		Neuro-Fuzzy-GA technologies	Evolution programming

In the adjacent research areas more attention has been directed toward biologically inspired models such as brain modeling, evolutionary algorithms, immune modeling. They simulate biological mechanisms responsible for generating natural intelligence.

Table 2 lists some tasks explored by AI researches. AI tasks are characterized by such attributes as knowledge content, knowledge and data processing rates, and response time.

Table 1-2. AI problem domains and their attributes.

Problem Domain	Knowledge Content	Data Rate	Response Time
Puzzles	Poor	Low	Hours
Chess	Medium	Low	Minutes
Theorem proving	Medium	Low	Variable
Expert Systems	Rich	Medium	Variable
Natural Language	Rich	Medium	Real time
Motor processes	Rich	High	Real time
Speech	Rich	High	Real time
Vision	Rich	Very high	Real time

AI Paradigms

1. Symbolic paradigm of AI

First AI research paradigm *focuses on an attempt to simulate human intelligent behavior by expressing it in language forms or symbolic rules*. Such AI systems are based on a knowledge base represented on some formal language. This is *symbolic AI* paradigm.
Remark: in some literature first AI research is called as Conventional AI.
Symbolic AI have been applied to a lot of tasks: natural language processing, machine learning, modeling cognitive processes, playing chess, automatic proving of theorem, etc.
But, the most successful symbolic AI product is expert systems widely used in practice in many areas.

In Fig.1-1 the structure of traditional expert system (ES) is shown. The main parts of ES are a Knowledge Base (KB) and an inference mechanism based on a given knowledge manipulation model.

Design of expert systems led to a special direction in AI called knowledge engineering. It represents the methods of knowledge acquisition, representation and processing.
Unfortunately, symbolic AI provides a good basis for modeling human experts only in some narrow problems area where explicit knowledge is available. The process of knowledge acquisition turned out to be much more difficult than initially expected. In some domains experts showed substantial problems in articulating their knowledge. All these difficulties supported the suggestion that the idea of symbol systems is not so good and appropriate for complex domains.
Symbolic AI based systems do not perform well in all cases when inexact, missing or uncertain information is used, or when parallel solutions need to be elaborated.
AI research field is steadily expanding. New methods of knowledge processing are developing, and new paradigms of AI are appeared.

(Copia modificata per una migliore consultazione on-line)

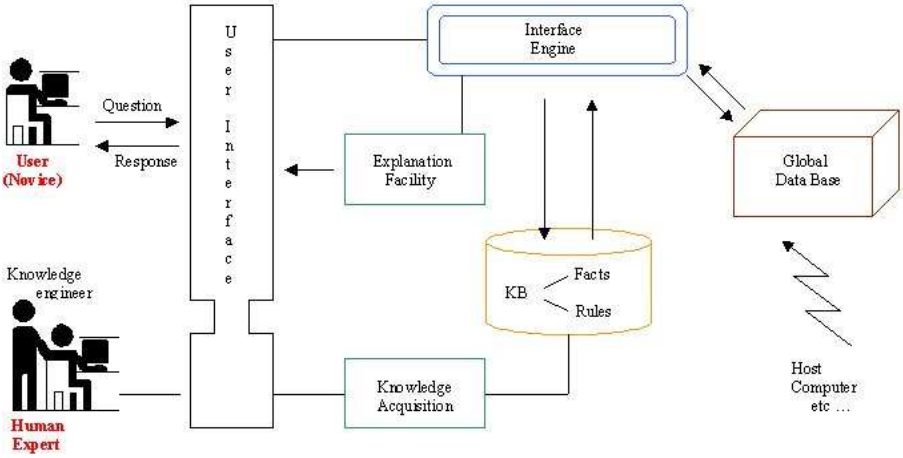
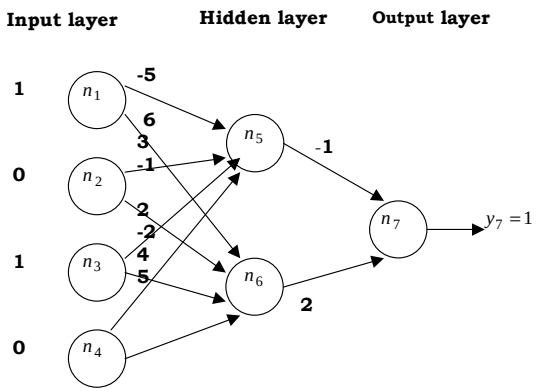


Fig.1-1. An Expert System Structure

2. Non-Symbolic paradigms of AI

2.1 Connectionist paradigm of AI

Connectionist models are usually considered as large networks of simple computing units which act in parallel. Example of connectionist architecture is shown in Fig. 1-2.



$$f(n_5) = 1(-5) + 0 \cdot 3 + 1 \cdot 2 + 0 \cdot 4 = -3 < 0 \rightarrow y_5 = output(n_5) = 0$$

Figure 1-2. Example of connectionist architecture.

Each computing unit carries a numerical activation value which is computed from the activation values of connected units in the network. The network's elements influence each other's values by connections of specific strengths. The strength of connection may change over time. The change of a connection may depend on particular training pattern. The network's overall function is strongly dependent on the currently present connection strengths or weights. Hence, the weights of a network are also encoding the system's knowledge. Many of the connectionist models have learning procedures for tuning their weights to implement a specific I/O-function for the overall network.

2.2 Subsymbolic paradigm of AI

At the heart of the connectionist paradigm lies the idea of subsymbol (Smolensky,1990 [3]). According to Smolensky, subsymbols, roughly speaking, are numerical values which cannot be interpreted in any sensible way individually. Only a large set of such subsymbols and only if they show certain patterns can be said to correspond to symbols. In other words, only certain patterns among a large number of subsymbolic values can be assigned any meaning. Hence, subsymbols differ in their interpretability substantially from the symbols of the symbolic paradigm, although numerical values may also appear in the symbolic paradigm.

Subsymbolic systems is based on a distributed (subsymbolic) representations described by some kind of connectionist architecture.

Non-symbolic AI systems are usually based on artificial neural networks models, and have been applied to tasks of self learning, pattern recognition, control, individual knowledge acquisition, skill and intuition behavior simulation.

3. From conventional AI to Intelligent Computation

Advanced AI lies in the integration of several adjacent research areas such as soft computing (including fuzzy models, neural networks and genetic algorithms), evolution modeling, cognitive science, and others (Table 1-1). Finally, we may consider the research paradigm of advanced AI as a *mixture of symbolic and subsymbolic* paradigms. We call it the *intelligent computation* paradigm.

Let us return to the AI research goal and discuss the principles and mechanisms of intelligent action (behavior). Fifty years of AI research allow us to understand the nature of intelligent action. The following five laws describe this nature [4].

Five Laws of Intelligent Behavior

1. Bounded rationality

The first law says that

"Computational constrains" on human thinking lead people to be satisfied with a "good enough" solution rather than waiting for the optimal solution.

Much of AI research is the study of approximate algorithms of optimal search.

(Copia modificata per una migliore consultazione on-line)

2. A physical symbol system hypothesis

The second law of intelligent behavior says that

A physical symbol system is necessary and sufficient for intelligent action.

The physical symbol system hypothesis (formulated by Allen Newell [5]) has the following properties:

- Physical symbols are symbols that are realizable by engineering components;
- A physical symbol system is a set of these entities;
- A symbolic structure is an expression whose components are symbols;
- Operations on the expression include creation, modification, reproduction, and destruction of symbols;
- Expressions can be interpreted as action.

Does the human brain have the mechanisms and properties of a physical symbol system?

We in AI believe it does, but we can not prove it. That is why this law is hypothesis.

3. The principle of rationality

This law says:

If an agent has knowledge that one of its actions lead to one of its goals, then the agent will select that action.

The given formulation of the principle of rationality poses the question of how to characterize more precisely the notion of *knowledge*. A knowledge level description of a system is assumed to be incomplete. In general, the knowledge level is not meant to provide a deterministic description of a system's (agent's) behavior. That is, sometimes, behavior of a system can be predicted by a knowledge level description, but sometimes the behavior cannot be predicted.

4. Search principle

This law tells us about the important role of searching processes:

Search compensates for a lack of knowledge

If we faced with a situation when we have no knowledge, we use trial-and-error behavior usually until a solution is found. This law lies in the ground of problem solving methods for such tasks as puzzles, games, self-organization, adaptation simulation, etc.

5. Knowledge principle:

Knowledge compensates for a lack of search

This law is opposite to the previous law. It means that the knowledge can reduce uncertainty and helps a system to decrease a search space and find an optimal solution (behavior) in real time.

The Structure and Properties of Intelligent Systems

Return now to the structure of traditional intelligent system shown above in Fig.1-1. You can see that it is the system capable to interact (on some language) with a human and solve a task given by the human by using knowledge from its knowledge base. Let us modify the structure of intelligent system by the way shown in Fig.1-3.

AI system is considered now as an *intelligent agent* (*system*) capable to *sensing* its environment and *acting* intelligently according to its perception.

Consider the following main properties of intelligent systems:

- *adaptive goal-oriented, or rational, behavior*;
- *reasoning and generalization*;
- *learning from experience*;
- *acquisition* and *use vast amounts of knowledge*;
- *self-awareness*;
- *interaction with human beings using natural language and speech*;
- *toleration to error and ambiguity in communication , and*
- *capability to respond in real time.*

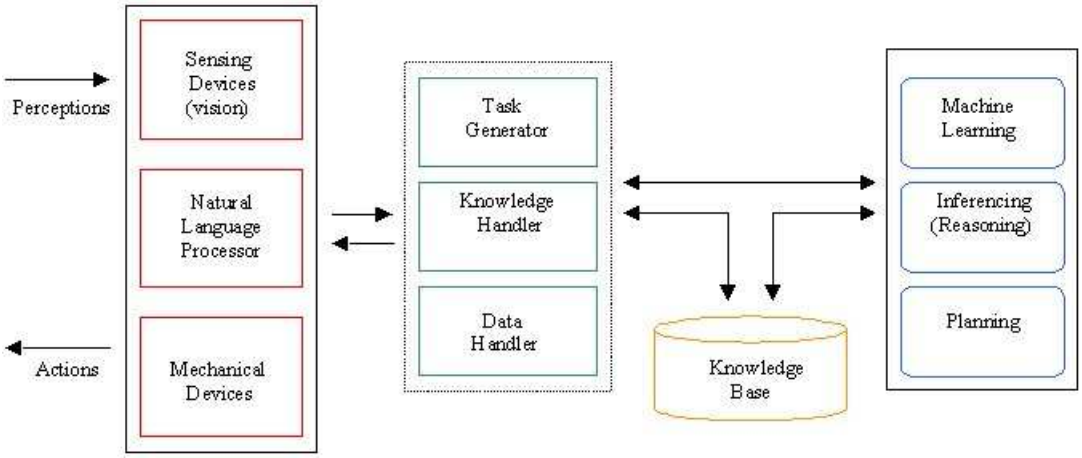


Fig.1-3. An Intelligent System Structure

Consider briefly these capabilities.

1) Goal -oriented behavior

(Copia modificata per una migliore consultazione on-line)

Suppose you want to create an agent that can make a telephone call. This task requires converting the goal into subgoals, such as creation of the phone directory, dialing the number, and talking with answering agent. Then we must translate the subgoals into a sequence of actions. To solve this problem an algorithm must create an agenda of goals and subgoals, use knowledge about operations and methods that can translate a desired goal into a sequence of actions.

2) Reasoning and Generalization

These capabilities allow to find new laws in the nature, for example, as Newton laws, or to construct new thoughts, ideas, associations, and so on.

3) Learning from experience

Learning means that the system is capable of acquiring new knowledge and using them to resolve ambiguity and adapt to new tasks and conditions. Consider, for example, the phone task "get me *X*". If *X* is ambiguous and a human being helps resolve the ambiguity, the system must remember (acquire) that knowledge, and in the next time must solve this task without a human help. Learning also implies that AI system has algorithms for automatically modifying structure and function of the system based on experience.

4) Use vast amounts of knowledge

Effective intelligent system design requires a very big knowledge base equivalent to human memory in solving similar problem. Using vast amounts of knowledge not only requires large memory capacity but creates the problem of selecting and applying the right knowledge for a given task and context. For example, someone ask a system " Is Mary staying in "Standa" right now?" A system with a large Data Base of facts must search millions bytes of data before to give output that it does not know an answer. So, algorithms that "know what they do not know" are needed. It is interesting area of research in AI.

5) Self awareness

AI system needs the capabilities to explain their behavior and to monitor, diagnose, and repair themselves in the presence of viruses. This requires internal mechanisms that can be called self-awareness.

6) Language and speech

Intelligent system will be required to interact with humans and other intelligent systems. In this case it must be capable of using language close to natural language (NL) and speech. Use of language and speech implies not only NL parsing and interpretation algorithms, but handling ambiguity and nongrammaticality as well.

7) Error and ambiguity

Human communication is necessarily imprecise and inaccurate. Thus, AI system must be able to tolerate error and ambiguity in communication. So, we must develop algorithms that can detect ambiguity and resolve it.

8) Real time

Intelligent systems such as autonomous mobile robots or intelligent communication agents will need to respond in real time. This implies the development of real time (on-line) algorithms for their behavior.

If we can create a system, which can duplicate mentioned above abilities, then we can consider the system as intelligent one. How to simulate all these abilities in AI system?

Some models of such a kind of abilities we will consider in our lectures.

Human Intelligence and Emotions

AI researches try to integrate results and ideas developed in cognitive science. Cognitive researches examined human brain structures and its functions and received a very important result that is:

Human intelligence can not be separated from human intuition, imagination, emotions, affects and will processes.

Researches have conducted a series of experiments, which have showed that left and right hemispheres of human brain are not symmetric, and have different functions.

Left part is responsible for textual data, speech, for a work with numbers and formulas, that is for symbolic information processing, while right part supplies imagination, intuition, dreams, associations, image processes and creativeness processes, etc. A human being which has destroyed relationships between left and right parts of brain has no intuition, can not see dreams at night, do not understand humor, and others.

One important feature of human intelligence is the ability to transfer image representation (non verbal information) to textual representation (verbal information) and vice versa.

How to simulate all mentioned above phenomenon in computer systems? Some models we will discuss in our lectures.

Main Directions of Advanced AI Research

Finally, we can summarized the main directions of advanced AI researches which are:

- *knowledge representation* seeks to discover expressive and efficient forms and methods for representing information about all aspects of the world;
- *learning and adaptation* seeks to discover techniques and mechanisms for this tasks;
- *deliberation, planning and acting* concern methods for making decision, constructing plan and achieve specified goals;
- *speech and language processing* seeks to create systems capable of communicating;
- *image understanding and synthesis* seeks algorithms for analyzing visual data;
- *manipulation and locomotion* seeks to replicate the abilities of natural hands, arms, feet and bodies;
- *autonomous agents and robots*, which integrates to the other areas to create robust, active entities capable of independent intelligent real time interactions with the environment;
- *multiagent systems* seeks the tools and methods for cooperative working of several agents;
- *cognitive modeling* seeks methods for human cognition simulation;
- *mathematical foundations* takes the concepts and techniques for formalizing of mentioned above researches.

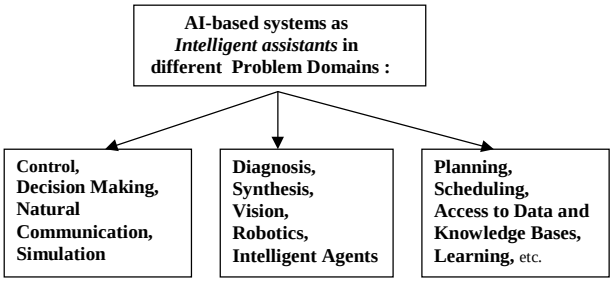


Figure 1-4. Different Application Areas of AI systems.

As shown in Fig.1-4, AI technologies have been used successfully in different tasks involving analysis (for example, machine diagnosis), synthesis (such as design), planning and scheduling, simulation, decision-making, all of which lead to significant economic gains. As we noted earlier, the best known economic impact of AI is in the use of expert systems. Expert systems save over a billion dollars each year through this technology. AI technologies are useful also for chemists and biologists in studying complex phenomena. Medical doctors, for example, use medical expert systems as intelligent assistants in patient's diagnosis and treatment, and so on. The computer and communication technologies make it possible for a rapid inexpensive sharing of knowledge.

Vision and robotics transform manufacturing. Through 1980s, industrial robots have been proved their usefulness in industrial environment and become inevitable tools in advanced production systems. Now the main interests of researchers and engineers are changing from tile robots for manufacturing automation to robots in nonindustrial environments. The latter includes robots for disabled persons assistance, service use, amusement, medical purposes, educational uses, personal use, etc. Requirements for such robots in nonindustrial environments are growing year after year.

As example, we will consider AI application for the task of development of intelligent mobile service robot. Let call it IROS. This robot works in complex office buildings with many rooms, floors, corridors, elevators, other robots and human beings. The robot is expected to perform various kinds of tasks or works instead of a secretary or service person 24 hours in an office building autonomously.

The supposed tasks or applications of the intelligent mobile service robot are set mainly by next four categories:

1. Guidance of visitors and delivery of mails or documents;
2. Maintenance and control of building utilities;
3. Safety guard and prevention of fires;
4. Cleaning floor and carrying trash.

A human-operator can directly communicate with IROS robot on restricted Natural Language by writing (on a computer keyboard installed on the robot) instructions what the robot must to do. For example, instructions may be as follows: "In the room 113 there is a business correspondence for our laboratory. Go to the room 113 and bring this correspondence." or "Grasp a book located on the John's table and bring to room 212 on the second floor".

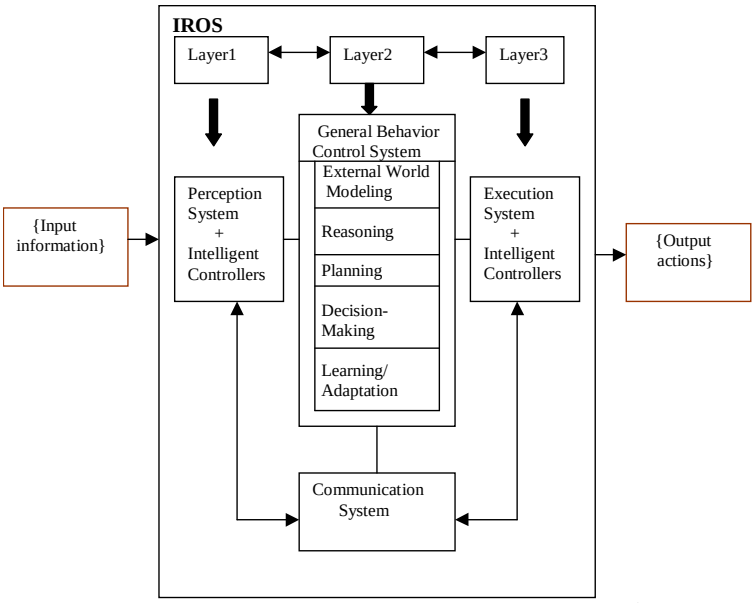


Figure 1-5. The general layered scheme of intelligent control system for IROS

Delopment of intelligent robots with physical and intelligent abilities requires a solution of different scientific and technical problems connected with the realization of the following properties:

1) various *physical motor-sensory skills* (such as vision, hearing, objects manipulation and locomotion) and
2) intelligent capabilities such as *rational behavior*, *task-level planning*, *self-learning*, *direct human-robot communication*, etc.

We will study AI methods for robot's intelligent capabilities design. In Fig.1-4 a general layered scheme of intelligent control system for IROS is shown.

We introduce *two levels of intelligence in an intelligent behavior control system.*

Layer2 (*Modeling, Reasoning, Planing, Decision-Making, Learning, Adaptation and Language-based Communication*) represents the *high level of intelligence*, while Layers 1 and 3 (*Perception, Task execution and Motion*) represent the *low level of intelligence*.

(Copia modificata per una migliore consultazione on-line)

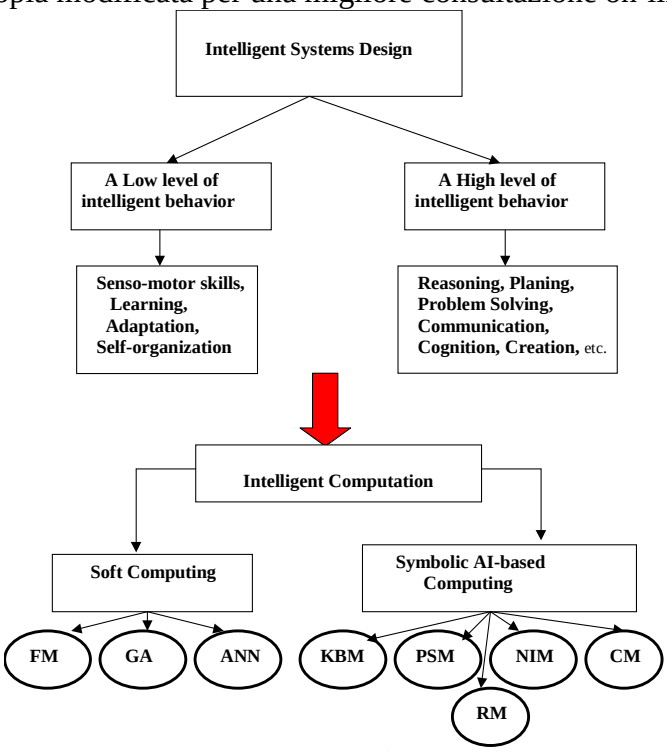


Figure 1-4. General Computing Paradigm for Intelligent Systems Design.

(FM – Fuzzy Models; GA – Genetic Algorithms; ANN – Artificial Neurons Models; KBM – Knowledge Bases Models; PSM – Problem Solving and Planning Models; NIM– Natural Interaction Models; RM – Reasoning Models, CM– Creation Models.)

Low level intelligence-based behavior appears in solving of different sensor-motor tasks (including intelligent control of motion, recognition, and adaptation). For modeling low level intelligent behavior we will apply soft computing methodology. This methodology and its application to intelligent control will be considered in the second part of our Lectures Notes.

High level intelligence-based behavior appears in solving of communication tasks, in thinking (including decision-making, reasoning and generalization), in explanation, learning, cognition and memorizing.

We believe that namely the usage of language distinguishes the high level of intelligence from another levels of intelligent behavior, and we affirm that a *high level intelligent behavior is based on a language*. Therefore for modeling high level intelligent behavior we will apply different approaches of symbolic paradigm of AI. Symbolic AI methodology and its application to intelligent behavior simulation will be considered in the first and third parts of the Lectures Notes.

In Fig.1-5 the general computing paradigm for intelligent systems design is shown.

Finally, we note that there are two main approaches to design intelligent systems. First approach is manual or *knowledge-based* approach, when intelligent systems are directly programmed to perform given tasks. In a second approach, which is a *learning-based* approach, a system is evolved over time controlled by a task-specific learning program. The first approach consists of the following steps:

- Start with a task, which is understood by a human engineer;
- Design task-specific representations (a system structure, Data and Knowledge Representations, Model of Knowledge processing, etc.);
- Program a system for the given task and representations;
- Run program on the computer and validate program's work.

According to the second paradigm, a system should be designed to go through a long period of autonomous evolution from "infancy" to "adulthood" stage. This period can be named as artificial life of designed system. The second approach is based on ideas of artificial neural networks, "supervised" and "unsupervised" learning, evolutionary methods, etc. We will study both approaches and will consider their applications to intelligent systems design.

Lecture N 2 Expert Systems and Knowledge Engineering Problems

In the Lecture 1 we have introduced two levels of intelligence in a behavior of intelligent system.

In AI it is affirmed that the *high level intelligent behavior is based on a language*. Therefore the *symbolic* AI research paradigm focuses on the attempt to simulate human intelligent behavior by expressing it in language forms or symbolic rules.

The language serves four major functions in human intelligence: *communication, thinking* (including reasoning and decision making) and *learning, cognition* and *memorizing*.

All these functions are connected with *representation* and *manipulation of knowledge*, therefore *any intelligent system may be considered as a knowledge-based system*.

The most successful symbolic AI product is *expert systems* widely used in practice in many areas.

We define the *expert system as a knowledge-based system that contains expert knowledge and provides an expertise for solving problems in a defined application area*.

In Fig.2-1 the structure of traditional expert system (ES) is shown.

(Copia modificata per una migliore consultazione on-line)

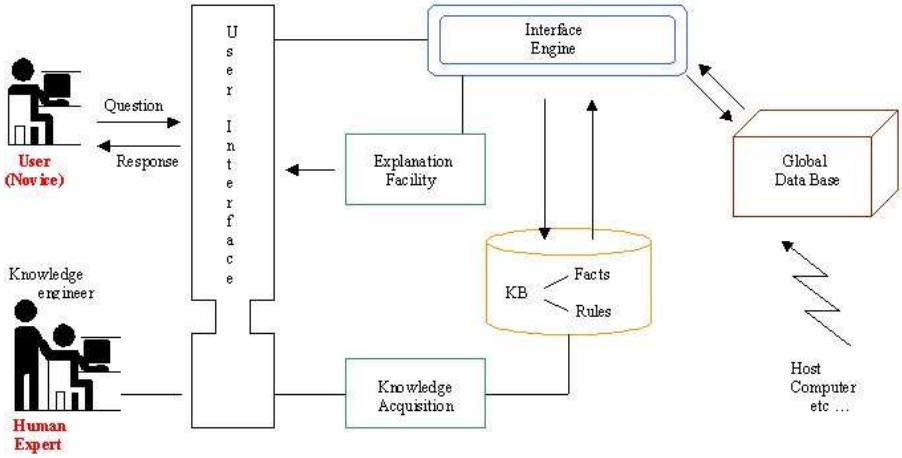


Figure 2-1. Expert System Structure.

ES has the following facilities:

1) representing existing expert knowledge;
2) learning and accumulating new knowledge;
3) making logical inferences and decisions;
4) giving recommendations;
5) communicating with users in a friendly way (often in a restricted natural language), and
6) explanation of their "behavior" and decisions.

Expert systems apply a simple theoretical idea: symbolic reasoning guided by heuristics over declaratively specified knowledge. But implementation of this idea results in impressive problem-solving ability.

ES development initiated investigation of *knowledge engineering problems.*
The main research issues in ES developing are:

1) the extraction of domain knowledge and the criteria used for decision making;
2) the knowledge representation and manipulation on knowledge.

Before considering these problems let us discuss the following questions:
What is *Data*? What is *Information*? What is *Knowledge*? and How they do differ?

We will consider *data* as the "raw material", the "mess of numbers". They could be numbers only, without contextual meaning, for example, such numbers as: 3 , 78.3, -23.

If data can be contextually explained, structured, or organized in groups and structures, the such data are called *information*. So, information is any structured data which have contextual meaning, for example, the temperature is 20 C.

We define *knowledge as high level structured information.* Another words, knowledge is "condensed" information and represents our experience. For example, "If the temperature is over 20, you may switch off a heater".

It is clear that any system can not exist in a complex real world, if properties and laws of that world do not represented in the system. Human beings need knowledge in order to survive in the real world. A huge amount of knowledge has been accumulated in the world during conscious existence of humanity. To describe this knowledge human beings use a natural language (NL). NL texts (in verbal, symbolic, representation) save our knowledge in the form of books, papers, etc.

Remark 1. There are another (nonverbal) languages for knowledge representation. For example, graphical images, pictures also save our knowledge about real world. But, the language is the most powerful tool for knowledge representing.

The base of our knowledge about the world consists of *systems of classification.* There are two types of human knowledge classification: *category-based* and *situation-based* classifications.

In the first classification, elements are joined in one system according to the relation "*general category – concrete elements*", for example "a book – the book". In the second one, elements are joined in one system according to some *typical situation*, for example, " tools for work" or "crowd of people" or "jam on roads", and so on.

A natural way of human knowledge representation is based on Natural Languages (NL). Let us analyze what kind of knowledge can be represented in NL.

1. Elements of NL may describe concrete objects and their classes. For example, "a table" describes the class of all tables and "the table" describes the concrete representative of that class of objects.
2. Some NL-elements describe object's properties (like " small", "sharp", etc).
3. Some elements of NL represent different relations (between objects, events, processes, ideas, etc.), actions, and mental actions. For example, " The lamp is on the table", or " Peter is walking on the street", "At 9 o'clock I begin my work", "A robot open a door and come in an office room", " I am going to buy a book for my son".
4. NL has imperative elements, for example, "do it", "take the book", "wait".
5. NL can describe states (initial, final, dangerous, etc.), evaluations ("good", "useful", etc.), modificators (like "strongly", "deeply",etc.), quantificators (like "often", "any", "always", etc.), and modalities (like "may be", "necessary", etc).

Human knowledge does not only consist from mentioned above elements. These elements connected in some structure representing relationships which are characteristic for real world. These relationships may represent a snapshot of a real world, or may be long-term relationships, that is, nature's laws, ways of doing things, commonsense, ideas, methods, skills, etc.

Human knowledge about the world is represented on many levels with different degree of generalization.

<u>Two questions are very important:</u>

1) *How to make use of existing knowledge in a computer program?*
2) *How to represent it in such a way that we keep the richness and the depth of the knowledge and also make it reasonable to use?*

Knowledge engineering seeks answers on these questions.

(Copia modificata per una migliore consultazione on-line)

Knowledge engineering problems

Knowledge engineering includes the following main aspects: *representation of knowledge*, *inference*, *acquisition of knowledge* and *learning*, *generalization*, *explanation*, *interaction*, *validation* and *adaptation* problems. Consider them [3].

1. *Representation* is the process of transforming existing problem knowledge to some of the known knowledge engineering schemes in order to process it by applying given model of knowledge processing. The result of representation process is the knowledge base in a computer format.

When we choose a method for knowledge representation some questions must be considered.

a) *What kind of knowledge is it*? Structured or unstructured? Exact or inexact? Precise or imprecise? Complete or incomplete?
b) *Which method of representation best* suits the way people solve that problem?
c) *Are there alternative methods for representing* the problem knowledge? *Which one is the simplest*?

2. *Inference* is the process of matching current facts from the domain space to the existing knowledge and inferring new facts.

Different inference models may be chosen: classical logic-based inference, fuzzy inference, neural-based reasoning, case-based reasoning, etc.

3. *Learning* is the process of obtaining new knowledge. Learning is a major characteristic of intelligent systems.

Three main approaches to learning are the following:

a) *Learning through examples.* Examples of the form (x_i y_i) where x_i is a vector from domain space D and y_i is a vector from solution space S , i = 1,2,…n, are used to train a system about the goal function $F : D \rightarrow S$. This type of learning is typical for neural networks. Symbolic learning from examples is also very popular scheme in AI.

b) *Learning by being told.* This is a direct or indirect implementation of a set of heuristic rules into a system. For example, the heuristic rules to monitor a car can be directly represented as rules or instructions. Given to a system in a text form by an instructor (written text, speech, natural language) they can be transformed into internal representation in computer. This kind of "learning" called a *knowledge acquisition* and it is typical for symbolic AI systems and fuzzy systems.

c) *Learning by doing.* This way of learning means that the system starts with empty (nil) or little knowledge. During its functioning it accumulates valuable experience and profits from it, so it performs better over time. This method of learning is typical for genetic algorithms.

4. *Generalization* is the process of matching new, unknowm input data with the problem knowledge in order to obtain the best possible solution, or one close to it. Generalization means reacting properly to new situations, for example, recognizing new images, or classifying new objects and situations.

Generalization can also be described as a transition from a particular object description to a general concept description. This is also a major characteristic of intelligent system.

5. *Interaction* means a communication between a system, on the one hand, and an environment or user, on the other hand, in order to solve a given problem. Interaction is important for a system because it allows to adapt to a new situation, improve itself, learn.

6. *Explanation* is a desirable property for many AI systems. It means *tracing*, the process of inferring the solution, and reporting it. Explanation is easier for the symbolic AI systems when sequential inference takes place. But it is difficult for parallel methods of inference and especially for difficult for the massive parallel ones.

7. *Validation* is the process of testing how the solutions produced by a system are good. The results produced by the system are usually compared with the results obtained by a human expert or by other systems.

Validation is very important part of the process of developing every knowledge-based system. Without comparing the results produced by the system with reality, there is no sense to use it.

8. *Adaptation* is the process of changing a system during its operation in a dynamically changing environment. Without adaptation there is no intelligence.

Separating Knowledge from Data and from Inference in the Knowledge-based Systems

There are many reasons for separating data from knowledge, and separating both from the inference (or control) mechanism in a system.

Some of these are the following:

- Data may represent a current situation, for example, the temperature of the cooling system in a car. A characteristic of data is that they may vary frequently.

- Rules are stable, long-term information. Rules do not depend on slight variations in data describing a current situation.

- Separating the control, or an inference procedure applicable to a set of rules and data, provides an opportunity to expand the knowledge when necessary without changing the inference procedure.

It also makes the decision process clearer and the process of designing the whole system program much easier.

Remark 2. Separating data from knowledge and from control contrasts with standard programming techniques. Imperative computer languages, for example, C, PASCAL, MODULA 2, separate data (variables and constants) from knowledge (procedures), but the control mechanism is still embodied in the procedures. A full separation is achieved in the so-called declarative computer languages, for example, logic programming languages (like PROLOG) and production languages (like CLIPS), etc. These languages require the presence of structured problem knowledge.

In symbolic AI systems explicit knowledge is separated from the representation of the current data and from the inference (control) mechanism.

Knowledge Representation Models

Knowledge representation models can be divided on three main classes (Fig.2-2):

- *declarative models*;
- *procedural models*; and
- *subsymbolic models*.

(Copia modificata per una migliore consultazione on-line)

Declarative models of knowledge representation desribe typically knowledge about objects, classes of objects and their properties, relations between objects, actions, states , events and their properties. They include logical models, frame/script-based formalism, semantic networks representation or some hybrid models.

Procedural models desribe typically knowledge in the form of different instructions (usually "if-then" rules), they are based on general production rules models or fuzzy models.

Subsymbolic models join Neural Networks approach and Connectionists Models.

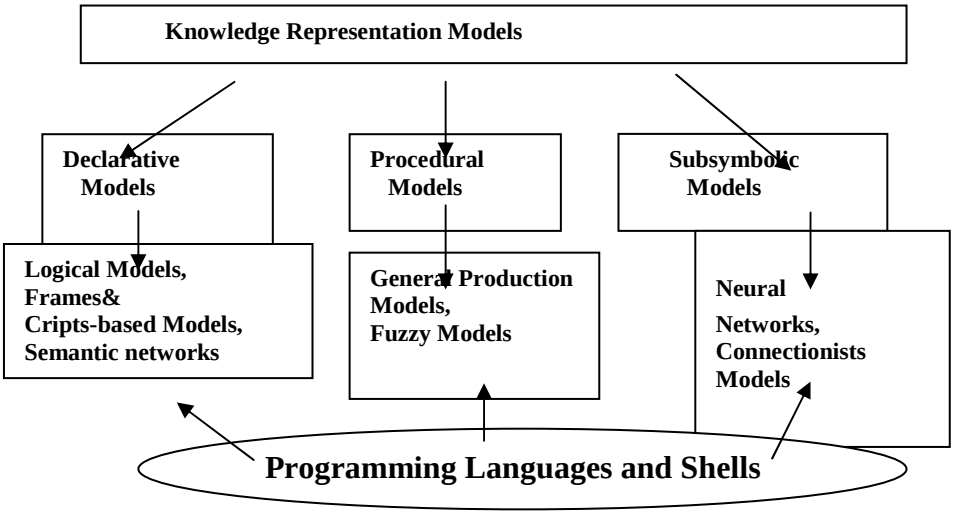


Figure 2–2. Knowledge representation Models.

Different aspects of knowledge representation problem can be distinguished:
Global vs. local knowledge. Knowledge can be global, for example, the knowledge that human beings have learned throughout their evolution, or local, for example, a rule on how to cross a road.
Shallow vs. deep knowledge. Knowledge can be shallow, for example, based on stimulus-reaction associations, or deep, for example, complex models for explanation and analysis.
Explicit vs. implicit knowledge. Knowledge can be explicit, structured, for example, written in a text, or implicit, unstructured and hidden, for example buried in data.
Complete vs. incomplete knowledge. Knowledge is complete if it ensures a solution to the problem in all cases or situations, or incomplete if it has a restricted applicability.
Exact vs. inexact knowledge. Knowledge is considered to be exact if it can be used and an exact solution to the problem is produced when exact input data are supplied, or inexact, uncertain, when it produces an approximate solution when exact or inexact data are supplied.
Hierarchical vs. flat knowledge. Knowledge is hierarchical when some pieces of knowledge apply over others, or flat when all the knowledge is applicable at the same level.
Hierarchical knowledge contains also *meta-knowledge* (knowledge applicable over knowledge).

Popular Schemes of Data, Information and Knowledge Representations in AI Systems

Before considering knowledge representation models let us refresh main schemes for data and information representation in computer systems.
Different *data representation* schemes are known from the area of database systems. For example, an object (data) can be described by a relation and represented as a tuple of the following type :
<name -of- the-object attribute1 value1 attribute2 value2 … attributeN valueN >
For ex.: (object15 S = 5 D = 3.8 PW = 0.1) or in a short form object15 = (5 3.8 0.4).

In general notation, a data set is characterized by an attribute vector $X = (x_1, x_i, \ldots, x_k)$ where x_i is the *i*-th attribute. One data instance is a point (a vector) in a *k*-dimensional attribute space.
Visualizing data may help to choose a good data representation, a proper dimensionality and proper method for solving the problem.
Information is a collection of structured data. It includes knowledge as well as simple meaningful data. An information structure is a collection of information elements with a defined organization, operations over the elements, and a defined method of access to every element.
Structures can be organized as *static* or *dynamic*. *Static structures* have a fixed number of elements.
Dynamic structures do not have a fixed number of elements. Its elements are created and deleted during process of handling the information.
Dynamic structures are based on using elements called pointers. A *pointer* (*Pnt*) is a data element whose content is the location of another element.
Sets, stacks, queues, linked lists, trees and graphs are used as dynamic information structures. Let us shortly refresh these notations.
A *set* is defined as a collection of objects with operations over it .
A *stack* is a collection of ordered elements and two operations ("push" and "pop") which can be performed only over the element that is currently at the "top" of the stack (Fig.2-3, (a), here "nil" means empty reference).

(Copia modificata per una migliore consultazione on-line)

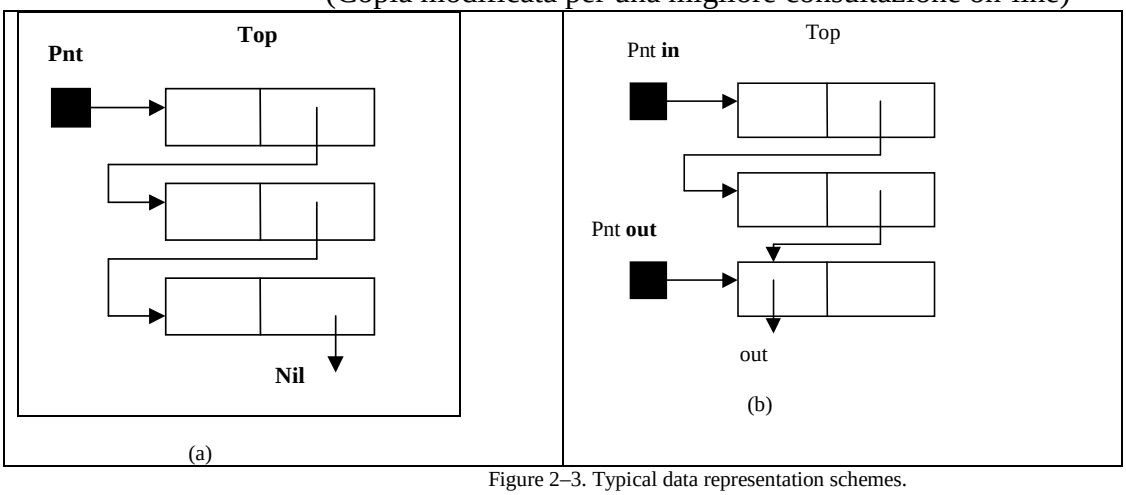


Figure 2–3. Typical data representation schemes.

A *queue* is a data structure similar to the stack, but there are two pointers used – one for the input, and one for the output element of the structure (Fig.2-3,(b)).
Linked lists are more general structure than stacks or queues.
Elements in a list may be linked in different ways (Fig.2-4). Typical operations over lists are *insert* an element in a list, *delete* an element, and *search* an element in a list.

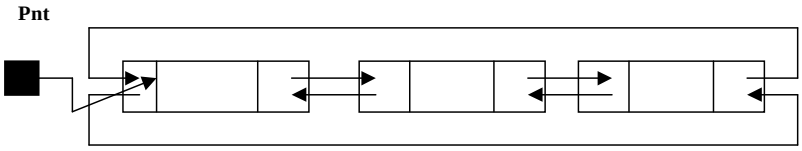


Figure 2-4. A general representation of linked list.

Knowledge is a high-level structured information representing our experience. It represents long-term relationships, that is, ways of doing things, commonsense, ideas, methods, skills, etc. Consider popular schemes of knowledge representations in AI systems.
Very useful structures in knowledge engineering are *trees* and *graphs*. They are often used for representing decision charts, classification structures, plans, scripts, etc.
A *directed graph* is a collection of a finite set of elements called *nodes* or *vertexes*, and a finite set of *directed arcs* that connect pairs of vertexes (Fig.2-5,(b)).

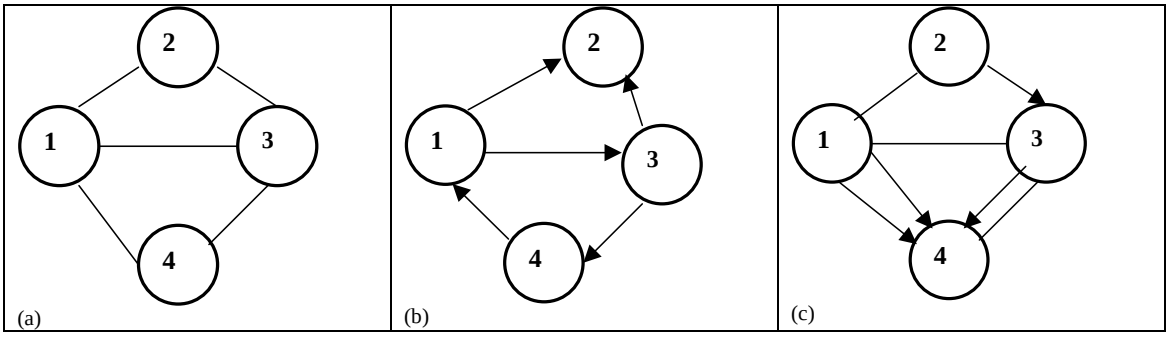


Figure 2 –5. Graphs structures: (a) simple nondirected graph; (b) directed fully connected graph;
(c) multigraph.

If the arcs in a graph are nondirected, then we have a *nondirected graph* (Fig.2-5,(a)).
If there are a few of arcs between a pair of vertexes, then we have a *multigraph* (Fig.2-5,(c)).
Remark 3. Refresh some useful definitions. A *path in a graph* is a sequence of contiguous arcs in a graph. The path which starts from one node and ends at the same node is called a *cycle*.

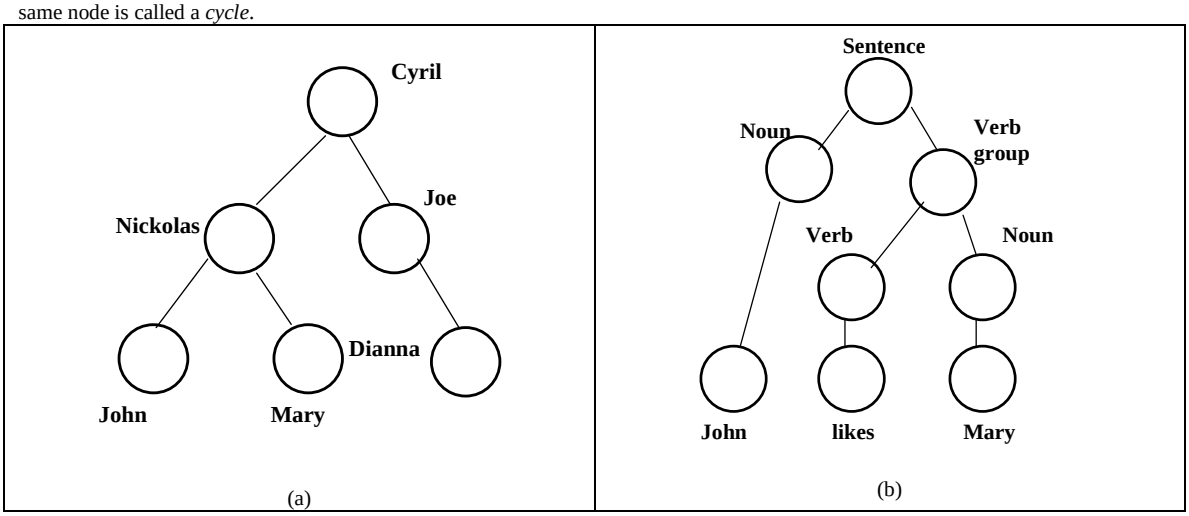


Figure 2-6. (a) A genealogical tree, (b) A Parsing tree.

(Copia modificata per una migliore consultazione on-line)

A *tree* is a directed graph in which one of the nodes, called *root*, has no incoming arcs, but from which each node in the tree can be reached by exactly one path. Vertexes that have no outgoing arcs are called *leaves*. Examples of trees are shown in Fig.2-6, 2-7. You can see a *genealogical tree* (Fig.2-6,(a)), a *parsing tree* (Fig.2-6,(b)) and an *algebraic expression tree* (Fig.2-7).

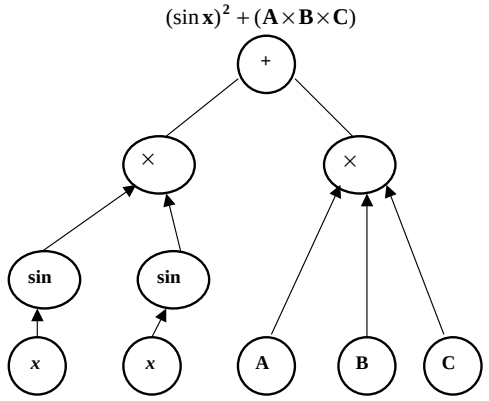


Figure 2–7. An Algebraic expression tree.

Semantic networks use *directed graph* (*or multigraphs*) to represent contextual information, natural language (NL) text's meaning, etc.
Figure 2-8 shows the contextual information representation by a simple semantic network. The nodes represent objects and concepts of a given problem area, and the arcs represent relations between them. Here "is-a" relation means the relation "to be a representative of a class".

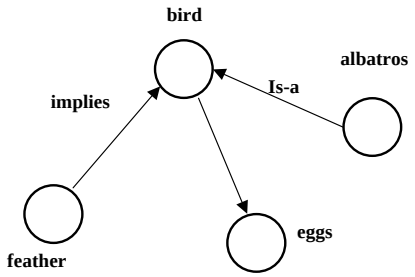


Figure 2-8. Semantic network-based knowledge representation.

Figure 2-9 shows a meaning representation of the following NL-sentence "A small cap of coffee is staying on a desk near a window".

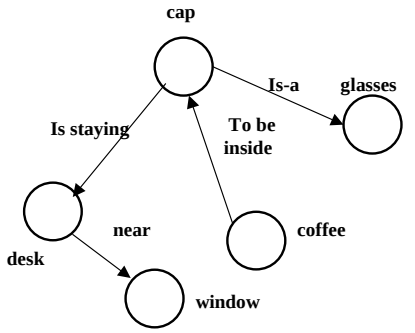


Figure 2-9. Semantic network-based NL-meaning representation.

Semantic networks are useful for hierarchical knowledge, but they are developed mainly for static information representation. Updating data by learning and changing knowledge may be difficult to handle.

Logical Models of Knowledge Representation and Reasoning

Logical models often use the predicate logic of the first order for representation of problem knowledge. A lot of different logical models are developed in AI research area. Some of these logical models, namely spatio-temporal and action logics we will discuss in the third part of lectures. Here we discuss main ideas of logical approach to the knowledge representation.

Logic is the study of the methods and principles of reasoning in all its possible forms.

Any logical system consists of five parts:

1. An *alphabet* – a set of basic symbols.
2. *Syntax* – a set of rules or operators for constructing sentences (expressions) from the alphabet elements. These structures are syntactically correct sentences and called well-formed formulas (WFF).
3. *Laws of inference* – a set of rules or laws for constructing semantically equivalent but syntactically different sentences; this set of laws is also called as a set of inference rules.
4. *Axioms* – describing all tautologies in the system. *Tautology* is a *WFF* which is *always true* regardless of the truth values of variables participating in the formula.
5. *Semantics* (or *Interpretation Model*) – for defining the meaning of the constructions in the logical system.

(Copia modificata per una migliore consultazione on-line)

Propositional Logic

Propositional logic dates back to Aristotle. There are three types of symbols in propositional logic: (1) propositional symbols (the alphabet); (2) connective symbols, and (3) symbols denoting the meaning of the sentences.
There are rules (Syntax) in propositional logic to construct *WFF* and rules to evaluate the semantics of the sentences.
Consider, for example, the following syntax rules for constructing *WFF*.
1. If *a*is any term, then (*a*) - *WFF*.
2. If *a* and *b* are any *WFF*, then

$(\neg a), (a \wedge b), (a \vee b), (a \rightarrow b), (a = b)$ are *WFFs*,

where the following logical operations are introduced:

$\neg$ - negation , $\wedge$ - conjunction (called also AND operator), $\vee$ - disjunction (called also OR operator) , $\rightarrow$ - implication and "=" – equality.

3. There is no another *WFF*.

A propositional symbol represents a statement (or proposition) about the world, for example, "The temperature is over 20".
The semantic meaning of a propositional symbol is expressed by two possible semantical symbols – *true* (*t*) or *false* (*f*). Propositions can be either true or untrue (false), nothing in between.
The semantics of the compound expressions are defined by the truth table given in Table 2-1.

Table 2-1. The truth-table of the basic connectives in propositional logic.

P	Q	$P \wedge Q$	$P \vee Q$	$P \rightarrow Q$	$P = Q$	$\neg P$	$\neg Q$
t	t	t	t	t	t	f	f
t	f	f	t	f	f	f	t
f	t	f	t	t	f	t	f
f	f	f	f	t	t	t	t

Example. Given that the propositions *P* = "Temperature is 20" and *Q* = "Humidity is high"are true. Compound propositions $P \wedge Q, P \vee Q$ are also true.

Propositional logic has rules or laws for defining semantic equivalence of syntactically correct structures. These rules make *the process of inference* in the space of possible propositions and give the truth value for some propositions when the truth values for other propositions are known.
Consider laws of inference in propositional logic. The most popular laws of inference in propositional logic are called *modus ponens* and *modus tollens*.

Modus ponens: $\dfrac{P \rightarrow Q, P}{Q}$,

which means that if we have the rule $P \rightarrow Q$ and true input *Y* and *Y* =*P* , then we can infer that *Q* is true. *Q* is considered as an output of the inference process.

Modus tollens: $\dfrac{P \rightarrow Q, \neg Q}{\neg P}$,

which means that if we have true input $\neg Q$ and rule $P \rightarrow Q$, then $\neg P$ is true, that is, *P* is false.
Two other important inference rules are

the *laws of De Morgan*: $\dfrac{\neg(P \wedge Q)}{\neg P \vee \neg Q}$, $\dfrac{\neg(P \vee Q)}{\neg P \wedge \neg Q}$.

Some simple inference laws are also valid in propositional logic.

Law of syllogism: $\dfrac{P \rightarrow Q, Q \rightarrow R}{P \rightarrow R}$.

Law of contrapositive: $\dfrac{P \rightarrow Q}{\neg Q \rightarrow \neg P}$.

Law of double negation: $\dfrac{\neg(\neg P)}{P}$.

Disjunctive inference: $\dfrac{P \vee Q, \neg P}{Q}$.

All the laws of inference can be proved for correctness using the truth table method, that is, by obtaining the truth table for the left and right sides of the inference chain and then comparing them.
The inference rules allow deduce new true *WFF* from a set of axioms and given true *WFF*. These new formulae are called *deducible* formulae.

For example, the following *WFF* are the axioms:

$$((a \rightarrow b) \wedge (b \rightarrow c)) \rightarrow (a \rightarrow c)$$

$$(a \rightarrow b) = (\neg a \vee b)$$

The inference process consists of two stages:
(1) *matching* input with a left part of a rule;
(2) *inferring* of output expression by using a law of inference.

(Copia modificata per una migliore consultazione on-line)

Propositional logic is a useful way of representing a simple knowledge base consisting of propositions and logical connectives between them. The main problem is that propositional logic can only deal with complete statements. That is, it cannot examine the internal structure of the statement. A classic example given in all the AI books is the following.
Example. The following inference is not possible in propositional logic:

$$\frac{\textit{All humans are mortal. Socrates is a human}}{\textit{Therefore, Socrates is mortal.}}$$

Propositional logic can not even prove the validity of a simple syllogism such as that above, because there is no tools to describe variables and quantifiers. In order to avoid these problems predicate logic was constructed.

Predicate Logic

The following types of symbols are allowed in predicate logic:
- *constant symbols* are symbols, expressions, or entities, which do not change during execution. Constant symbols "true" and "false" are used to represent the truth of the expressions;
- *variable symbols* are symbols which represent entities that can change during execution;
- *function symbols* represent functions which process input values for a predefined list of parameters associated with the function, and obtain resulting values. The number of parameters in a function is called *arity*.

Constant symbols, variable symbols and function symbols are called *terms*.
- *predicate symbols* represent predicates which are true/false-type relations between objects. Objects are represented by constant symbols. The number of arguments attached to a predicate define the arity of the predicate. For example, "father (person, person)" is a predicate of arity 2, and "father (John,William)" is the constant predicate.
- *Connective symbols* are the same as those which are valid for propositional logic, i.e., conjuction, disjunction, negation, implication and equivalence. They are defined by the same truth table as given for propositional logic.
- *Quantifiers* are valid for variable symbols. An *existential quantifier* $\exists x$ means "there exists at least one value for *x* from its domain", and a *universal quantifier* $\forall x$ means "for all *x* in its domain".

For example, the Socrates syllogism considered above can be expressed in predicate logic by the following way. We introduce two predicates $Human(x)$ and $Mortal(x)$. Using a universal quantifier and an implication we can write:

$$\forall x Human(x) \rightarrow Mortal(x), \quad Human(Socrates).$$

Well-formed expressions in predicate logic are all syntactically correct sentences. If a set of sentences in predicate logic is matched by a domain *D*, which means that every variable is assigned a value and every predicate is assigned a truth value, this is called *interpretation*.
First-order predicate logic allows quantified variables to refer to objects and not to predicate or function. This is not a limit for higher-order predicate logic.
In order to apply an inference to a set of predicate expressions, the system should be able to determine when two expressions match each other. The process of matching is called *unification*. In order to allow more freedom for matching and not restricting the variable domains, the existential quantifier has been eliminated by a so-called *skolemization process*. A skolem function replaces the existential quantifier $\exists$ by a function which is returning a single value.
For example, the expression

$$\forall x Human(x) \rightarrow \exists y Mother(x, y)$$

can be skolemized as follows:

$$\forall x Human(x) \rightarrow Mother(x, f(x)),$$

where *f* is a skolem function.

Programming in Logic: PROLOG

PROLOG is a new programming language designed for expert systems based on logical models of knowledge representation.
In PROLOG, a quantifier-free, so-called *Horn-clausal* notation is adopted, which can adequately represent a first-order predicate logic system.
A rule $(A_1, A_2, ..., A_n) \rightarrow B$ in a *Horn-clausal* form is written by the following way:

$$B :- A_1, A_2, ..., A_n \qquad (2\text{-}1)$$

which means that the goal *B* is true if all subgoals $A_1, A_2, ..., A_n$ are also true, where $A_1, A_2, ..., A_n, B$ are correct predicate expressions. The symbol "," in Eq.(2-1) is used to denote "AND" connective. To denote "OR" and "NOT" connectives may be used symbols ";" and "$\neg$" respectively.
The left side of the clause is called also *a conclusion*, and the right side of the clause is called *premises*.
A *fact* is represented as a *literal clause* with a right side being the constant "true". So the clause representing a fact is always true. For example, $Human(Socrates) :- true$ or shortly $Human(Socrates)$.

The Socrates syllogism is represented in PROLOG as follows:

$$Mortal(x) :- Human(x)$$
$$Human(Socrates).$$

Knowledge is represented in PROLOG as a set of clauses which have one conclusion and premises (see Eq.(2- 1)).
A program in PROLOG consists of clauses and facts. The PROLOG-program is of a declarative, rather than of a procedural type. The engine starts when a goal is given. A goal can only be a predicate used in the program. A block diagram of a PROLOG program is given in Fig.2-10.
Remark 4. A backward chaining inference engine is implicitly built in the PROLOG interpreter. We will talk about this kind of inference in the next lecture.

Example. Consider design of a dialog system for *searching of a family relationship*(given from [3]). The input to the system is an answer about a family relationship between two person, for example, "Is John a grandfather of Peter?" The system consists of a Knowledge Base (a set of rules describing family relationships), a Facts Base (a set of the existing facts) and an Inference Engine (for finding answers on the given questions). To design this system let us choose the Turbo Prolog programming language.

(Copia modificata per una migliore consultazione on-line)

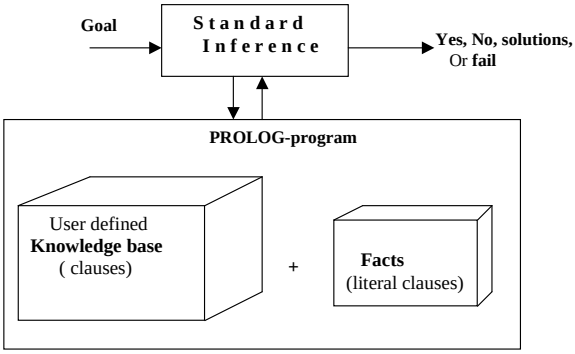


Figure 2–10. A block diagram of a PROLOG program.

(*Turbo PROLOG program "Searching for a Family Relationship"*)

(* defining the objects in the domain*)
domain
name = *symbol*

(*defining the relations between the objects*)
predicates
father (name,name)
mother (name,name)
parent (name,name)
grandfather (name,name)
grandmother (name,name)

clauses
(*facts - defining the existing initial facts*)
father (John,Mary)
father (Jack,Andy)
father (Jack,Tom)
mother (Helen,Jack)
mother (Mary,Peter)
grandfather (Barry,Jim)

(*rules – defining the knowledge base*)
grandfather (X,Y):- father (X,Z), parent (Z,Y)
grandmother (X,Y):- mother (X,Z), parent (Z,Y)
parent (Z,Y) :- mother (X,Y)
parent (Z,Y) :- father (X,Y)

Given in the program above Knowledge Base for a family relationship may be graphically represented as an AND-OR tree shown in Fig.2-11.

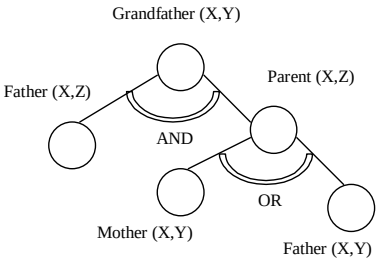


Figure 2–11. AND-OR tree representing relations between predicates.

A sample dialogue between user and the system for searching for family relationship is given as follows:

User: Goal?- grandfather(John, Peter)
System: yes
User: Goal?- grandmother(Helen, X)
System: 2 solutions
X = Andy X = Tom
User: Goal?- grandmother(X,Y),X = Y
System: fail

The system tries to find facts (in the Facts Base) which prove the goal and the subsequent subgoals (Fig.2-12). The matching operation during the inference process is called "unification", that is, it matches predicates, variables and facts in order to find an answer. In Fig.-3 the unification procedure in PROLOG is illustrated on the family relationship example.

(Copia modificata per una migliore consultazione on-line)

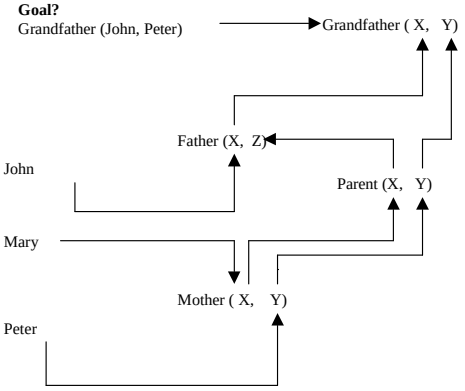


Figure 2–12. Example of the unification in PROLOG program.

Frame-based Knowledge Representation Model

Frames are structures representing structured information for standard, typical, situations of problem area. The frame consists of *slots* (variables) and *fillers* (values). Slots may represent not only static information but dynamic information as well. For example, the slot may represent a procedure to be processed over data from other slots, or it may be a reference on another frame. By using the references frames may constitute some structure represented by a semantic network. Nodes of this semantic network are names of frames, and arcs are relations between them, for example "is-a" relation (the relation "general concept – concrete example") or "part-of"-relation, temporal or causal relations, and so on.

The notion of a frame was introduced by M.Minsky [6]. He considers *the frame as a minimal description of an appropriate appearance of the real world*. "Minimal description" means the following: if we delete some part in this description, then the frame lost their capability to describe this appearance.

Usually two types of frames are used for knowledge representation: *frame-prototype* and *frame-representative*. First one describes some typical entity of real world (for example, objects, relations, events, processes, scripts, etc.), second one – a concrete representative of that frame.
Formally the frame may be described as a following structure:
(< *name of frame*>:
<*name of slot*1> = <*value of slot*1>, …
<*name of slotN*> = <*value of slotN*>),
where N is the number of slots, < *name of frame*> and <*name of slot*> are symbols.
The <*value of slot*> may be introduced more detail, for example, as follows:
<*value of slot*> ::= <*descriptor of value*> / "*constant term*"/ <*name of frame*>/ (list of names of frames) / number/ <procedure>/ WFF / *comments* / etc.
(Here WFF is a well formed formula representing some fact of a real world). Examples of frames are given below.
Example. Let us describe a *fragment of knowledge representation* for the task *of understanding and visualization of natural language texts describing spatial relations in some real world.* Consider, for example, the following NL-text:
" Mary opened a door and saw a desk in the center of a room. A small lamp was on the desk. In the left back angle of the room was a wardrobe. A sofa was located right and not far from the wardrobe. A small nice cat was sleeping on the sofa."
This text describes one situation of a real world at some moment of time (a *world's state*) including objects in this state and spatial relationships between them.
Development of the system capable to understand the NL-text and make the corresponding picture includes the knowledge base design stage. We will use the frame-based formalism for this task, and will construct the Knowledge Base as the set of all frames prototypes for the given problem area. The Data Base, or Facts Base, contains the set of frame-representatives for the given input.
In different realizations may be different syntax of frames. We choose the syntax shown in the following examples.
Frame-prototypes

(world: isa = "prototype"; name = room ; type = "simple" ; x = [number] ; y = [number] ; z = [number] ; size-x = [number] ; size-y = [number] ; size-z = [number] ;)	(table: isa = "prototype"; class = object ; type = "complex" ; x = [number] ; y = [number] ; z = [number] ; size-x = [number] ; size-y = [number] ; size-z = [number] ; color = [number of color] ; configuration = (top left-front-leg right- front-leg left-back-leg right-back-leg);)

In the left-hand side (relative to "=") of this frame structure there are names of slots, and in the right-hand side descriptors of slots values are given. The names of slots reflect the properties of the considered fragment of knowledge. In our case, the frame "world" represents the world named "a room". This world is represented as a parallelepiped with a coordinate system attached to it. The location of the coordinate system origin is described by values of the "x","y" and "z"- slots. The world's sizes along *X-Y-Z* axes (i.e. a length, a width and a height of a room) are described by values of the "size-x", "size-y", "size-z" slots.
The frame-prototype "table" represents a class of physical objects named "table" and a corresponding fragment of knowledge needed for our task. We describe a table as a parallelepiped with a coordinate system attached to it. The location of the coordinate system initial point is described by the slots "x","y" and "z" (see Fig.2-13). The table has a length, a width and a height (values of the "size-x", "size-y", "size-z" - slots).
We also consider a table as a complex physical object consisting of the following parts: top, left-front-leg, right-front-leg, left-back-leg, and right-back-leg. This list of object's parts is the value of "configuration" slot. Each object's part is also a physical object and is described by corresponding frame-prototypes.
Figure 2-13 shows a graphical image of the object "table".

(Copia modificata per una migliore consultazione on-line)

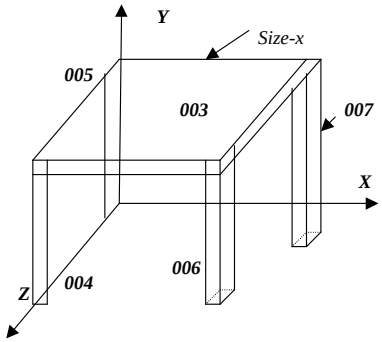


Figure 2-13. A graphical image of the object “table” described by a frame-prototype.

(left-front-leg: isa = “prototype”; class = object ; type = “simple” ; part-of = table ;	x = [number] ; y = [number] ; z = [number] ; size-x = [number] ; size-y = [number] ; size-z = [number] ; color = [number of color] ;)

Frame-prototypes for spatial relations:

(near:
isa = “prototype”;
type = “position relation”;
membership-function = “F1”;
argument1 = object ;
argument2 = object ;
rule = Prog {“R001”} ;)

(far:
isa = “prototype”;
type = “position relation”;
membership-function = “F2”;
argument1 = object ;
argument2 = object ;
rule = Prog{ “ R002”} ;)

The knowledge fragment connected with the spatial relation called “near” describes a position relation between two objects (for example, object1 and object2). In order to represent procedural knowledge we introduce special symbol Prog. The expression “rule = Prog {“R001”}” means that the value of the slot “rule” can be defined by running the procedure named R001. For example, it may be the program that calculates a location of object1 if the position of object2 is known.

Frame-prototypes for actions:
(move- itself:
isa = “prototype” ;
type = “physical action”;
who = agent-of-action ,
to = place ;
from = place ;
initial time = *t* ;
duration = time interval ;
necessary-condition = Prog {check: *<does the path from initial place to final place exist in a current scene ?* }
action = *GO-TO*{*initial point final point*} ;
result-of-action = Prog{*WFF-design*}

The frame called “move itself” represents knowledge connected with this physical action.
The action is described as a command “*GO-TO*” (from an initial point or place to a final point or place) performed by a person (agent of action) at initial time *t*. The person of the action is described by the value of the “who” slot, the initial place of moving is given by the value of the “from” slot, and the final point of the action is given by the value of the “to”slot. In order to describe necessary conditions needed for the action performance and results of the action, we introduce slots called “necessary-condition” and “result-of-action”. An output of the procedure called *WFF-design* is a well formed formula (*WFF*) describing results of the given action. For example, for the action “move itself” its result may be described as following *WFF*: (<the agent of the action><is situated in> <the final place > <at time equal “*t*+duration”>).

Frame-representatives:
Consider examples of frame-representatives for the frame-prototypes considered above.
Names of frame-representatives are given by the system.

(001: isa = world; name = room ; type = “simple” ; x = 0 ; y = 0 ; z = 0 ; size-x = 4000 ; size-y = 4000 ; size-z = 4000 ;)	(002: isa = table; class = object ; type = “complex” ; x = ; y = ; z = ; size-x = 120 ; size-y = 80 ; size-z = 60 ;) color = 1 ; configuration = (003 004 005 006 007);) *list of names of frames-representatives*

Frame representative 001 means that it is the concrete representative of the frame “world” with given values of slots. “002” is representative of the frame

(Copia modificata per una migliore consultazione on-line)

"table" with given values of slots, where 003,004,005,006,007 are representatives of table's parts.

(003:
isa = top;
class = object ;
type = "simple" ;
part-of = table ;
x = ;
y = ;
z = ;
size-x = 100 ;
size-y = 60 ;
size-z = 10;
color = 1 ;)
	(004:
	isa = left-front-leg;
	class = object ;
	type = "simple" ;
	part-of = table ;
	x = ;
	y = ;
	z = ;
	size-x = 15 ;
	size-y = 15 ;
	size-z = 50;
	color = 1 ;)
		(005:
		isa = left-back-leg;
		class = object ;
		type = "simple" ;
		part-of = table ;
		x = ;
		y = ;
		z = ;
		size-x = 15 ;
		size-y = 15 ;
		size-z = 50;
		color = 1 ;)
			(006:
			isa = right-front-leg;
			class = object ;
			type = "simple" ;
			part-of = table ;
			x = ;
			y = ;
			z = ;
			size-x = 15 ;
			size-y = 15 ;
			size-z = 50;
			color = 1 ;)
				(007:
				isa = right-back-leg;
				class = object ;
				type = "simple" ;
				part-of = table ;
				x = ;
				y = ;
				z = ;
				size-x = 15 ;
				size-y = 15 ;
				size-z = 50;
				color = 1 ;)

So, shown above frame-representatives describe the object "table" in the room 001.

We have considered a few examples of frames prototypes for the task of NL-texts understanding and visualization. Finally, we may note that a set of frames prototypes graphically can be represented as some semantic network, where nodes are the frame-prototypes names and arcs are relationships between the frames, for example "is-a" relation, or "part-of" relation, etc.

Script-based Knowledge Representation

By using the frame formalism we can represent knowledge about different classes of objects, their properties, and different kinds of relations between them. Consider now how to describe more complicated knowledge concerning our experience. For example, how to describe different typical situations and plans for achieving goals. For this aim we introduce a new knowledge structure called a script [7].

Scripts are structures for representation of chains of typical situations of problem area for achieving a given goal. *The script describes a plan of actions, which must be performed in order to achieve the goal of the script*. On the conceptual level scripts are more complex structures, but on the syntactical level they can be represented by frame-like structures.

Let us discuss how to describe a typical situation in our real life named as "going to restaurant". Consider the following *script* represented as a frame-prototype:

(going-to-restaurant):
isa = "prototype";
type = "script";
goal = {to have a lunch or dinner/to celebrate a birth day/etc.};
time = [date] ;
duration = time interval,
who = person ;
guests = (list of persons)
plan-of actions = ("move itself into restaurant", "find where to sit", "move itself to sitting position", "sit down", "waiter is moving to the person", "receiving menu", "reading menu", "making a choice", "waiting a food", "eating", "waiting a bill", "move itself to cashier", "giving money and bill to cashier", "move itself out of restaurant"))

You can see that the notion of script is applied for the description of situations when one event follows the other. This is a simple script structure. More complex typical situations can be described by introducing different temporal relations inside a plan of actions.

Remark 5. Special tools and languages capable to work with frame-script-based representations must be developed. In different intelligent systems different tools are used [8].

Procedural Models for Knowledge Representation

Procedural models desribe typically knowledge in the form of different instructions (usually "if-then" rules), and they are based on general *production rules* models or fuzzy models.

Production rules are used to represent knowledge in the form of rules "If --Then". The *generalized production rule* may be represented as follows:

IF A_1 and A_2 and A_n , THEN $C_1, C_2, ..., C_n$, (NT , SF, CF) ,

where condition elements A_i are either fuzzy or exact propositions of the form (*x is A*);

$C_1, C_2, ..., C_n$ are actions (or facts); and

NT is a *noise tolerance parameter*, SF is *a sensivity factor*, CF is a *confidence factor* .

Consider, for example a generalized production rule typically used in medical expert systems:

IF *A1* is High and *A2* is Medium and *A3* is High THEN D1 is High and D2 is No (0.7, 1, 0.9) .

Here *A1*,*A2*,*A3* are medical symptoms, D1 and D2 are diagnoses, and (0.7 ,1, 0.9) are values of the *noise tolerance parameter*, the *sensivity factor*, and the *confidence factor* respectively.

This kind of rules are very useful for modeling reasoning in social, economic, medical, political, control and so on expert systems. In different expert systems different reasoning techniques are developed. In the second part of our lectures we will consider different fuzzy models for expert knowledge representation and reasoning.

Subsymbolic models for knowledge representation join Neural Networks approach and Connectionists Models considered also in the first part. The models are based on representing knowledge as a stable state of a system consisting of many small elements which interact with one another. This way of representing knowledge is distributed one: knowledge is represented as a state of a coalition of element. We will consider different neural models for expert knowledge representation and reasoning in the first part of our lectures.

In advanced expert systems a hybrid knowledge representation is used, for example one part of a knowledge base is described by frames-prototypes (declarative knowledge), and the second part of the knowledge base (procedural knowledge) is described by production rules.

So, we discussed different methods of knowledge representation. There is no universal method that can handle the variety of problem knowledge. What method for representing problem knowledge can be chosen depends from problem area and system designer.

Discuss now very important question.

Symbolic, Fuzzy, Neural Systems – which one is the best for knowledge engineering?

Table 2-2 gives a rough comparison between them from the standpoint of expert system's capabilities.

Table 2-2. A comparison of symbolic, fuzzy, and neural-based expert systems.

(Copia modificata per una migliore consultazione on-line)

Issue	*Symbolic*	*Fuzzy*	*Neural*
representation	*structured*	*structured*	*unstructured*
inference	*exact*	*approximate*	*approximate*
learning	*modest*	*no*	*very good*
generalization	weak	*very good*	*very good*
interaction	*good*	*good*	*good*
explanation	*very good*	*good*	*weak*
validation	*good*	*good*	*modest*
adaptation	*modest*	*modest*	*good*

Lecture N 3 Knowledge Processing

Knowledge engineering requires mathematical models and theories to represent and do reasoning over knowledge in a consistent way. Such theories are the logic systems. So, a lot of logical models are developed in AI. They are used for *external world modeling, knowledge representation and reasoning.* Some of them we have discussed in our previous lecture, namely: propositional logic, predicate logic, another (fuzzy logic, spatio-temporal logics, action logics, etc.) we will consider in following lectures.

Let us talk now about knowledge processing. The main mechanism of the knowledge processing is an inference over knowledge (or reasoning).

We define *inference over knowledge* as *the process of acquiring new facts when interpreting existing knowledge with a current data*.

The whole process of inferring new facts and their manipulation is called *reasoning*.

Inference (*or reasoning*) *in AI system* is what corresponds to the *control* in control systems, and what corresponds to the *process of* "*thinking*" in the human brain.

Reasoning is differently realized in different logic systems and AI methods. There is a lot of reasoning schemes in human reasoning: classical deductive reasoning; approximative reasoning, including fuzzy reasoning; neural-based reasoning; inductive reasoning; abductive reasoning, including case-based reasoning, reasoning by analogy, scale-based reasoning etc.

Main part of reasoning process is an inference procedure. Consider typical inference schemes in AI.

Inference as a process of matching

Suppose the knowledge base contains input-output associations (rules, data pairs) $R_j = (X_j, Y_j)$, where all X_j describe a domain space *D*, and all Y_j describe a solution space *S*. The inference problem stated in the definition is to find a solution for a new input vector x_i^*.

This vector is mapped to the solution space through a chain of rules until a solution y_i^* (satisfying a given goal or constraint) in the solution space is found. Figure 3-1 illustrates this process. The solution is found after firing three rules in a chain. The first two infer partial solutions which are used as input data for the next rules in the chain.

The matching process can be either exact, or partial.

Exact matching means that the matching facts coincide exactly with the condition elements in the left side of the rules. For example, if the fact is x_i^*, there has to be a rule $X_j \to Y_j$, such that $X_j = x_i^*$. Exact matching is typical for symbolic AI systems.

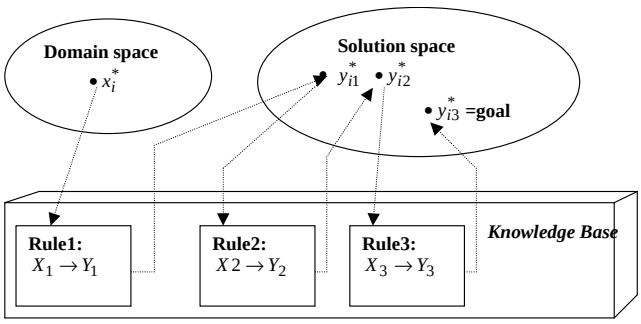


Figure 3-1. Inference as a chain of matches.

A *partial match* does not require equality between facts and the left side of a rule. In this case an approximation or closeness is enough. But how approximately and how closely should a fact match a rule in order to fire that rule? This point has to be determined by the method used for approximate reasoning. Partial match is typical for fuzzy systems and for neural networks-based models.

The inference process can be considered as *a trace of states* ($y_{i1}^*, y_{i2}^*, y_{i3}^*$) in the problem (solution) space, where every state represents a closer approximation to the solution. *The trace can be used for* <u>*explaining*</u> *the solution process.*

(Copia modificata per una migliore consultazione on-line)

Forward and Backward Inference Schemes

Consider typical inference problem as follows:

Given a rule-based Knowledge Base and a set of current facts from problem domain considered as the input to an inference procedure. Find a solution satisfying a given goal or constraint.

Return to our definition of an *inference process as a chain of matchings.* Two mechanisms to organize these matchings are called *forward chaining inference* and *backward chaining inference.*

A *forward chaining inference* applies all the facts that are available at a given moment to all rules to infer all the possible conclusions. So, the forward chaining inference starts with data, this is a *data-driven inference scheme*.

Remark 1. Simple neural networks and fuzzy systems can also realize this type of reasoning because they require data to be supplied to the inputs in order to start reasoning (see Part 2 of Lectures Notes).

Example. Consider the *Car monitoring expert system*.

The expert rule base consists of "if, then" rules describing different situations with a car and possible actions, for example,
" If <There is overheating > OR <the sensor respond slowly>, then <Stop a car>";
"If < The temperature is over 120> AND <the sensor works properly>, then <There is overheating >".

The rules can be represented as *a decision tree* shown in Fig. 3-2 (a,b).

Forward chaining inference starts with data ("the temperature is 130" and "the sensor is OK") and infers the solution "stop the car" (Fig. 3-2, (a)).

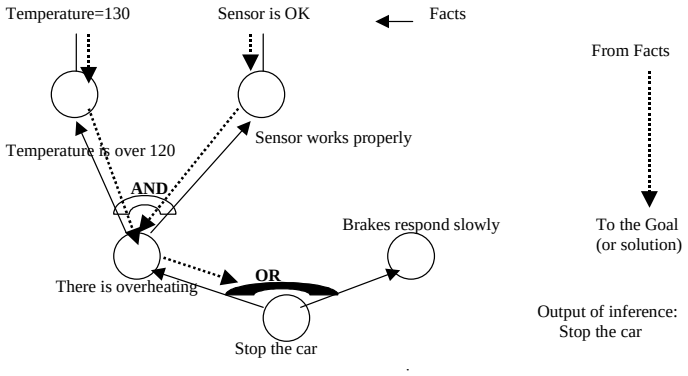
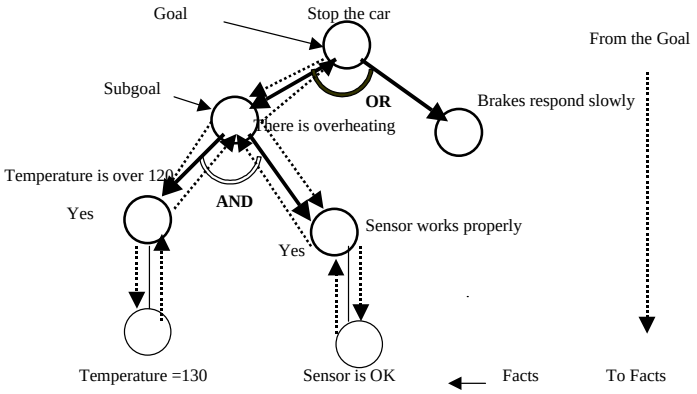


Figure 3-2 (a). Forward chaining inference example.

Backward chaining inference starts after a goal is specified. *It is* a goal-driven inference scheme. *Then a search a rule which has this goal in its conclusion (right) part is performed, and then data (facts) which satisfy all the conditions for this rule (left part) are sought in the DataBase (see Fig. 3-2, b). The process is recursive, that is, a condition in a rule may be a conclusion in another rule (or other rules).*

Backward chaining may imply a *back-tracking* procedure. That is, if the goal is not satisfied with the first checked fact, then other conditions and facts are checked (if more are available).

In our example, if the concrete fact is as follows: "the temperature is below 120"(that is, there is no overheating), then inference procedure goes back to OR-connection in the decision tree and considers another subgoal "Brakes respond slowly". This mechanism is implemented in the AI logic programming languages.



(b) Backward chaining inference example.

Figure 3-2. Two mechanisms of inference in expert systems.

(Copia modificata per una migliore consultazione on-line)

Inference as a search in a problem state space

Generally, *inference may be considered as a process of solution searching in a problem state space which optimizes a goal function* (or *achieves a given goal*).

Problem State Space Representation

There are problem areas (for example, games and puzzles) where it is convenient to represent problem domain as a space of states.

State Q is defined as a set of variables $Q = \{q_0, q_1, \ldots, q_n\}$ describing situations or events of some class of a problem area. For example, for chess playing task a state may be considered as one position on a chess board.

A way that is used to transform one state to another is called *operator*. An operator may be an action (movement, for example), rule, mathematical operator, logical symbol, etc.

State space of a problem is described as a triple $\{S, O, G\}$, where S is a *set of all possible initial states of problem*, O is a set of operators, and G is a set of goal states for the given problem.

So, a state space may be described by a graph that represents all possible states and their relationships. If an initial state's set consists of only one state, then the state space may be described by a tree structure. Consider for example an old child game called "cross-zero" (Fig. 3-3).

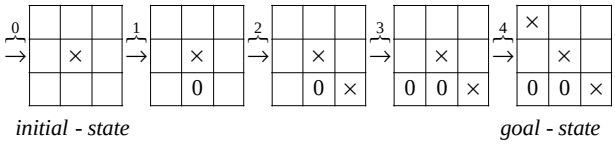


Figure 3-3. Example of the "cross-zero" game: one player begins and wins.

Here 0,1,2,,3,4 are the numbers of steps performed subsequently by two players. For this simple game we can describe the state space for all initial states. In Fig.3-4 an example of a state space for one initial state is shown.

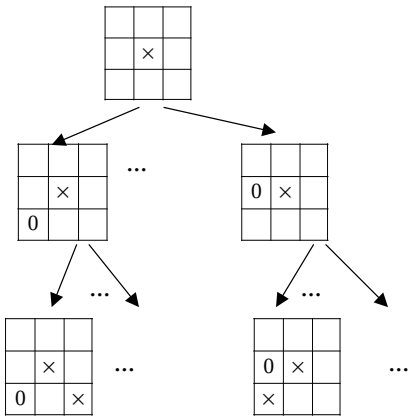


Figure 3-4. A "cross-zero" game state space representation for the given initial state.

In the case of "cross-zero" game, the inference problem is formulated as *finding a path from initial state to a goal state, or finding an optimal path to reach a goal state* (by using some optimization criterion, for example minimum of time).

Remark. For the chess game there are a huge number of all possible states, about 10^{116}. Of course, a human being can not consider all possibilities in his/her brain, and uses heuristics in order to find optimal solution.

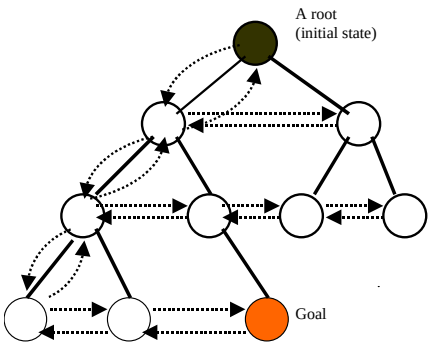
If the problem state space is <u>unstructured,</u> that is there is *no structure that represents relations between states in the space*, then a random search may be used (see quantum search algorithm approach). A state is generated randomly, and a criteria for "fitness" is applied to evaluate how good the generated state is as possible solution.

When the problem state space is <u>structured</u>, for example in a tree, then the most favored *searching strategies* are:

(1) "*breadth first search* is testing first all the states which are at a certain level in the tree and then going to search among the lower levels;

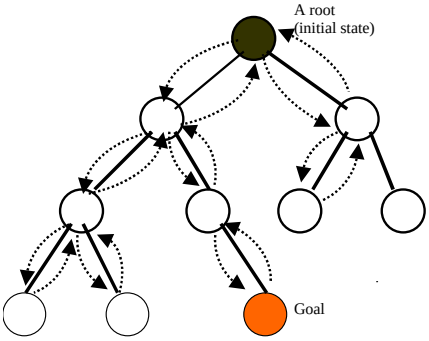
(2) "*depth at first*" is searching for the leftmost leaf of the tree and then for the next one.

Figure 3-5 (a,b) illustrates these two strategies.



(a) Breadth first search.

(Copia modificata per una migliore consultazione on-line)



(b) Depth at first search.

Figure 3-5. Two strategies of searching in problem state space.

So, we discussed main inference (reasoning) schemes applied in AI.

Finally, let us classify reasoning methods from the following point of view:

- *Monotonic* versus *non-monotonic* reasoning. An inference process is *monotonic* when every new fact contributes to an increase in present knowledge. It is *non-monotonic* when knowledge may decrease in "volume" as new facts are entered into the system. In the latter case, some facts that have been previously inferred based on the previous set of facts might need to be revised and retracted as being no longer valid.
- *Exact* versus *approximate* reasoning. A reasoning process is *exact* if it produces exact solutions when a current data are supplied. It is *approximate* if it ends up with an approximate solution or a degree of approximation attached to the inferred solution.

Let us return to beginning of this topic – to expert systems, which are the most visible contribution of AI to industrial applications, and consider as example the expert system from the area of medical diagnosis.

"Dasha" : the expert system for the choice of lower limbs prosthetics and assessment of the prosthetic quality

"Dasha" (Central Scientific Research Institute of Prosthetic, Russia) [9] is the hybrid expert system with depth representation of knowledge for the design, diagnostic and tune-up of bioengineering products (such as lower limb's prosthetics). It allows optimize the design process saving manufacturing time and improving product quality. It helps also to find a necessary solution for a given patient in the case of young, non-experienced, medical person.

The system performs two main functions: 1) the *design of a bio-engineering product*; and

2) *diagnostics of the designed product for real patient*.

These two stages are realized in two system's shells shown in Fig.3-6. Shell 1 is used for the design of patient's prosthetic (of lower extremities) and Shell 2 is used for evaluation of the prosthetic quality.

Shell 1 includes the following expert subsystems (ES):

- Evaluation of the condition of a patient's organism. Exposing contraindications to prosthetic fitting;
- Treatment resulting in elimination of the contraindications;
- Selection of the components of the prosthesis, the parameters of its assembly scheme;
- Training in use of the prosthesis.

Shall 2 includes expert subsystems for:

- Evaluation of the quality of the prosthesis; and
- Elimination of defects.

ES for evaluating the condition of the patient's organism automatically fills in the *medical chart* and provides the recommendations in preparation for the prosthetic fitting process.

ES for selecting prosthetic components includes the following stages:

1) selection of the casing material;
2) selection of the method of attachment;
3) selection of the knee mechanism;
4) selection of the foot material;
5) selection of the ankle type;
6) determination of the classification codes of the prosthetic and its elements.

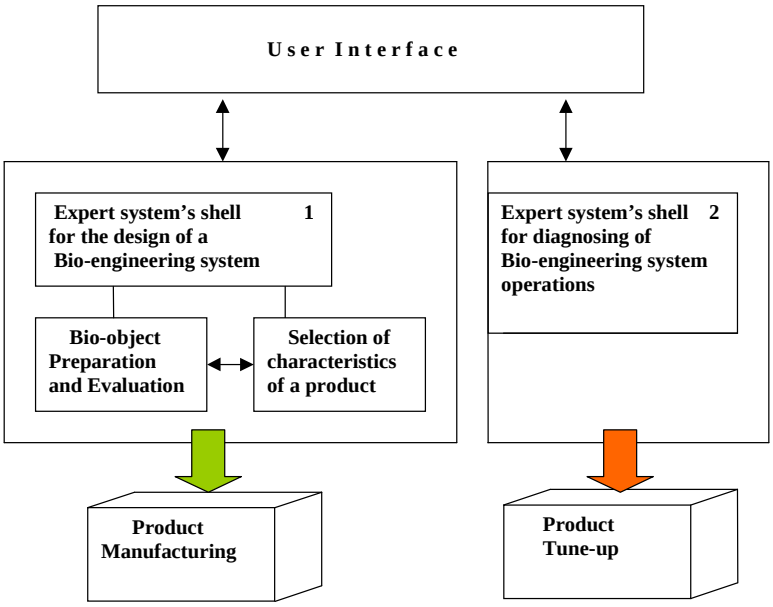


Figure 3-6. The architecture of the hybrid expert system "Dasha" for design and diagnostics of bio-engineering products.

(Copia modificata per una migliore consultazione on-line)

Special system functions provide the system user the following capabilities:
- organization of a question-answer dialog;
- connection with external software systems;
- file operations;
- decomposition of the knowledge base.

Each stage of any expert subsystem described above is characterized by a strictly-specified solution space. The knowledge bases include a wide collection of cause-and-effect relations with reliability factors to account for the uncertainty in the knowledge. The presence of different structure knowledge bases and the use of different types of logical inference are distinguishing features of the described expert system. The basic elements of knowledge are represented as fuzzy rules (or productions) in the form “if <condition>, then <action>”. Each production *j* is written as follows:
(*j*) IF <antecedent of *j*> THEN <consequent of *j*> and (*j*,CF,<author>,<date> <explanation>), where <antecedent of *j*> and <consequent of *j*> are described in terms of linguistic variables represented by fuzzy sets; CF is the degree of truth of the rule which depends on the rank of the expert who authored the rule; <author> is the identifier of the expert who is the author of the rule; <explanation>) is the text of the author’s explanation of the rule.

For example, fuzzy rules may be as follows. Left parts of the rules describe different symptom’s names, and right parts of the rules describe names of anomalies of a bio-engineering product. Examples of symptoms and anomalies are shown in Fig 3-8.
The rules are autonomous in the sense that none of the rules contains references to others.
The DataBase of the system is represented as a set of following libraries:
- Library of absolute and relative contraindications to prosthetic fitting;
- Archive of the patient’s medical data;
- Library of the elements of the prosthetic;
- Library of training samples;
- Patient’s complains and requests.

The model of a user interaction with the system is designed for untrained user (for example, an unexperienced medical person). The version of system operation as a trainer includes a collection of explanation on the decision making process.

The main menu of the system includes the following positions:
- medical hart;
- contraindications;
- element selection;
- assembly;
- quality control;
- system manual;
- “prosthetic history” print-out;
- exit from the system.

The system is implemented in two versions: professional and training. The latter is distinguished by a large number of prompts, explanations and interpretations. The system also includes a number of service functions: knowledge base creation and editing; printing and a multi-window interface.
Main menu and subsystems are shown in the following below Figures 3-7, 3-8, 3-9.

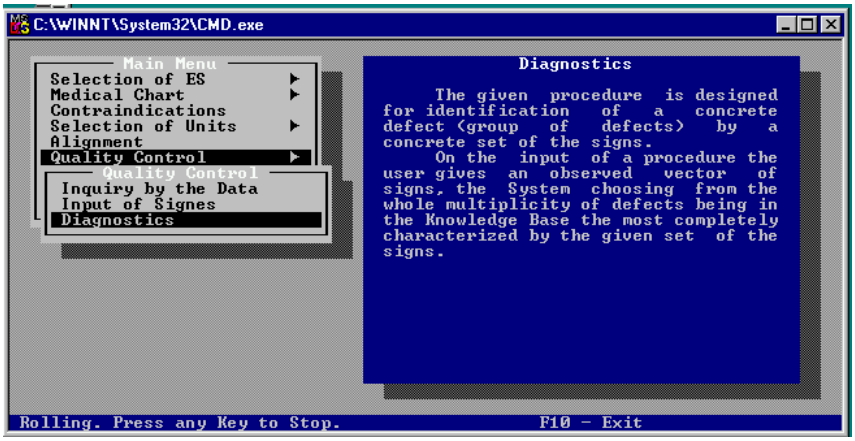


Figure 3-7. Main menu of the “Dasha” Expert System.

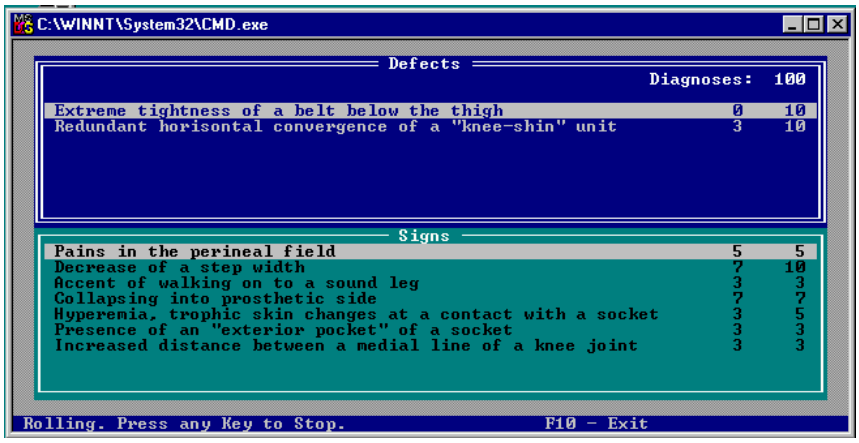


Figure 3-8. Examples of symptoms and anomalies.

(Copia modificata per una migliore consultazione on-line)

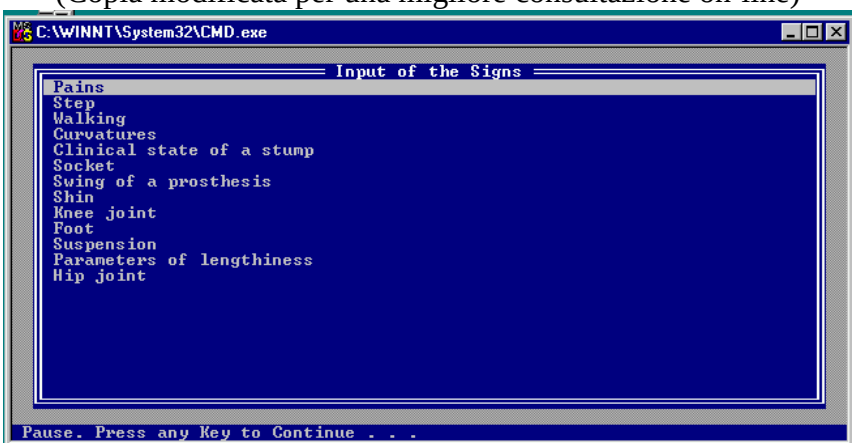


Figure 3-9. Menu for different signs input .

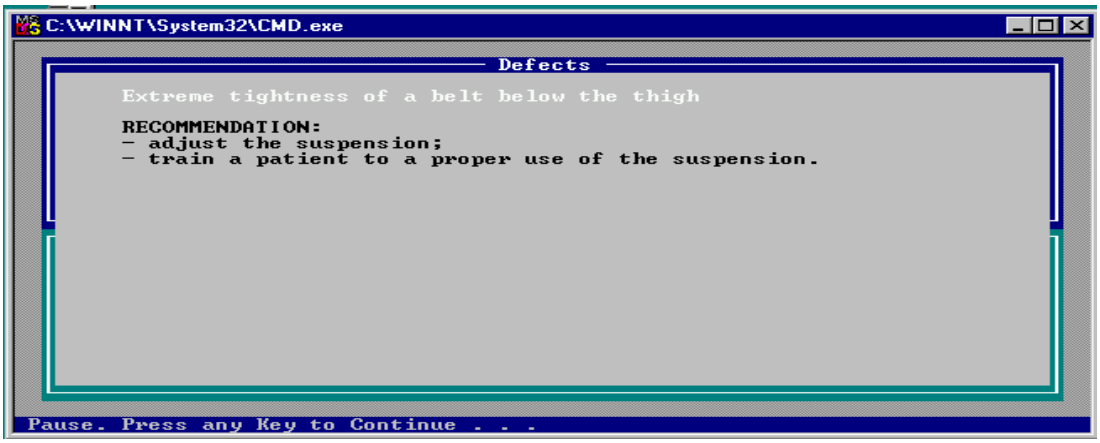


Figure 3-10. Inferred Recommendation of ES.

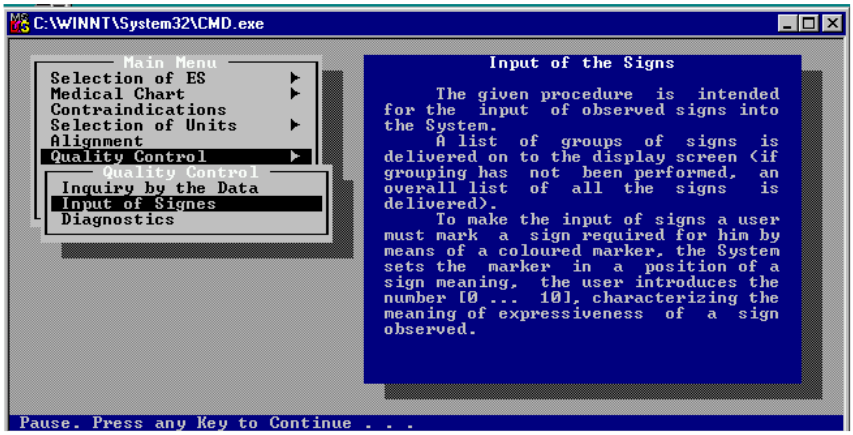


Figure 3-11. Input of signs submenu .

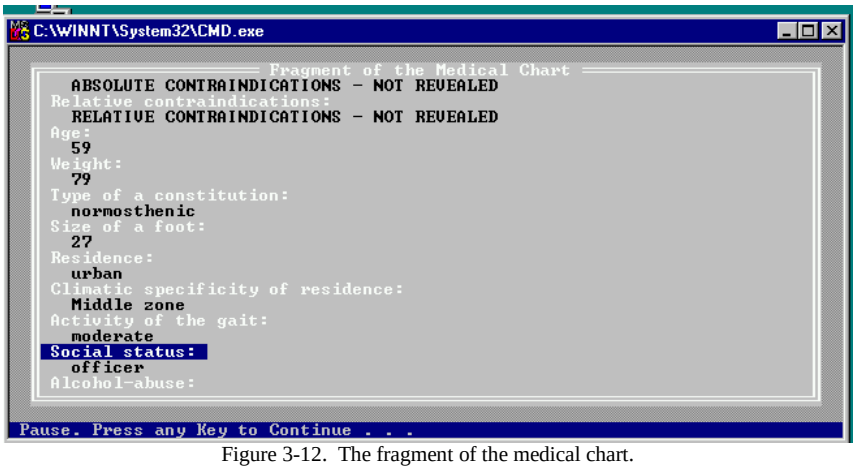


Figure 3-12. The fragment of the medical chart.

(Copia modificata per una migliore consultazione on-line)

P a r t 2 Soft Computing as Flexible Tools for Knowledge Representation, Processing and Acquisition

Lecture N 4 Human Decision Making and Fuzzy Sets

In our everyday life we often use a linguistic, imprecise approach to describing input conditions and rules for decision making. For example, we often use such linguistic concepts as " very rapid", " very cold", "not very large", "not so warm", "very young", "near", "often", etc. There are a lot of situations when we say, for example: "If tomorrow will be very cold, I will stay at home", or "If a velocity of a car on a road is small, and it is far from me, then I can cross the road", etc.

For many practical problems an important information comes from *human experts*. Usually, information is not precise and is represented by vague terms like mentioned above (small, large, not very large, and so on).

There are many reasons why expert information is usually expressed in a vague form, such as: for convenience, or lack of more precise knowledge, or ease of communication. May be also such sensitive way is much closer to human feelings than exact numerical information.

How do simulate such a kind of human decision- making and how do formalize such a kind of vague information? In this lecture we will talk about this.

Fuzzy systems arose from the desire to describe complex systems and decision making process with linguistic descriptions.

In 1965, *L. Zadeh* [10] published the first paper on a novel way of characterizing nonprobabilistic uncertainties, which he called "fuzzy sets".

Today fuzzy sets theory evolved into different disciplines, such as fuzzy logic, fuzzy graphs, fuzzy control, fuzzy modeling, and so on.

The applications of fuzzy systems include automatic control, data processing, computer vision, decision making, and others. A lot of control systems design is based on fuzzy modeling. Therefore we begin to study this approach. We will consider the following topics.

Classical (Crisp) Sets	$\Leftrightarrow$	*Fuzzy Sets*
Classical set's operations	$\Leftrightarrow$	*Fuzzy sets operations*
Classical Arithmetic	$\Leftrightarrow$	*Fuzzy Arithmetic*
Classical Logic	$\Leftrightarrow$	*Fuzzy Logic*
Classical Reasoning	$\Leftrightarrow$	*Fuzzy Reasoning*
Classical Control	$\Leftrightarrow$	*Fuzzy Control*

Let us refresh now basic concepts of classical set theory, and then we will introduce and consider the basic concepts of fuzzy set theory.

Classical Sets Theory Background

Basic notions

We shall consider the *universe of discourse*, or *universal set X*.

A classical set is a set with a crisp boundary. For example, a classical set *A* can be expressed as

$$A = \{ x | x > 6 \},$$

where there is a clear, unambiguous point "6" such that if *x* is greater this number, then *x* belongs to the set *A* , otherwise *x* does not belong to this set.

A set *A* can be described either by naming all its elements (the *list method*) or by specifying some well-defined properties satisfied by the elements of the set (the *rule method*).

For example, the following sets

$A = \{ a_1 , a_2 ,...,a_n \}$ is defined by the list method ;

$B = \{ b \mid b$ has properties $P_1, P_2,...,P_n \}$ is defined by the rule method. (Here the symbol "|" denotes the phrase "such that".)

A set may contain as finite number of elements and infinite number of elements.

An important and often used universal set is the set of all points in the *n*-dimensional Euclidean vector space R^n (that is all *n*-tuples of real numbers).

We shall consider also the set *N* to denote all positive integers, that is $N = \{ 1,2,3,...\}$, and the set $N_n = \{1,2,3,\ldots, n \}$.

Consider now basic definitions.

Definition 1:

A set *A* in R^n is called *convex* if, for every pair of points $r = r_i | i \in N_n$ and $s = s_i | i \in N_n$ in *A* and for every real number λ between 0 and 1, exclusively, the point $t = (\lambda r_i + (1-\lambda) s_i | N_n$) is also belongs to *A*.

Another words, a set *A* in R^n is convex, if for every pair of points *r* and *s* in *A*, all points located on the straight line segment connecting *r* and *s* also in *A*.

The process by which individuals from the universal set *X* are determined to be or not to be elements of a set *A* can be defined by a following *characteristic function*:

$$\mu_A(x) = \begin{cases} 1 & \text{if and only if } x \in A, \\ 0 & \text{if and only if } x \notin A \end{cases}$$

Basic classical sets operations

Definition 2: (Subset)

(Copia modificata per una migliore consultazione on-line)

If every element *x* of *A* $(x \in A)$ is also belongs to a set *B* $(x \in B)$ then *A* called a *subset* of *B*, and this operation is written as $A \subseteq B$.

If $A \subseteq B$ and $B \subseteq A$, then *A* and *B* contain the same elements and called equal sets, this is written as *A* = *B*.

If both $A \subseteq B$ and $A \neq B$ then A is called a *proper subset* of B. It is denoted as $A \subset B$.

Definition 3: (Relative Complement)

The *relative complement* of a set *A* with respect to set *B* is the set containing all the elements of *B* that are not also the elements of *A*. This operation is written as

$$B - A = \{x | x \in B \ \ and \ \ x \notin A\} \ .$$

If the set *B* is universal set *X*, then the complement of *A* is absolute and is usually denoted as $\overline{A}$. The following formulas are true:

$$\overline{\overline{A}} = A, \overline{\varnothing} = X, \overline{X} = \varnothing, where \varnothing \ is \ empty \ set.$$

Definition 4*:* (Union)

The *union* of sets *A* and *B* is the set containing all the elements that belongs either to set *A* alone, to set *B* alone, or to both set *A* and set *B*. This operation is denoted as:

$$A \bigcup B : \{x | x \in A \ \ or \ \ x \in B\}$$

The follows is true: A $\bigcup X = X$, $A \bigcup \varnothing = A, A \bigcup \overline{A} = X$.

Definition 5*:* (Intersection)

The *intersection* of sets A and B is the set containing all the elements belonging to both set A and B. It is denoted as $A \bigcap B$:

$$A \bigcap B : \{x | x \in A \ \ and \ \ x \in B\}$$

The follows is true: A $\bigcap X = A$, $A \bigcap \varnothing = \varnothing, A \bigcap \overline{A} = \varnothing$.

In Fig.4-1 you can see the graphical interpretation of mentioned above basic sets operations.

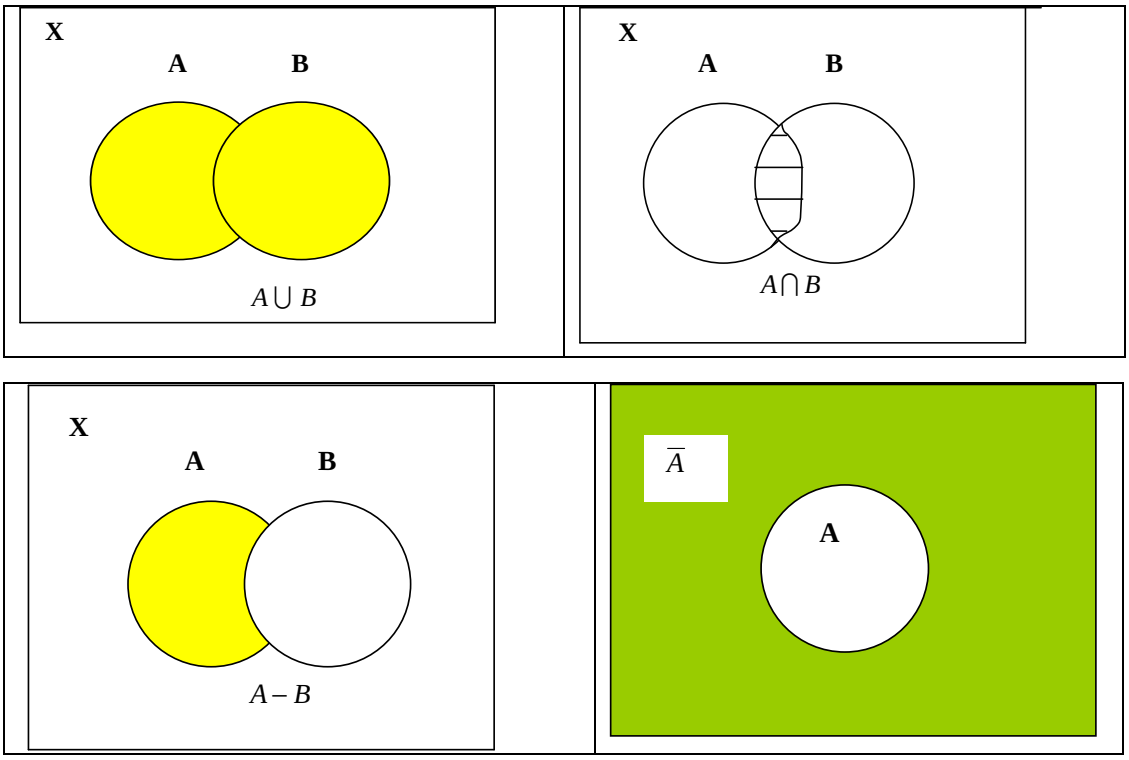


Figure 4-1. Basic sets operations.

Properties of classical sets

There are several important properties that are satisfied by the sets operations. All these properties are shown in the Table 4-1.

Table 4-1. Properties of classical sets.

(Copia modificata per una migliore consultazione on-line)

1. *idempotant law :* $A \cup A = A,$ $A \cap A = A$	6. *double negation law :* $\overline{\overline{A}} = A,$
2. *commutative law :* $A \cup B = B \cup A,$ $A \cap B = B \cap A$	7. *De Morgan law :* $\overline{A \cup B} = \overline{A} \cap \overline{B},$ $\overline{A \cap B} = \overline{A} \cup \overline{B}$
3. *associative law :* $(A \cup B) \cup C = A \cup (B \cup C),$ $(A \cap B) \cap C = A \cap (B \cap C)$	8. *complementarity laws :* 8.1 : *law of excluded middle :* $A \cup \overline{A} = X,$ 8.2 : *law of contradiction :* $A \cap \overline{A} = \phi$
4. *absorbtion law* $A \cup (A \cap B) = A,$ $A \cap (A \cup B) = A$	9. *identity laws :* $A \cup X = X,$ $A \cap X = A$
5. *distributive law :* $A \cap (B \cup C) = (A \cap B) \cup (A \cap C),$ $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$	10. $A \cup \phi = A, A \cap \phi = \phi$

Fuzzy sets

Definition 6: (fuzzy set)
If *X* is the *universe of discourse*, then *fuzzy set A* in *X* is defined as a set of ordered pairs:

$$A = \{(x, \mu_A(x)) | x \in X\},$$

where $\mu_A(x)$ is called the *membership function* (MF) of x in *A*. The MF maps each elements of *X* to a continuous membership value (or membership grade) between 0 and 1.
This definition of fuzzy set is an extension of the definition of a classical set in which the characteristic function is permitted to have continuos value between 0 and 1.
Usually the universal set *X* may contain either discrete objects or continuous values.

Examples of Fuzzy Sets

Example 1. *Fuzzy sets with Discrete X.*
Consider the universal set *X* of ages of human beings as follows:
X = { 5,10,20,30,40,50,60,70,80}. Introduce the following fuzzy sets labeled as:
infant, *adult*, *young*, and *old*.
In Table 4-2 the elements of this fuzzy sets are defined.

Table 4-2

Elements (ages)	*Infant*	*Adult*	*Young*	*Old*
5	0	0	1	0
10	0	0	1	0
20	0	0.8	0.8	0.1
30	0	1	0.5	0.2
40	0	1	0.2	0.4
50	0	1	0.1	0.6
60	0	1	0	0.8
70	0	1	0	1
80	0	1	0	1

In Fig.4-2 the membership functions for this fuzzy sets are shown.

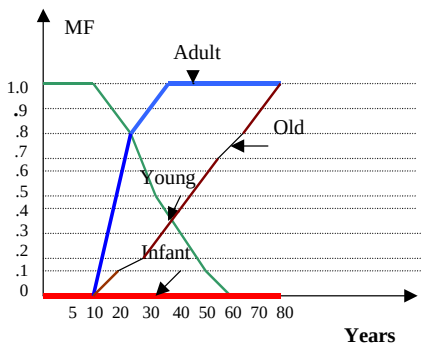


Figure 4-2. Membership functions for fuzzy sets shown in Table 4-2.

(Copia modificata per una migliore consultazione on-line)

You can see here an *empty fuzzy set* - called *infant*. It is an example of *empty fuzzy set within the chosen universe*.
Consider another universal set *X* defined as follows.
Let *X*= {1,2,3,4,5,6,7,8} be the set of numbers of courses a student may take in a semester. Introduce the fuzzy set *A*= "appropriate number of courses taken":

A = { (1,0.1), (2,0.3), (3,0.8), (4,1), (5,0.9), (6,0.5) (7,0.2) (8,0.1) }

This fuzzy set is shown in the Fig.4-3(a).

Example 2. *Fuzzy set with Continuous X*.
Let $X = R^+$ be the set of all possible ages for human beings. Then the fuzzy set *B* = "people about 50 years old" may be expressed as

$B = \{(x, \mu_B(x)) | x \in X\}$, where $\mu_B(x) = \dfrac{1}{1+[(x-50)/5]^4}$ (see Fig.4-3(b)).

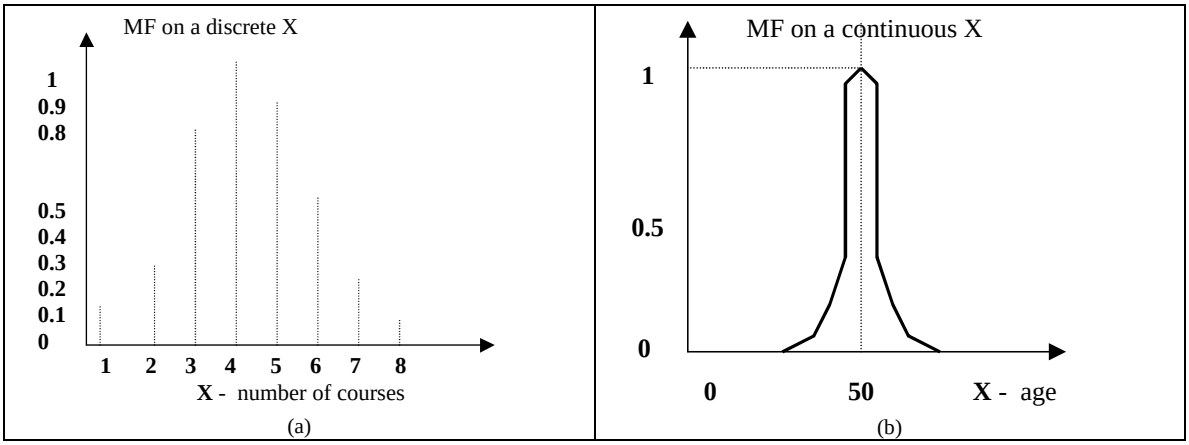


Figure 4-3. Examples of fuzzy set on the discrete universe (a) and on the continuous universe (b).

Remark 1.
There is an alternative way of denoting of a fuzzy set *A* as shown below.

$$A = \sum_{x_i \in X} \mu_A(x_i)/x_i, \quad if\ X\ is\ discrete,$$

or

$$A = \int_X \mu_A(x_i)/x_i \qquad if\ X\ is\ continuous. \qquad (4\text{-}1)$$

The summation and integration signs in Eq.(4-1) stand for the union of $(x, \mu_A(x))$ pairs; they do not indicate summation or integration. Similarly, "/" is only a marker and does not imply division.

Using this notation, we can rewrite the fuzzy sets in Examples 1 and 2 as

A = 0.1/1+0.3/2+0.8/3+1.0/4+0.9/5+0.5/6+0.2/7+0.1/8

and

$$B = \int_{R^+} \frac{1}{1+[(x-50)/5]^4} / x.$$

From examples 1 and 2, we can see that the construction of a fuzzy set depends on two things:
1) identification of a suitable universe of discourse; and
2) specification of an appropriate membership function.

Representation of membership functions

We will consider a few classes of parametrized functions commonly used to define MF's.

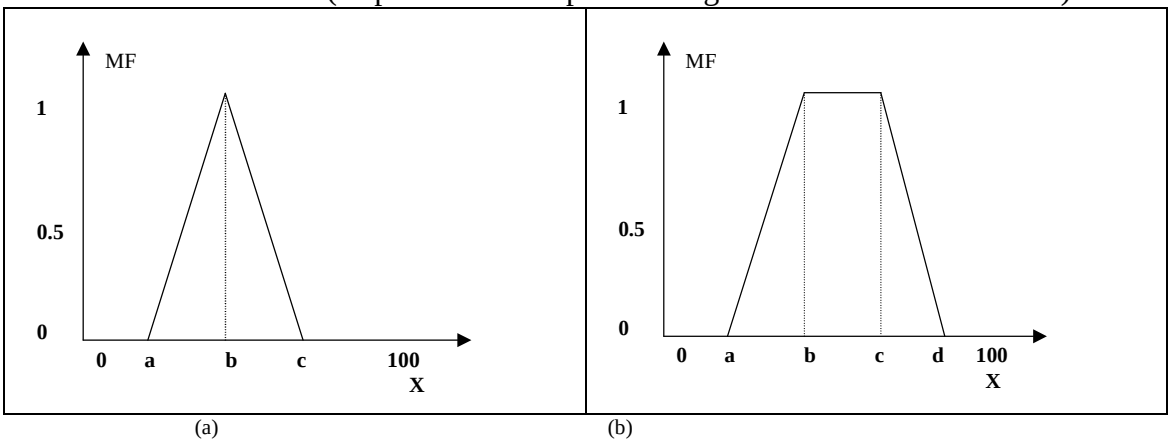
Definition *7: Triangular MF's.*
Triangular MF (Fig.4-4,a) is specified by three parameters {*a*,*b*,*c*} which determine the *x* coordinates of three corners.

$$triangular(x,a,b,c) = \max\left[\min\left(\frac{x-a}{b-a}, \frac{c-x}{c-b}\right), 0\right]$$

Definition 8: Trapezoidal MF's.
Trapezoidal MF (Fig.4-4,b) is specified by 4 parameters {*a*,*b*,*c*,*d* } as follows

$$trapezoid(x,a,b,c,d) = \max\left[\min\left(\frac{x-a}{b-a}, 1, \frac{d-x}{d-c}\right), 0\right]$$

(Copia modificata per una migliore consultazione on-line)



(a) (b)

Figure 4-4. Triangular and trapezoidal membership functions.

Definition 9:Gaussian MF's.

Gaussian MF (Fig.4-5, a) is specified by 2 parameters as follows:

$$gaussian(x,\sigma,c) = e^{-[(x-c)/\sigma]^2} .$$

Definition 10*:* Generalized Bell MF's.

A generalized Bell MF (Fig.4-5,b) is specified by 3 parameters as follows:

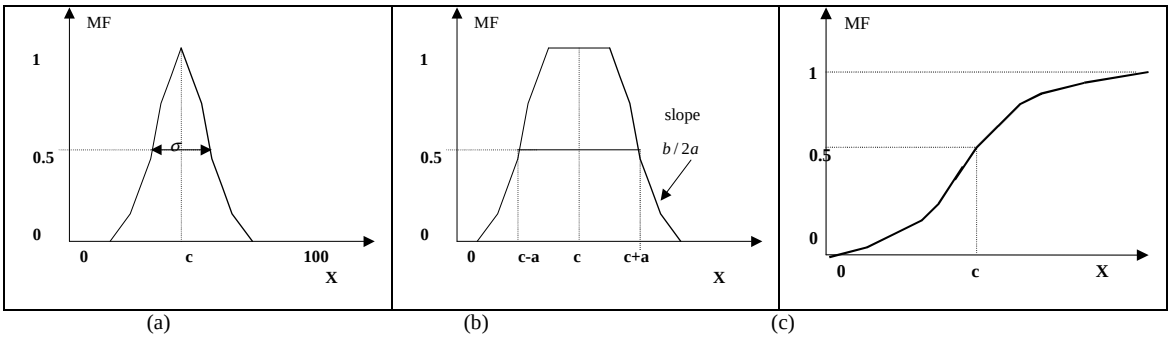
$$bell(x,a,b,c) = \frac{1}{1+\left|\frac{x-c}{a}\right|^{2b}} .$$

Definition 11: Sigmoidal MF's.

Sigmoidal MF (Fig.4-5,c) is specified as follows:

$$sigmoid(x,a,c) = \frac{1}{1+e^{-a[(x-c)]}} .$$

Triangular and trapezoidal membership functions are often used in many applications, especially in real time implementations. Sigmoidal MFs are used widely as the activation function of ANN (Artificial Neural Networks).



(a) (b) (c)

Figure 4-5. Gaussian, generalized Bell and Sigmoidal membership functions.

Basic fuzzy sets operations

Definition 12: Fuzzy Containment (or Fuzzy Subset)

Fuzzy set *A* contained in fuzzy set *B* (or, equivalently, *A* is a subset of *B*) if and only if $\mu_A(x) \le \mu_B(x)$ for all *x* .

In symbol form: $A \subseteq B \Leftrightarrow \mu_A(x) \le \mu_B(x)$.

Definition 13: Equality of Fuzzy Sets

The equality of fuzzy sets *A* and *B* is written as $A = B$ and defined as

$$A = B \Leftrightarrow \mu_A(x) = \mu_B(x) \text{ for every } x \in X .$$

Definition 14: Fuzzy Union (or Fuzzy Disjunction)

The union of two fuzzy set *A* and *B* is fuzzy set *C*, written as $C = A \bigcup B$ or *A* OR *B* or A $\bigvee$ B, whose MF is defined as

(Copia modificata per una migliore consultazione on-line)

$$\mu_C(x) = \max\{\mu_A(x), \mu_B(x)\} = \mu_A(x) \vee \mu_B(x).$$

Definition 15: Fuzzy Intersection (or Fuzzy Conjunction)

The intersection of two fuzzy sets *A* and *B* is fuzzy set *C*, written as $C = A \cap B$, or *C* = *A* AND *B*, or *C* = A $\wedge$ B, whose MF is defined as

$$\mu_C(x) = \min\{\mu_A(x), \mu_B(x)\} = \mu_A(x) \wedge \mu_B(x).$$

These definitions of fuzzy AND and OR operations are satisfy the *law of distributivity*:

$$\mu_{A\cup(B\cap C)}(x) = \mu_{(A\cup B)\cap(A\cup C)}(x)$$

$$\mu_{A\cap(B\cup C)}(x) = \mu_{(A\cap B)\cup(A\cap C)}(x) \qquad (4\text{-}2)$$

Definition 15: Fuzzy Complement (or Fuzzy Negation).

The complement of fuzzy set *A*, written as $\overline{A}$, or $\neg$ A , or NOT A, is defined as

$$\mu_{\overline{A}}(x) = 1 - \mu_A(x).$$

In Fig. 4-6 all these basic fuzzy operations are demonstrated.

Remark 2. Note that another definitions for fuzzy AND and fuzzy OR have been proposed in the literature. A popular alternative for fuzzy AND and OR-operations are:

$$\mu_{A\cap B}(x) = \mu_A(x)\mu_B(x),$$

$$\mu_{A\cup B}(x) = \mu_A(x) + \mu_B(x) - \mu_A(x)\mu_B(x), \qquad (4\text{-}3)$$

but these definitions do not satisfy the law of distributivity.

Properties of Fuzzy Sets

There are several important properties that are satisfied by the fuzzy sets operations. Some properties are valid for both fuzzy and crisp sets, and some properties are valid for crisp sets or for fuzzy sets only.

For example, *idempotant, commutative, associative* laws,*double negation* and *De Morgan* laws (see Table 4-1) are valid for both fuzzy and crisp sets. Definitions 14,15 of fuzzy conjunction, disjunction and negation given above satisfy the law of *distributivity*, but definitions given in Eq.(4-3) do not satisfy this law.

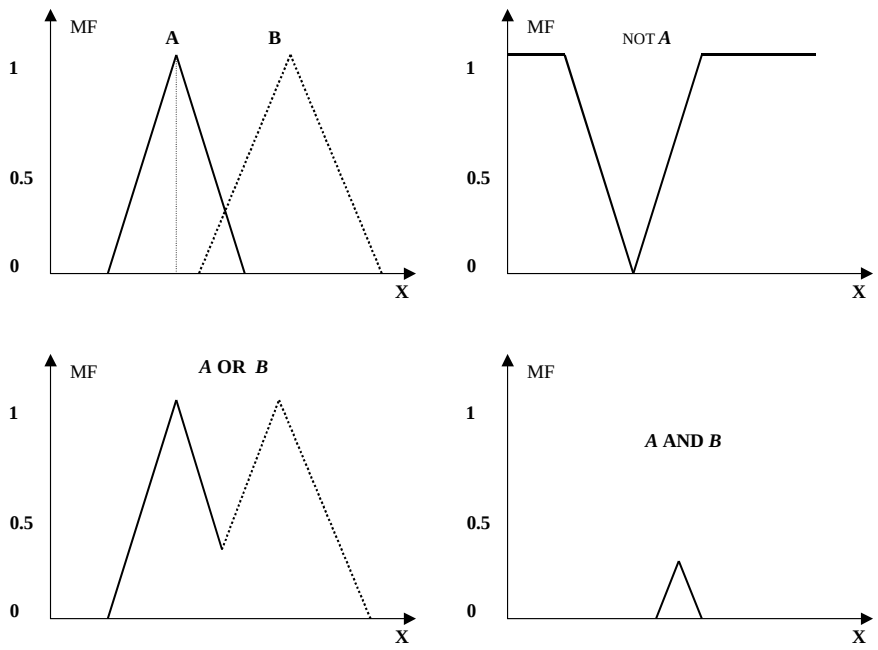


Figure 4-6. Basic fuzzy sets operations.

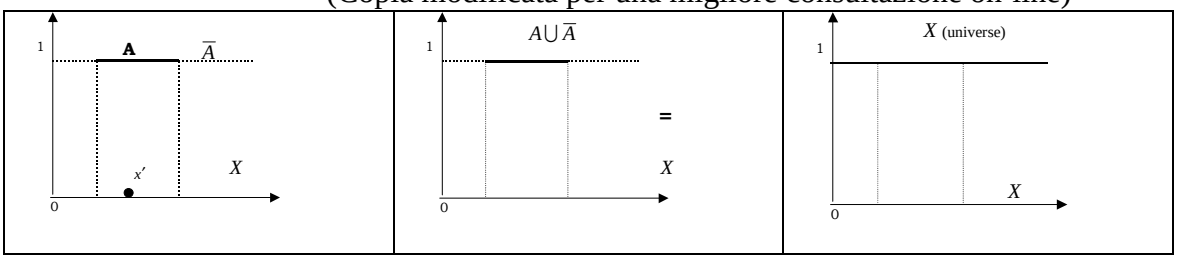
<u>Laws of *excluded middle* and *contradiction* (see 8.1 and 8.2 in Table 4-1) are valid for crisp sets, but in general are not valid for fuzzy sets</u>.

The law of excluded middle for fuzzy sets: $A \cup \overline{A} \neq X$.

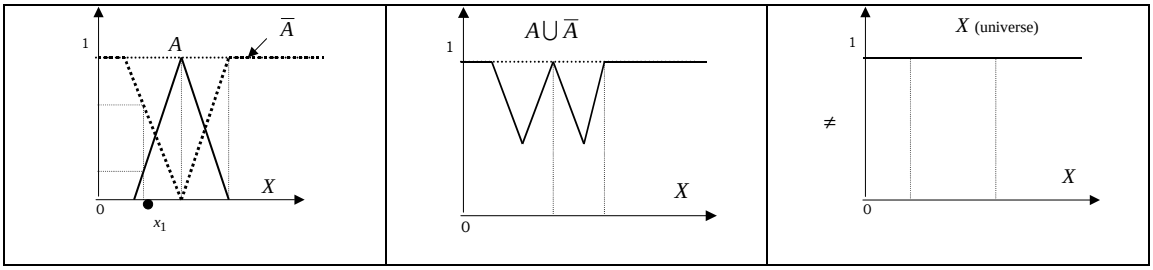
The law of contradiction for fuzzy sets: $A \cap \overline{A} \neq \phi$, where ϕ means an empty set.

Figure 4-7 illustrates this difference.

(Copia modificata per una migliore consultazione on-line)



(a)



(b)

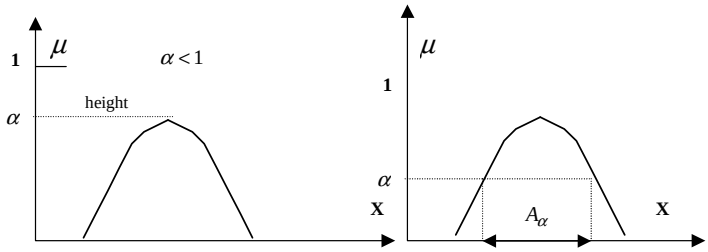
Figure 4-7. The laws of excluded middle and contradiction for crisp sets (a) and for fuzzy sets (b).

In Fig. 4-7(a), one point x' belongs only to the set *A* or to the set $\overline{A}$.

In Fig. 4-7(b), one point x_1 belongs to fuzzy set *A* and to fuzzy set $\overline{A}$ with different membership values.

Definition 16: (a height of a fuzzy set)
The *height* of a fuzzy set is the largest membership grade (value) attained by any elements in that set.

Definition 17: (a normalized fuzzy set)
A fuzzy set is called *normalized* when at least one of its elements attains the maximum possible membership grade.
In Fig.4-8, (a) examples of the height and nonnormalized fuzzy set are shown.



(a)

(b)

Figure 4-8. Graphical representation of a height and an α *-cut* of a fuzzy set.

Definition 18: (α *-cut* of a fuzzy set)
An α *-cut* of a fuzzy set *A* is a crisp set A_α that contain all elements of the universal set *X* that have a membership grade in *A* greater than or equal to the specified value of α . This definition can be written as

$$A_\alpha = \{ x \in X \mid \mu_A(x) \geq \alpha \} .$$

In Fig. 4-8,(b) an α -cut of fuzzy set *A* is shown.

For example, in Table 4-2 for α = 0.2, the α -cut of fuzzy set "*young*" is the following crisp set:

$young_{0.2}$ = {5,10,20,30,40}.

Definition 19: (a *convex* fuzzy set)
A fuzzy set is *convex* if and only if each of its α - *cuts* is a convex set.
In Fig. 4-9 convex and nonconvex fuzzy sets are shown.

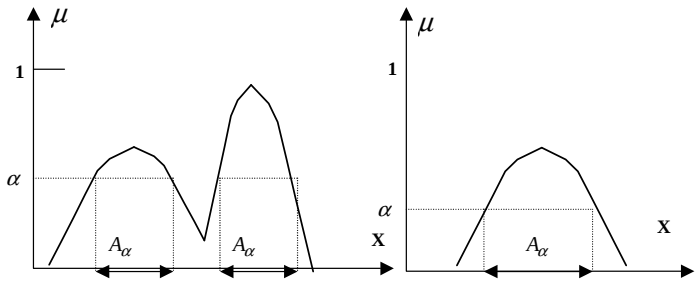


Figure 4-9. Nonconvex fuzzy set (a) and convex fuzzy set (b).

(Copia modificata per una migliore consultazione on-line)

Definition 20 (a fuzzy number)
A convex and normalized fuzzy set defined on R (a set of all real numbers) whose membership function is piecewise continuous is called a *fuzzy number*.

For example, the membership function given in Fig.4-10 can represent fuzzy numbers. You see here fuzzy number 50.

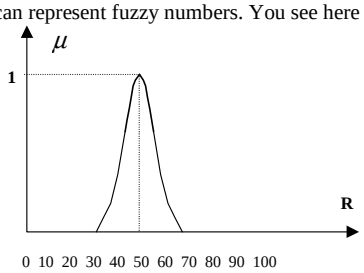


Figure 4-10. Fuzzy number 50.

Definition 21: (a support of a fuzzy set)

The support of fuzzy set *F* is the crisp set of all points $u \in U$ such that $\mu_F(u) > 0$

Definition 22: (a center of a fuzzy set)

The center of fuzzy set *F* is the point(s) $u \in U$ at which $\mu_F(u)$ achieves its maximum value.

Definition 23: (a fuzzy singleton)

If the support of fuzzy set F is a single point in *U* at which μ_F =1, then *F* is called the fuzzy singleton.

Linguistic Variables and Hedges

Consider now a formal description of linguistic variables in the fuzzy set theory.
If a variable can take words in natural language as its values, then this variable is defined as a linguistic variable. For example, the notion "size" may be considered as linguistic variable because its values can be described by the following natural language words: *small, big, not small-not big, very small*, etc. Linguistic variable can take either words or numbers values.
Consider another examples of linguistic variables. The linguistic variable "speed" can take the following values: *slow, medium, fast*, etc. It also can take any real numbers. Let us now introduce a formal definition of a linguistic variable.

Definition 24: (Linguistic variable)

Linguistic variable is characterized by a quintuple ($x, T(x), U, G, M$) in which *x* is the name of variable; $T(x)$ is the term set of *x*, that is, the set of names of linguistic values of *x* with each value being a fuzzy set defined on U ; G is a syntactic rule for generating the names of values of *x*; M is a semantic rule for associating each value with its meaning.
The linguistic variable is an important concept. It gives us a formal way to quantify linguistic descriptions about variables.
Let us describe the linguistic variable "speed" in accordance to the definition 24. In this case:
x is "a speed";
$T(x)$: "slow", "medium", "fast";
U : for example, [0-200] on a scale with a km/h unit;
G: introduced by a human expert a set of names, namely "slow", "medium", "fast";
M: description of fuzzy sets "slow", "medium", "fast" by introducing their membership functions.
In common human reasoning we also like to use such words as "very", "more or less", etc. How to define their meaning?
Definition 25: (Hedges)
Let *F* be a fuzzy set in *U* (for example, *F* = "small"). Then "very *F*" is called *a hedge of a fuzzy set* and defined as the fuzzy set in *U* with the following membership function:

$$\mu_{very}(u) = (\mu_F(u))^2 .$$

"More or less" is also considered as a hedge of a fuzzy set and defined as the fuzzy set in U with the following membership function

$$\mu_{more-or-less}(u) = (\mu_F(u))^{1/2} \text{, where } u \in U .$$

Comparison of representations by classical sets and fuzzy sets

Consider the following three sets: -a set of high persons; - a set of middle persons; and
- a set of low persons;
Let us describe them as crisp sets (a) and fuzzy sets (b), compare and discuss what representation is better.

(a) Crisp sets based representation (Fig.4-11):

$\mu_{Low} = \begin{cases} 1 & if \quad x < 170 \\ 0 & if \quad x \geq 170 \end{cases}$	$\mu_{Middle} = \begin{cases} 1 & if \quad 170 \leq x \leq 180 \\ 0 & else \end{cases}$	$\mu_{High} = \begin{cases} 1 & if \quad x > 180 \\ 0 & if \quad x \leq 180 \end{cases}$

(Copia modificata per una migliore consultazione on-line)

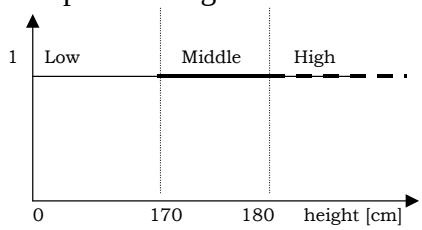


Figure 4-11. Crisp representation of set.

(b) Fuzzy sets based representation (Fig.4-12).

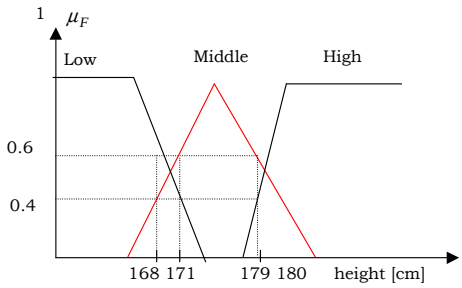


Figure 4-12. Fuzzy sets representation.

Suppose we have three persons with the following heights: Person A has 179 cm; person B has 171 cm; person C has 168 cm. What sets do they belong to? According to the classical (crisp) set representation we have the following Table 4-3.

Table 4-3

	Height	*Low*	*Middle*	*High*
A	179	0	1	0
B	171	0	1	0
C	168	1	0	0

According to the fuzzy set-based representation we have the Table 4-4.

Table 4-4

	Height	*Low*	*Middle*	*High*
A	179	0	0.6	0.4
B	171	0.4	0.6	0
C	168	0.6	0.4	0

Compare now these two representations: *what representation is better?*
According to Table 4-2 person A do not belong to the set of high persons, but according to Table 4-3 he belongs to that with grade 0.4.
You can see that the fuzzy sets based representation is *more flexible, more sensitive representation*.

Summarizing all mentioned above, we can say that the construction of a fuzzy set consists of following steps:
1) identification of a suitable universe of discourse; and
2) identification of the appropriate membership function.

Example of fuzzy sets application

In this example (taken from [11]), we will consider a computer-assisted matchmaking process for marriage using fuzzy sets. Suppose a client wants to find an ideal partner for marriage. He introduces what he wants: “neither young nor old” and “annual income must be of several thousands US dollars or more”.
Let be there are three potential candidates for marriage: B, C, and D. A database tells us their ages and incomes (see Table 4-5).

Table 4- 5

Name	*Age*	*annual Income (thousandUS$)*
B	38	100
C	32	50
D	48	20

Table 4-6

Name	*Age*	*Income*	*Total*
B	0.9	1.0	0.9
C	0.6	0.5	0.5
D	0.2	0	0

We can express the notions “young”, “old” and “several thousands dollars of income” by using fuzzy sets as shown in Fig.4-13(a),(c).

(Copia modificata per una migliore consultazione on-line)

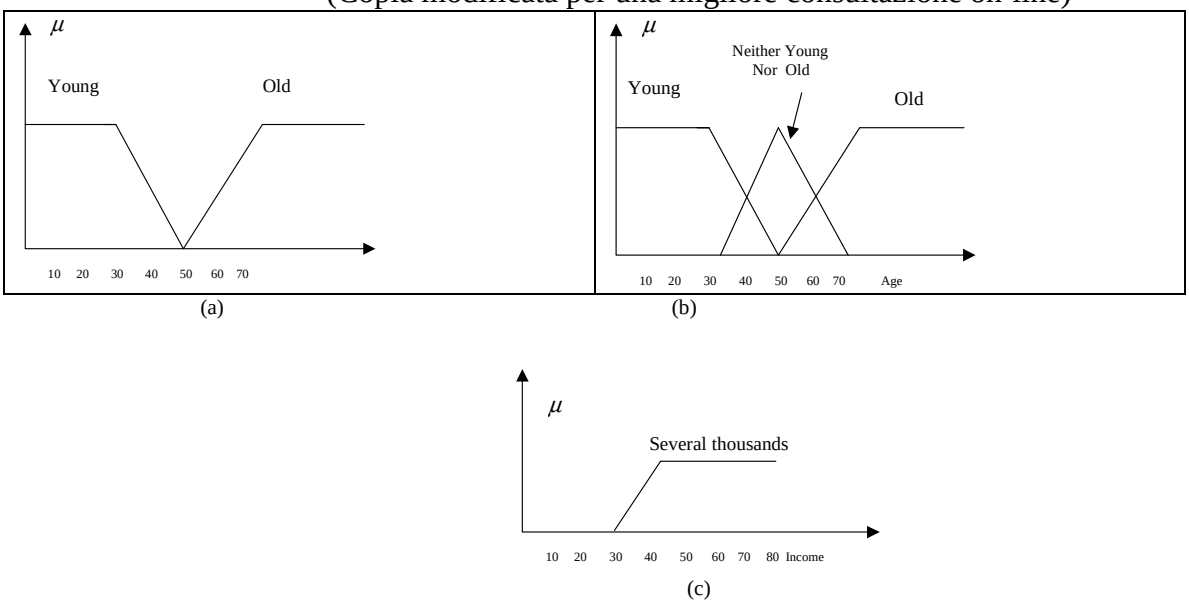


Figure 4-13. Membership functions images.

Because the client specifies "neither young nor old" we have to define a corresponding fuzzy set *F* (shown in Fig.3-1 (b)) as follows. We introduce fuzzy set Y as "not young", and fuzzy set Z as "not old", then our fuzzy set can be described as

$F = \overline{Y} \wedge \overline{Z}$ with $\mu_F = \mu_{\overline{Y} \wedge \overline{Z}} = \min((1-\mu_Y(x)),((1-\mu_Z(x))$.

Comparing fuzzy set *F* and the actual age of each candidate, we can obtain the membership value of the individual age. If we make the same to the income level, we will get Table 4-6. Here the *total evaluation* is obtained by the min-operation of membership values for age and income. Finally, you can see that a better candidate for the client is the candidate B.

Discussion about the nature of fuzzy sets and membership functions

Fuzzy sets deal with linguistic concepts, which are uncertain and imprecise. In many applications linguistic concepts are used for describing behavior of dynamic systems with unknown dynamic and structure.

Fuzzy sets are often incorrectly assumed to indicate some form of probability. It is important to note that the membership values are not probabilities.

Fuzzy set's theory and probability theory are very different approaches. Primary difference is the *subjectivity* and *nonrandomness* of fuzzy set.

The mathematical probability is described by values of a real numerical function defined on a class of idealized events, which represent results of an experiment or observation. The concept of mathematical probability deals with random processes and is related with a relative frequency of event occurrence.

The concept of a fuzzy set introduces another representation of uncertainty known as the fuzzy measure. The value of membership function, the *fuzzy measure*, indicates the degree of evidence or subjective certainty that element *x* belongs to the given fuzzy set.

The specification of membership function is quite subjective, and may be different for different persons. For example, fuzzy set "cold" may be described differently by different persons.

The subjectivity comes from the indefinite nature of abstract concepts and has nothing with probability, or randomness. It represents a *knowledge* of a given human expert in a given problem domain.

Finally, note main differences between crisp and fuzzy sets as follows.

Main differences between crisp and fuzzy sets

Classic(crisp)sets	*versus*	*Fuzzy sets*
$1. A = \{x\}, x \in U$ $\mu_A(x) = \begin{cases} 1, & if\ x \in A \\ 0, & otherwise \end{cases}$	$\Leftrightarrow$	$1. F = \{x, \mu_F(x)\}, x \in U,$ $\mu_F(x) \in [1,0]$
2. $\mu_A(x)$ *has only two values*	$\Leftrightarrow$	2. $\mu_F(x)$ *is many - valued*
3. A crisp set has a clear boundary	$\Leftrightarrow$	*3. A fuzzy set has a vague, fuzzy, boundary*
4. Laws of excluded middle and contradiction for crisp sets : $A \cup \overline{A} = X; A \cap \overline{A} = \phi$	$\Leftrightarrow$	*4. Laws of excluded middle and contradiction for fuzzy sets :* $A \cup \overline{A} \neq X, A \cap \overline{A} \neq \phi$

Lecture N 5 Fuzzy Numbers and Fuzzy Arithmetic Fuzzy Relations

Let us talk now about a "fuzzy calculation" such as "about 2" plus "about 3" or "about5" minus "about 2". When there is, for example, a relation $y = 3x + 2$ between *x* and *y*, the value of *y* for *x* = 4 can be calculated as $3 \times 4 + 2 = 14$. But how can we calculate the value of *y* when *x* is given by a fuzzy set such as *x* = "about 4"?

How to introduce such kind of a fuzzy arithmetic? The extension principle gives us a method for doing this.

Extension Principle

By introducing the extension principle, we can define various operations of fuzzy sets. Firstly, discuss necessary ideas to explain the extension principle [11].

Consider a mapping from a set *X* to another set *Y* such as $f : X \to Y$. Let *A* be a subset of *X*. Then

(Copia modificata per una migliore consultazione on-line)

$$f(A) = \{y | y = f(x), x \in A\} \qquad (5\text{-}1)$$

is called the *image of A by f*. Note that *f*(*A*) is a subset of *Y*.
Similarly, let *B* be a subset of *Y*. Then

$$f^{-1}(B) = \{x | f(x) = y, y \in B\} \qquad (5\text{-}2)$$

is called the *inverse image of B by f*. $f^{-1}(B)$ is a subset of *X*.

Relations (5-1) and (5-2) are defined for fuzzy sets *A* and *B* by the extension principle given as follows.

Extension Principle:

Extend mapping $f : X \to Y$ to relate fuzzy set *A* on *X* to fuzzy set *B* on *Y*:

$$\mu_{f(A)} = \begin{cases} \sup\limits_{y=f(x)} \mu_A(x), & f^{-1}(y) \neq \phi \\ 0 \quad , & f^{-1}(y) = \phi \end{cases} \qquad (5\text{-}3)$$

(Here the sign "ϕ" means the empty set).
When *f* is a one-to-one mapping, we can write Eq.(5-3) simply as

$$\mu_{f(A)}(y) = \mu_A(x). \qquad (5\text{-}4)$$

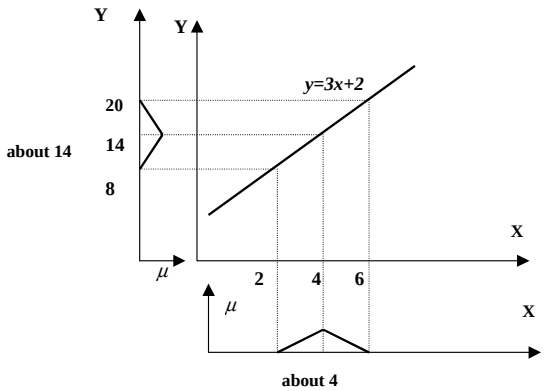
Example. Let us return to our example: how to find the value of *y* = $3x+2$ when *x* is given by a fuzzy set such as *x* = "about 4"?
Consider the mapping *y* = $f(x) = 3x+2$. Let *A* be the fuzzy set "about 4" described as

$$A = 0.5/3 + 1.0/4 + 0.5/5.$$

Define $x_1 = 3$, $x_2 = 4$, $x_3 = 5$ so that $y_i = 3x_i + 2, \quad i = 1,2,3$. Because *f* gives one-to-one mapping, we can apply Eq.(5-4) to get for the fuzzy set *A* as follows.

$$f(A) = \sum_{i=1}^{3} \mu_{f(A)}(y_i)/(y_i) = \sum_{i=1}^{3} \mu_A(y_i)/(3x_i + 2) =$$
$$= 0.5/(3\times3+2) + 1.0/(3\times4+2) + 0.5(3\times5+2) =$$
$$= 0.5/11 + 1.0/14 + 0.5/17 = "about\ 14"$$

Because $f(A)$ is a symmetrical fuzzy set with the membership value of 1 at 14, we can interpret this fuzzy set as "about 14".
Figure 5-1 shows the idea of the extension principle. The process of calculation can be interpreted as: $3\times"about\ 4" + 2 = "about\ 12" + 3 = "about\ 14"$.



.

Figure 5-1. The notion of extension principle.

Let us extend the extension principle to a general case. In the definition the notion of Cartesian product is used.

Definition 1 (Cartesian Product):
Let $x_1, x_2, ..., x_n$ be the elements of $X_1, X_2, ..., X_n$. The set of all combinations of ($x_1, x_2, ..., x_n$) is called the *Cartesian product* of sets $X_1, X_2, ..., X_n$ and is denoted as $X_1 \times X_2 \times ... \times X_n$.

Definition 2 (Cartesian Product of Fuzzy Sets):
Let $X_1 \times X_2 \times ... \times X_n$ be the Cartesian product of the universes $X_1, X_2, ..., X_n$ and let $A_1, A_2, ..., A_n$ be fuzzy sets on $X_1, X_2, ..., X_n$. The Cartesian product of fuzzy sets $A_1, A_2, ..., A_n$ can be defined by

(Copia modificata per una migliore consultazione on-line)

$$A_1 \times A_2 \times \ldots \times A_n = \int_{X_1 \times \ldots \times X_n} \min\{\mu_{A_1}(x_1), \ldots, \mu_{A_{n1}}(x_n)\} / (x_1, \ldots, x_n) \qquad (5\text{-}5)$$

on the universe $X_1 \times X_2 \times \ldots \times X_n$.

Extension Principle (general case):

Let f be a mapping from $X_1 \times X_2 \times \ldots \times X_n$ to satisfy $y = f(x_1, x_2, \ldots, x_n)$, that is,

$$f : X_1 \times X_2 \times \ldots \times X_n \to Y.$$

Extending the function *f*, we get the relation between the Cartesian product $A_1 \times A_2 \times \ldots \times A_n$ of fuzzy sets $A_1, A_2, \ldots, A_n$ and a fuzzy set *B* = $f(A_1 \times A_2 \times \ldots \times A_n)$ on *Y* such that

$$\mu_B(y) = \begin{cases} \sup\limits_{y = f(x_1, \ldots x_n)} \min\{\mu_{A_1}(x_1), \ldots \mu_{A_n}(x_n)\}, & f^{-1}(y) \neq \phi \\ 0 \quad , & f^{-1}(y) = \phi \end{cases}, \qquad (5\text{-}6)$$

where $f^{-1}(y)$ means the inverse image of *y*.

Refresh now the definition of fuzzy numbers and define operations on fuzzy numbers.

We call a fuzzy set A as a *fuzzy number* if it satisfies the following conditions (see Lecture 4, Def.4-20):

1. *A* is a convex fuzzy set defined on the universe *R* of real numbers;
2. There is only one x_0 that satisfies $\mu_A(x_0) = 1$;
3. μ_A is continuous in an interval.

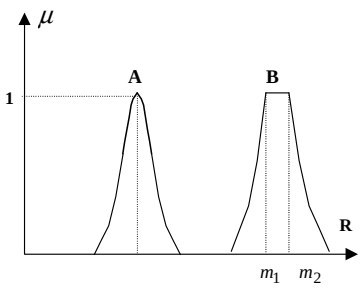


Figure 3-2. Fuzzy number *A* and Flat fuzzy number *B*.

Definition 3 (Flat Fuzzy Number):

If a fuzzy number *A* satisfies the following condition:

$$\begin{aligned} &(m_1, m_2) \in R \quad m_1 < m_2 \\ &\mu_A(x) = 1 \qquad \forall x \in [m_1, m_2] \end{aligned}, \qquad (5\text{-}7)$$

we will call it as a *flat fuzzy number*.

The difference between flat fuzzy numbers and fuzzy numbers is shown in Fig.3-2.

Operation of fuzzy numbers based on the extension principle

The binary operation $*$ of real numbers can be extended to binary operation ($*$) of fuzzy numbers *A* and *B* on the universe *X* such as

$$\mu_{A(*)B}(z) = \sup_{z = x * y} [\mu_A(x) \wedge \mu_B(y)] \qquad (5\text{-}8)$$

If we rewrite Eq.(5-8) using fuzzy sets, we get

$$A(*)B = \int_{X \times X} [\mu_A(x) \wedge \mu_B(y)] / (x * y) \qquad (5\text{-}9)$$

where $x, y, z \in X$.

Now using the preceding definition we can derive the arithmetic of fuzzy numbers.

Fuzzy Arithmetic

Addition:

$$\mu_{A \oplus B}(z) = \sup_{z = x + y} [\mu_A(x) \wedge \mu_B(y)] \qquad (5\text{-}10)$$

Subtraction:

$$\mu_{A \Theta B}(z) = \sup_{z = x - y} [\mu_A(x) \wedge \mu_B(y)] \qquad (5\text{-}11)$$

Multiplication:

(Copia modificata per una migliore consultazione on-line)

$$\mu_{A\otimes B}(z) = \sup_{z=x\times y} \left[\mu_A(x) \wedge \mu_B(y)\right] \qquad (5\text{-}12)$$

Division:

$$\mu_{A:B}(z) = \sup_{z=x/y} \left[\mu_A(x) \wedge \mu_B(y)\right] \qquad (5\text{-}13)$$

If an α - cut of a fuzzy set gives a closed interval, we can replace the preceding arithmetic of fuzzy numbers with operations of intervals:

$$[a,b] * [c,d] = \{z | z = x * y, x \in [a,b], y \in [c,d]\} \qquad (5\text{-}14)$$

When we choose + or – for the operation $*$, we get

$$[a,b] + [c,d] = [a+c, b+d] \qquad (5\text{-}15)$$

$$[a,b] - [c,d] = [a-d, b-c] \qquad (5\text{-}16)$$

For example,

$$[3,5] + [4,8] = [7,13],\ [3,5] - [4,8] = [-5,1].$$

Remark 1. Multiplication and division cannot be written in a general form as addition and subtraction.

If we assume $a,b,c,d > 0$, we can write

$$[a,b] \times [c,d] = [a \times c, b \times d] \qquad (5\text{-}17)$$

$$[a,b] : [c,d] = [a : d, b : c] \qquad (5\text{-}18)$$

For example,

$$[3,5] \times [4,8] = [12,40],\ [3,5] : [4,8] = [0.375, 1.25].$$

Remark 2. Note that in fuzzy arithmetic the subtraction is not an inverse operation of addition, and the division is not an inverse operation of multiplication. For example, if we subtract from a number the same number $(A - A)$, the result is not zero, but a fuzzy number "about 0".

Examples of fuzzy arithmetic operations

Example 1. Figure 5-3 shows the addition of fuzzy numbers "about 2" and "about 3".

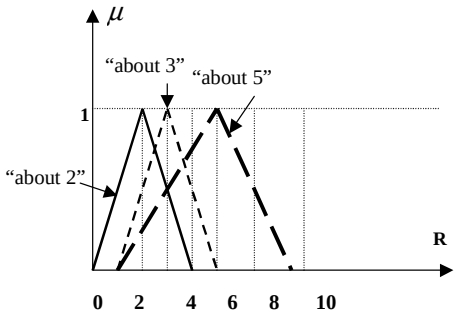


Figure 3-3. Addition of fuzzy numbers.

Remark 3. Notice that the fuzzy number obtained from the calculation of fuzzy numbers has an increased degree of fuzziness. The base of the resultant fuzzy number is wider than the original fuzzy numbers.

Example 2. Figure 5-4 shows the subtraction of fuzzy numbers "about 5" and "about 3".

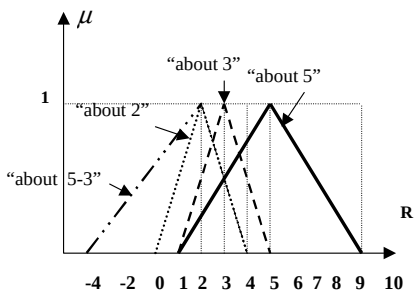


Figure 5-4. Subtraction of fuzzy numbers.

Remark 4. From the comparison of Figures 5-3 and 5-4, we can see that the result of subtraction "about 5" Θ "about 3" is not "about 2".
Figure 5-5 shows the subtraction of fuzzy numbers "about 3" and "about 3".

(Copia modificata per una migliore consultazione on-line)

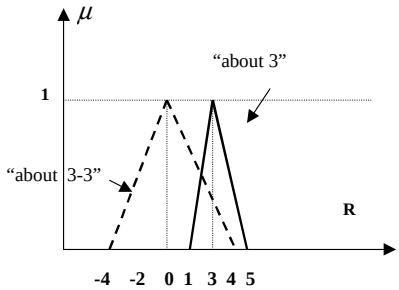


Figure 5-5. Subtraction of the same fuzzy numbers.

Remark 5. From Figure 5-5 we can see that "about 3" Θ "about 3" is not zero but "about 0" which is a fuzzy number.

L-R Fuzzy Numbers

Consider now a special class of fuzzy numbers called *L-R* fuzzy numbers [12].
Introduce functions *L* and *R* satisfying the conditions:

1) $L(x) = L(-x), R(x) = R(-x)$;
2) $L(0) = 1, R(0) = 1$;
3) both *L* and *R* are not increasing functions.

Definition 4 (*L-R* fuzzy number):
L-R fuzzy number *M* is defined by *L-R* functions such as

$$\mu_M(x) = \begin{cases} L\left(\dfrac{m-x}{\alpha}\right), & x \le m, \quad \alpha > 0 \\ R\left(\dfrac{m-x}{\beta}\right), & x > m, \quad \beta > 0 \end{cases} \qquad (5\text{-}19)$$

Here *L* and *R* are called *shape functions*, *m* is called the *mean value*, α and β define the length of the base in the triangular fuzzy set *M* to the left and to the right of the mean value, respectively (see, for example, Fig.5-6).
Functions *L* and *R* can be of any type as long as they satisfy conditions (1)-(3). Typical functions are:

$$L(x) = R(x) = \max(0, 1 - |x|^p), \; p \ge 0;$$

$$L(x) = R(x) = e^{-|x|^p}, \; p \ge 0;$$

$$L(x) = R(x) = 1/(1 + |x|^p), \; p \ge 0.$$

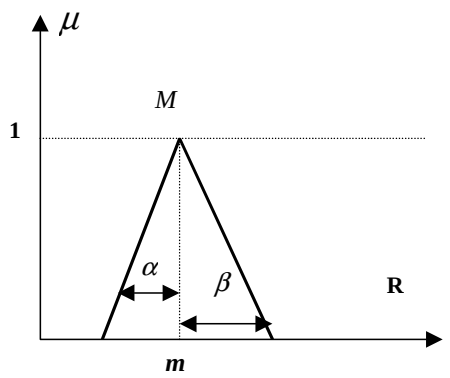


Figure 5-6. *L-R* fuzzy numbers representation.

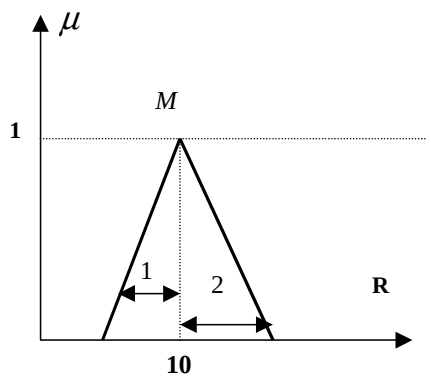


Figure 5-7. *L-R* fuzzy number (10,1,2).

Figure 5-7 shows the following *L-R* fuzzy number:

$$m = 10, \alpha = 1, \beta = 2, \quad \mu_M(x) = \begin{cases} \max\left\{0, 1 - \left(\dfrac{m-x}{\alpha}\right)\right\}, & x \le m, \quad \alpha > 0 \\ \max\left\{0, 1 - \left(\dfrac{m-x}{\beta}\right)\right\}, & x > m, \quad \beta > 0 \end{cases}.$$

(Copia modificata per una migliore consultazione on-line)

We will denote *L-R* fuzzy number *M* by the following simplified notation:

$$M = (m, \alpha, \beta)_{LR}.$$

The definition of *M* for *M* > 0 is $\mu_M(x) = 0$, $\forall x < 0$. The definition of *M* for *M* < 0 is $\mu_M(x) = 0$, $\forall x > 0$.

Operations on *L-R* fuzzy numbers

Addition:

$$(m, \alpha, \beta)_{LR} \oplus (n, \gamma, \delta) = (m + n, \alpha + \gamma, \beta + \delta)_{LR}.$$

Subtraction:

$$(m, \alpha, \beta)_{LR} \Theta (n, \gamma, \delta) = (m - n, \alpha + \gamma, \beta + \delta)_{LR}.$$
$$-(m, \alpha, \beta)_{LR} = (-m, \beta, \alpha)_{RL}.$$

Multiplication:
For *M* > 0 and *N* > 0

$$(m, \alpha, \beta)_{LR} \otimes (n, \gamma, \delta)_{LR} = (mn, n\alpha + m\gamma, n\beta + m\delta)_{LR}.$$

For *M* < 0 and *N* < 0

$$(m, \alpha, \beta)_{LR} \otimes (n, \gamma, \delta)_{LR} = (mn, -n\beta - m\delta, -n\alpha - m\gamma)_{RL}.$$

For *M* < 0 and *N* > 0

$$(m, \alpha, \beta)_{LR} \otimes (n, \gamma, \delta)_{LR} = (mn, n\alpha - m\delta, n\beta - m\gamma)_{RL}.$$

For $\lambda > 0$

$$\lambda \otimes (m, \alpha, \beta)_{LR} = (\lambda m, \lambda\alpha, \lambda\beta)_{LR}$$

For $\lambda < 0$

$$\lambda \otimes (m, \alpha, \beta)_{LR} = (\lambda m, -\lambda\beta, -\lambda\alpha)_{RL}.$$

Inversion:

$$(m, \alpha, \beta)_{LR}^{-1} = (m^{-1}, -\beta m^{-2}, -\alpha m^{-2})_{RL}.$$

Division:

This calculation is derived by the following approximation: $M : N = M \otimes N^{-1}$.

For *M* > 0 and *N* > 0

$$(m, \alpha, \beta)_{LR} : (n, \gamma, \delta)_{LR} = \left(\frac{m}{n}, \frac{m\delta + n\alpha}{n^2}, \frac{m\gamma + n\beta}{n^2}\right)_{LR}.$$

Another cases for *M* and *N* can be derived similarly.

Example 3. Figure 5-8 shows the result of addition of two fuzzy numbers $M = (2,1,2)_{LR}$ and $N = (3,2,1)_{LR}$, where shape functions defined as $L(x) = R(x) = \max(0, 1 - |x|)$. Applying the preceding formula we get: $M \oplus N = (2,1,2)_{LR} \oplus (3,2,1)_{LR} = (5,3,3)_{LR}$.

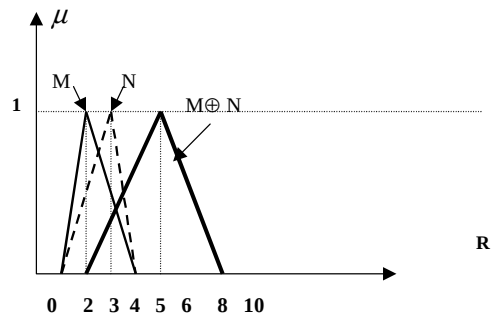


Figure 5-8. Addition of *L-R* fuzzy numbers.

Example 4: Fuzzy numbers based decision making

Consider a businessman traveling from Rome to Milan. Let the following four ways of transportation are evaluated.

(Copia modificata per una migliore consultazione on-line)

N	*Way - of - transportation*	*Cost (100 US$)*	*Time (hours)*	*Remarks*
1	*bullet train*	2.4	4	
2	*express train*	1.9	6	
3	*airplane*	2.8	2	*includes transit to airport*
4	*car*	2.7	6 – 8	*includes gas and highway toll*

The businessman chooses the best way of transportation based on the following calculation:

(evaluation) = (time) + (cost).

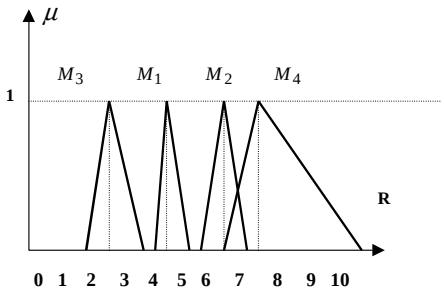
The best way is one with smaller evaluation's value.

The duration of each way of transportation does not include delays, a time required for transfer, and so on. If we include such factors, we can express this time as *L-R* fuzzy numbers, for example, as follows.

$M_1 = (4.5,\ 0.5,\ 0.5)_{LR}$, $M_2 = (6.5,\ 0.5,\ 0.5)_{LR}$, $M_3 = (2.5,\ 0.5,\ 1.0)_{LR}$,

$M_4 = (7.5,\ 1.0,\ 3.0)_{LR}$.

These fuzzy numbers are shown in Fig.5-9(a).



(a)

Figure 5-9 (a). *L-R* fuzzy numbers for ways of transportation.

Remark 6. *L-R* fuzzy numbers for the airplane and car have wider ranges of values compared with those of trains. The time may vary for a car depending on road traffic conditions and airplanes tend to be late compared to trains.

Although the costs are definite values, they are assumed as *L-R* fuzzy numbers with zero range. However, from the definition of *L-R* fuzzy numbers, $\alpha, \beta > 0$. Therefore we will use a very small value 10^{-5} as an alternative for the range.

$N_1 = (2.4,\ 10^{-5},\ 10^{-5})_{LR}$, $N_2 = (1.9,\ 10^{-5},\ 10^{-5})_{LR}$, $N_3 = (2.8,\ 10^{-5},\ 10^{-5})_{LR}$,

$N_4 = (2.7,\ 10^{-5},\ 10^{-5})_{LR}$.

The evaluation can be defined as a fuzzy number by the following way:

(fuzzy number of evaluation) = (fuzzy number for time) $\oplus$ (fuzzy number for costs).

So, we have:

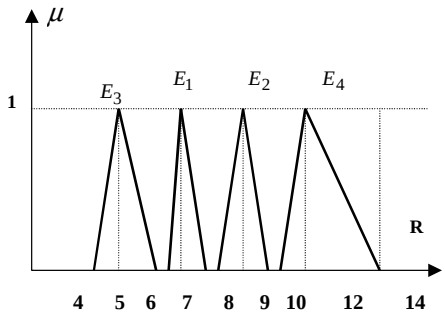
Way 1 of transportation: $E_1 = M_1 \oplus N_1 = (6.9,\ 0.5,\ 0.5)_{LR}$;

Way 2 of transportation: $E_2 = M_2 \oplus N_2 = (8.5,\ 0.5,\ 0.5)_{LR}$;

Way 3 of transportation: $E_3 = M_3 \oplus N_3 = (5.3,\ 0.5,\ 1.0)_{LR}$;

Way 4 of transportation: $E_4 = M_4 \oplus N_4 = (10.2,\ 0.5,\ 3.0)_{LR}$.

These fuzzy numbers are shown in Fig.5-9(b).



(b)

Figure 5-9 (b). *L-R* fuzzy numbers for evaluation

The result of businessman's decision making based on fuzzy numbers is the best choice of transportation way, which is Way 3.

Example 5. Qualitative Spatial Reasoning Based on Fuzzy Numbers

The term *spatial reasoning* refers in general to reasoning about problems dealing with entities occupying space. Spatial information is often imprecise and approximate, so researchers have to represent the fuzziness in spatial data. Qualitative spatial reasoning about space forms n integral part of our daily life. This

kind of reasoning can be important for intelligent autonomous agents (e.g., robots), especially for those operating in uncertain or unknown or dynamic environments. In such situations, several factors can enhance the benefits of using qualitative reasoning techniques:

- it may be difficult (or often impossible) to collect precise information about the environment;
- there may be real constraints of memory and time which prevent either the collection of large volumes of data or the utilization of a large amount of computation time;
- in hostile environments, quick reactions to sudden stimuli may be facilitated by qualitative reasoning;
- man-machine interaction can be enhanced (e.g., instructions to robot can be given in natural language).

The focus is on defining a suitable computational framework for correctly representing qualitative spatial information. Consider, for example, the following spatial information: "Object A is about 5 miles from object B in a north-easterly direction" (see Fig.5-10).

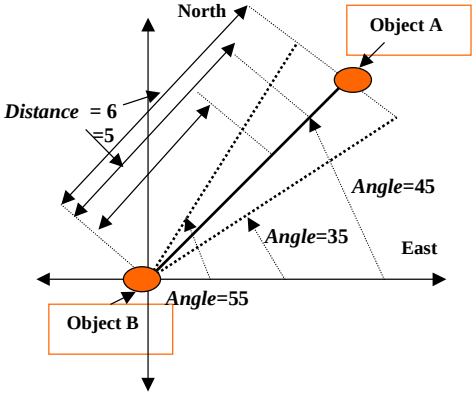


Figure 5-10. Fuzziness in location of object A relative to object B.

We can represent relative positions of objects A and B by using fuzzy numbers as follows (see Fig.5-11).

About 5 miles = $M = (5,1,1)_{LR}$ and north-easterly direction = $N = (45,10,10)_{LR}$.

Assume that we have the following spatial descriptor statement:

Object A is about 2 miles east of object B, and object C is more or less 3 miles of object B. (Note that in these two statements we are only concerned with the one-dimensional case.) How to find the answer on the following spatial query: *What is the spatial relation between objects A and C*?

Assume that fuzzy sets "about 2 miles" and "more or less 3 miles" are represented by fuzzy numbers $M = (2,1,1)_{LR}$ and $N = (3,1,2)_{LR}$ respectively. Then unknown spatial relation between objects *A* and *C* can be calculated as $M \oplus N = (5,2,3)_{LR}$ which is to be interpreted as saying that object *C* is about 5 miles east of object *A*.

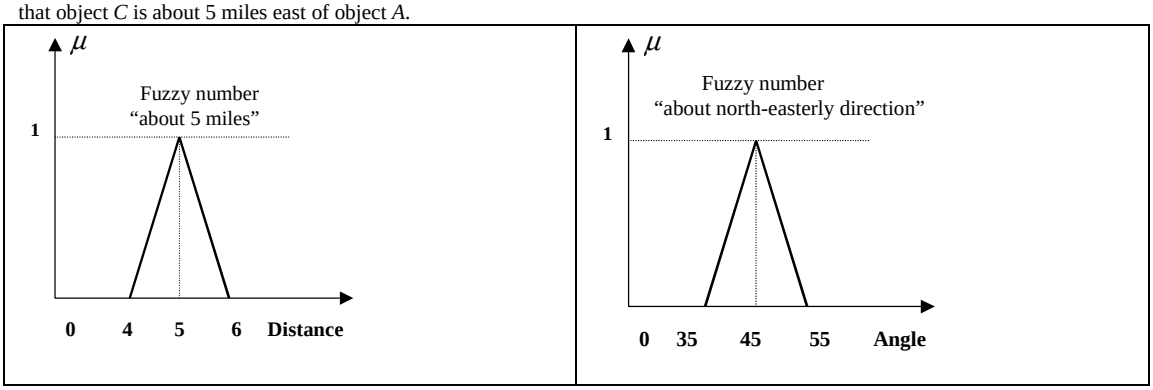


Figure 5-11. Relative positions representations as fuzzy numbers.

Fuzzy Relations

A *fuzzy relation* is an extension of a classical relation. Classical relations are given by classical, crisp, sets, fuzzy relations are given by fuzzy sets.

Crisp (classical) relation R is defined on the Cartesian product of two (or more) universal sets $X \times Y$ such that:

$$\mu_R(x,y) = \begin{cases} 1, & if \quad (x,y) \in R \\ 0, & otherwise \end{cases}$$

Binary (or more) crisp relation is given on 2-dimensional (or multidimensional) space. So, the binary crisp relation represents a 2-dimensional crisp set with a definite boundary, and the binary fuzzy relation represents a 2-dimensional fuzzy set with a vague boundary.

In Fig.5-12 examples of crisp and fuzzy relations are shown.

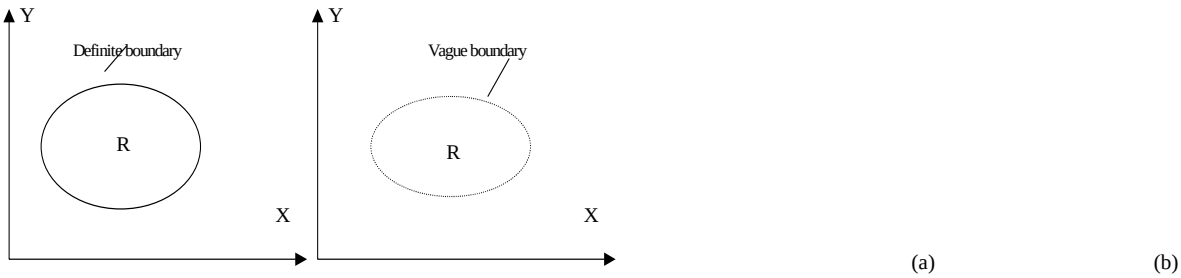


Figure 5-12. Image of binary relations: (a) a crisp relation; (b) a fuzzy relation.

(Copia modificata per una migliore consultazione on-line)

If we consider a definition of a simple fuzzy set, you may note that the fuzzy set is defined on one universe of discourse *X*. (see Fig.5-13 (a)).

A fuzzy relation considered as an extension of a crisp relation is a fuzzy set given on a multidimensional space (which is the Cartesian product of two (or more) universal sets).

Let us introduce now formal definitions of fuzzy relations for discrete and continuous cases.

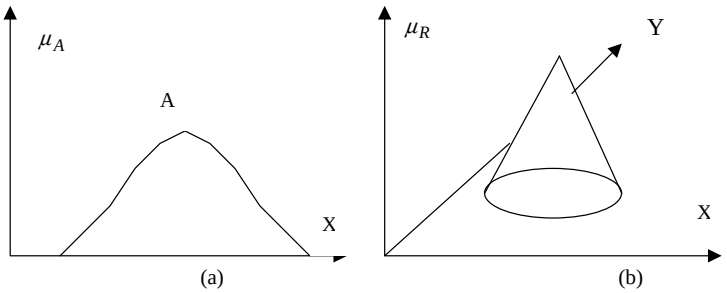


Figure 5-13. (a) Image of a fuzzy set; (b) Image of a fuzzy relation.

Definition 5: (Binary fuzzy relation (discrete case))

Let the universes *X* and *Y* are given as

$$X = \{x_1, \ldots, x_n\} \text{ and } \quad Y = \{y_1, \ldots, y_m\}$$

The binary fuzzy relation given on *X* and *Y* is expressed in the matrix expression:

$$R = \begin{array}{c|ccccc} & y_1 & y_2 & \cdots & y_{m-1} & y_m \\ \hline x_1 & \mu_R(x_1,y_1) & \mu_R(x_1,y_2) & \cdots & \mu_R(x_1,y_{m-1}) & \mu_R(x_1,y_m) \\ x_2 & \mu_R(x_2,y_1) & \mu_R(x_2,y_2) & \cdots & \mu_R(x_2,y_{m-1}) & \mu_R(x_2,y_m) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n & \mu_R(x_n,y_1) & \mu_R(x_n,y_2) & \cdots & \mu_R(x_n,y_{m-1}) & \mu_R(x_n,y_m) \end{array}$$

where $\mu_R(x_i, y_j) \in [0,1]$.

A matrix expressing a fuzzy relation is sometimes called a *fuzzy matrix*.

Definition 6: (Binary fuzzy relation (continuous case))

Let the universes *X* and *Y* be continuos. The binary fuzzy relation given on Cartesian product of $X \times Y$ is defined as $R = \int_{X \times Y} \mu_R(x,y)/(x,y)$,

where μ_R is the membership function of *R* given on the $X \times Y$ as $\mu_R : X \times Y \to [0,1]$ (see Fig.5-13,b).

Definition 7: (*n*- ary fuzzy relation (continuous case))

Let the universes $X_1, X_2, \ldots, X_n$ be continuos. The *n*-ary fuzzy relation given on Cartesian product of $X_1 \times X_2 \times .. \times X_n$ is defined as

$$R = \int_{X_1 \times \ldots \times X_n} \mu_R(x_1, \ldots, x_n)/(x_1, \ldots, x_n),$$

where μ_R is the membership function of *R* given on the $X_1, X_2, \ldots, X_n$ as $\mu_R : X_1 \times X_2 \times .. \times X_n \to [0,1]$.

Examples of fuzzy relations

Example 1. Consider two crisp relations: "*y* equals *x*" and "*y* is smaller than *x*". In both case, boundaries are explicitly defined (see Fig.5-14 (a),(b)).

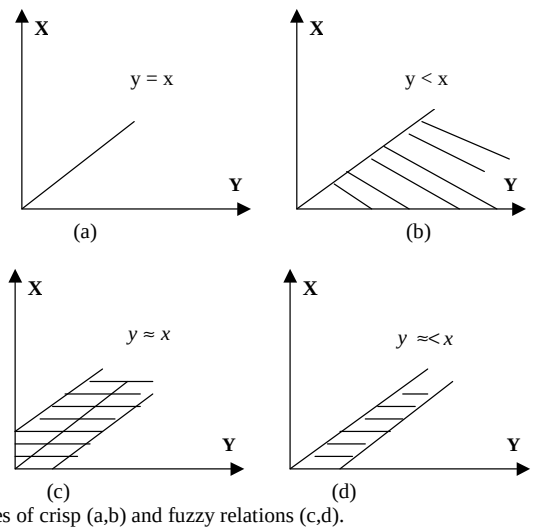


Figure 5-14: Images of crisp (a,b) and fuzzy relations (c,d).

On the other side, we cannot define the boundaries in such cases as: "*y* is about *x*" and "*y* is a little smaller than *x*". These relations are fuzzy relations (see Fig.5-14 (c),(d)).

(Copia modificata per una migliore consultazione on-line)

Example 2. Let us choose three cities x_1, x_2, x_3 in Cremona area and three cities y_1, y_2, y_3 from adjacent region Bologna.

Consider these cities as elements of sets $X = \{x_1, x_2, x_3\}$ and $Y = \{y_1, y_2, y_3\}$ respectively.

We may introduce the fuzzy relation R = "close" by the following fuzzy matrix:

$$R = \begin{array}{c|ccc} & y_1 & y_2 & y_3 \\ \hline x_1 & 1 & 0.6 & 0.3 \\ x_2 & 0.4 & 0.9 & 0.1 \\ x_3 & 0.5 & 0.2 & 0.7 \end{array}$$

You see that according to this fuzzy relation the degree of closeness between two cities x_1 and y_2 is 0.6, and so on.

Projection of Fuzzy Relations and Cylindrical Extensions of Fuzzy Sets

Definition 8: (Projections of a binary fuzzy relation (continuous case))

Let R be a fuzzy binary fuzzy relation given on Cartesian product of $X \times Y$. The *projection of R on X* is defined as

$$proj[R; X] = \int_X (\max_y \mu_R(x, y)) / x$$

The *projection of R on Y* is defined as

$$proj[R; X] = \int_Y (\max_x \mu_R(x, y)) / y$$

Figure 5-15 shows the ideas of projection and cylindrical extension. Obviously, the projection and cylindrical extension are opposite operations.

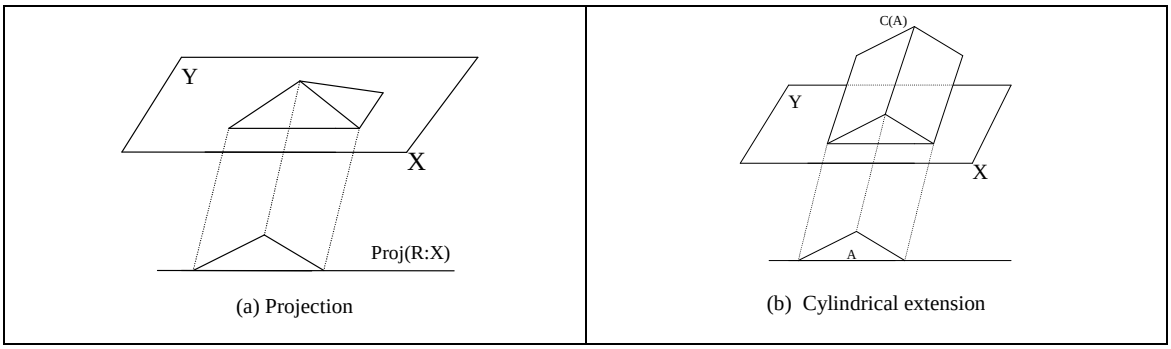


Figure 5-55. Graphical images of Projection (a) and Cylindrical extension (b) operations.

Definition 9: (Cylindrical extension of fuzzy set (continuous case))

Let A be a fuzzy set on the universe X. The *cylindrical extension* $c(A)$ *of fuzzy set* A given on $X \times Y$ is defined by the fuzzy relation

$$c(A) = \int_{X \times Y} \mu_A(x) / (x, y)$$

Let B be a fuzzy set on the universe Y. The *cylindrical extension* $c(B)$ *of fuzzy set* B given on $X \times Y$ is defined by the fuzzy relation

$$c(B) = \int_{X \times Y} \mu_A(y) / (x, y).$$

Definition 10: (Projections of n- ary fuzzy relation (continuous case))

Let R be a fuzzy binary fuzzy relation given on Cartesian product of $X_1 \times X_2 \times \ldots \times X_n$, and array $(i_1, \ldots, i_k)$ be a subarray of $(1, \ldots, n)$. The *projection of R on* $X_{i_1}, \ldots, X_{i_k}$ is defined as

$$proj\lfloor R; X_{i_1}, \ldots, X_{i_k} \rfloor = \int_{X_{i_1} \ldots X_{i_k}} (\max_{x_{j_1} \ldots x_{j_m}} \mu_R(x_1, \ldots, x_n) / (x_{i_1}, \ldots, x_{i_k}),$$

where the array $(j_1, \ldots, j_m)$ is the complementary of the array $(i_1, \ldots, i_k)$, that is, the subarray of $(1, \ldots, n)$ with $(i_1, \ldots, i_k)$ subtracted.

Definition 11: (Cylindrical extension of n-ary fuzzy relation (continuous case))

Let R be a fuzzy relation given on the Cartesian product $X_1 \times X_2 \times \ldots \times X_m$. The *cylindrical extension* $c(R)$ *of R* given on $X_1 \times X_2 \times \ldots \times X_n$ $(m < n)$ is defined by the fuzzy relation

$$c(R) = \int_{X_1 \times \ldots \times X_n} \mu_A(x_1, \ldots, x_m) / (x_1, \ldots, x_n)$$

Example of Projection and Cylindrical Extension Operations. Suppose the universes X and Y are given as

(Copia modificata per una migliore consultazione on-line)

$X = \{x_1, x_2, x_3\}$ and $Y = \{y_1, y_2, y_3\}$.

Now let us consider a fuzzy relation R on $X \times Y$ such as:

$$R = \begin{array}{c|ccc} & y_1 & y_2 & y_3 \\ \hline x_1 & 0.6 & 1 & 0.3 \\ x_2 & 0.5 & 0.2 & 0.8 \\ x_3 & 0.1 & 0.4 & 0.7 \end{array}$$

The projection of R on X is then calculated as

$$\text{Proj}[R, X] = \max(0.6,1,0.3)/x_1 + \max(0.5,0.2,0.8)/x_2 + \max(0.1,0.4,0.7)/x_3 =$$

$$= 1/x_1 + 0.8/x_2 + 0.7/x_3 = A. \qquad (5\text{-}20)$$

(Here A is the fuzzy set projected on X).
On the other hand, the projection of R on Y is calculated as

$$\text{Proj}[R,\text{Y}] = \max(0.6,0.5,0.1)/y_1 + \max(1,0.2,0.4)/y_2 + \max(0.3,0.8,0.7)/y_3 =$$

$$= 0.6/y_1 + 1/y_2 + 0.8/y_3 = B. \qquad (5\text{-}21)$$

The cylindrical extension of the fuzzy set A on the Cartesian product $X \times Y$ (c(A)) and the cylindrical extension of the fuzzy set B on the Cartesian product $X \times Y$ (c(B)) are as follows:

$c(A) = \begin{array}{c\|ccc} & y_1 & y_2 & y_3 \\ \hline x_1 & 1 & 1 & 1 \\ x_2 & 0.8 & 0.8 & 0.8 \\ x_3 & 0.7 & 0.7 & 0.7 \end{array}$	$c(B) = \begin{array}{c\|ccc} & y_1 & y_2 & y_3 \\ \hline x_1 & 0.6 & 1 & 0.8 \\ x_2 & 0.6 & 1 & 0.8 \\ x_3 & 0.6 & 1 & 0.8 \end{array}$

Now let us discuss the meaning of projection operation. Figure 5-16 shows the idea of projection.

height – of – shadow __0.6__1____0.8___*Y-axis*

↑↑ ↑ ↑↑ ↑↑↑

$$R = \begin{array}{c|ccc} & y_1 & y_2 & y_3 \\ \hline x_1 & 0.6 & 1 & 0.3 \\ x_2 & 0.5 & 0.2 & 0.8 \\ x_3 & 0.1 & 0.4 & 0.7 \end{array}$$

↑ ↑ ↑ ↑

a source of light

Figure 5-16. An idea of projection.

Assume the values of matrix elements (membership value) show the height of objects. By illuminating the objects in the direction shown in Fig.6, we will get shadows on a wall on the Y-axis, and the shadows portray the fuzzy sets obtained by the projection. Naturally, the shadows reflect the height of the highest objects in a column. This equals the max-operation in (5-20),(5-21).

Intersection of Fuzzy Sets with Different Universals

The main point of this paragraph is how we can obtain an intersection of fuzzy sets representing, for example, “moderate height” and “middle age”. Can we express the intersection of “moderate height” and “middle age” as

“moderate height” $\cap$ “middle age”?

If this operation could be defined, it would be something like that shown in Fig.5-17. However, because the horizontal axes are different for the height and the age, we cannot formally derive such an operation as “moderate height” $\cap$ “middle age”. In this case we go by the following way.
When we have fuzzy sets on different universes, such as “height” and “age”, we can apply an operation depicted in Fig.5-18.

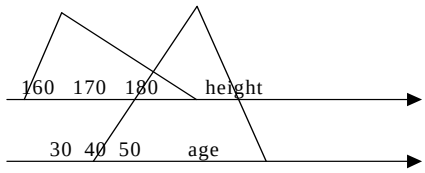
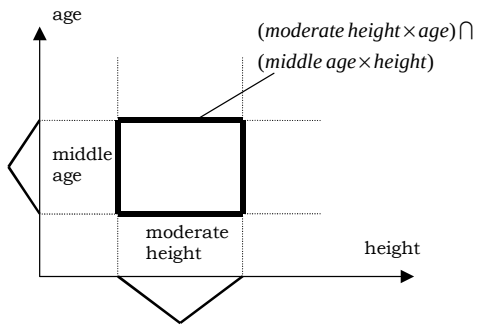


Figure 5-17. Intersection of “moderate height” $\cap$ “middle age”.



(Copia modificata per una migliore consultazione on-line)

Figure 5-18. Intersection of fuzzy relations.

Note that we have the universes “height” and “age” and we need to think of their intersection in the Cartesian space of $height \times age$. Because the fuzzy set “moderate height” is not related to age, we make a cylindrical extension of “moderate height” towards $height \times age$ space. Let us denote this extension as

$$c(\text{moderate height}) = moderate\ height \times age.$$

Similarly, we make a cylindrical extension of “middle age” towards $height \times age$ space and obtain

$$c(\text{middle age}) = middle\ age \times height.$$

Both these cylindrical extensions are fuzzy sets on $height \times age$ and now we can take the intersection of these two. Therefore we obtain the operation of

$$(moderate\ height \times age) \bigcap (middle\ age \times height).$$

In order to introduce a fuzzy logic notion and fuzzy reasoning models let us refresh main ideas of classical logic and inference.

Lecture N 6 Fuzzy Logic

In order to introduce a fuzzy logic notion and fuzzy reasoning models let us refresh main ideas of classical and nonclassical logics.

Any formal logical system is represented by the sets $< A, S, P, R >$, where *A* is an alphabet, *S* is a syntax, P is the set of *axioms*, *R* is the set of *inference rules*, and by *interpretation model IM* (or semantics). Interpretation model *IM* is a triple <*Z, H, V*>, where *Z* is the set of interpretable values; *H* are the rules of mapping of elements of alphabet onto Z;

V is the interpretation function by which any well formed formula (*WFF*) *F* has some value V(*F*) $\in$ Z. For example, the interpretation function for classical logical systems is shown in Table 2-1 (see Lecture 2).

Nonclassical logics overview

The basic assumption of classical logic is two-valued, that is every WFF is either true or false (1 or 0).

Nonclassical logics are extension of classical logic. There are different kinds of extension. Extension of two-valued into three-valued logic: truth (1), false (0) and indeterminacy (1/2).

Five of three-valued logics are very famous (see Table 6-1).

Table 6-1

a b	*Lukasiewicz*				*Bochvar*				*Kleene*				*Heyting*				*Reichenbach*			
	$\wedge$	$\vee$	$\Rightarrow$	$\Leftrightarrow$	$\wedge$	$\vee$	$\Rightarrow$	$\Leftrightarrow$	$\wedge$	$\vee$	$\Rightarrow$	$\Leftrightarrow$	$\wedge$	$\vee$	$\Rightarrow$	$\Leftrightarrow$	$\wedge$	$\vee$	$\Rightarrow$	$\Leftrightarrow$
0 0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
0 $\frac{1}{2}$	0	$\frac{1}{2}$	1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	0	$\frac{1}{2}$	1	$\frac{1}{2}$	0	$\frac{1}{2}$	1	0	0	$\frac{1}{2}$	1	$\frac{1}{2}$
0 1	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
$\frac{1}{2}$ 0	0	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	0	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	0	$\frac{1}{2}$	0	0	0	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$
$\frac{1}{2}$ $\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	1	1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	1	1	$\frac{1}{2}$	$\frac{1}{2}$	1	1
$\frac{1}{2}$ 1	$\frac{1}{2}$	1	1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	1	1	$\frac{1}{2}$	$\frac{1}{2}$	1	1	$\frac{1}{2}$	$\frac{1}{2}$	1	1	$\frac{1}{2}$
1 0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0
1 $\frac{1}{2}$	$\frac{1}{2}$	1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	1	$\frac{1}{2}$	$\frac{1}{2}$
1 1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

They introduce different interpretation of logical operations. For example, Lukasievich Logic uses the following equations for logical operations interpretation:

$\bar{a} = 1 - a$	$a \Rightarrow b = \min(1, 1 + b - a)$
$a \wedge b = \min(a, b)$	$a \Leftrightarrow b = 1 - \lvert a - b \rvert$
$a \vee b = \max(a, b)$	

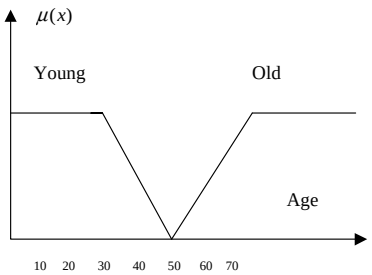
There are *many-valued* logics, where truth values are defined by the following set T_n:

$$T_n = \left\{ 0, \frac{1}{n-1}, \frac{2}{n-1}, \ldots, \frac{n-2}{n-1}, \frac{n-1}{n-1} = 1 \right\}.$$ These values can be interpreted as *degrees of truth*.

There are *infinite-valued logics* where truth values are represented by all the real numbers in the interval [0,1].

The *fuzzy logic* may be considered as the extension of an infinite-many-valued logic. Its ultimate goal is to provide foundations for *approximate reasoning with imprecise propositions* using fuzzy set theory as the principle tool. This is analogous to the role of quantified predicate logic for reasoning with precise propositions. In order to deal with imprecise propositions, fuzzy logic allows the use of *fuzzy predicates*.

Each simple fuzzy predicate such as “ X is *P* ” is represented by a fuzzy set. Assume, for example, that *x* stands for the age of a person and *P* has the meaning of *young*. Then, assuming that the universal set is the set of integers from 0 to 70 representing different ages, the predicate “ X is *P* ” may be represented by a fuzzy set whose membership function is shown, for example, in Fig.6 –1.



(Copia modificata per una migliore consultazione on-line)

Fig.6–1. Membership values of fuzzy predicates.

Consider now the truth value of a proposition obtained by a particular substitution for *x* into predicate, such as "Sergei is young". The truth value of this proposition is defined by the membership grade of Sergei's age (for example, his age is 25 years old) in the fuzzy set chosen to characterize the concept of a young person (Fig.6 –1) .

Fuzzy logic operations can be introduced, by different ways, for example, as follows.

Definition 1: (Fuzzy conjunction (or fuzzy AND))

The truth value of fuzzy conjunction $A \wedge B$ is given by

$$\mu_{A \wedge B}(x) = \min(\mu_A(x), \mu_B(x)),$$

where $\mu_A(x), \mu_B(x)$ are truth values of fuzzy predicates A, B respectively.

Definition 2: (Fuzzy disjunction (or fuzzy OR))

The truth value of fuzzy disjunction $A \vee B$ is given by

$$\mu_{A \vee B}(x) = \max(\mu_A(x), \mu_B(x)),$$

where $\mu_A(x), \mu_B(x)$ are truth values of A, B respectively.

Definition 3: (Fuzzy negation)

The truth value of fuzzy negation $\overline{A}$ (or $\neg A$) is given by

$$\mu_{\overline{A}}(x) = 1 - \mu_A(x),$$

where $\mu_A(x)$ is truth value of A.

Remark 1. Note that another definitions for fuzzy AND and fuzzy OR have been proposed in the literature. These operations are also called as *"T-norm"* and *"T-conorm"* respectively (see Appendix to the lecture). A popular alternative for AND and OR are:

$$\mu_{A \wedge B}(x) = \mu_A(x)\mu_B(x),$$

$$\mu_{A \vee B}(x) = \mu_A(x) + \mu_B(x) - \mu_A(x)\mu_B(x)$$

Fuzzy Implication and Fuzzy Reasoning

There are a few terms for the *fuzzy implication*. It is called also as a *fuzzy if-then rule*, or a *fuzzy rule*, or a *fuzzy conditional statement*. Further we will use the terms *fuzzy if - then rule* or *fuzzy rule*. A fuzzy rule assumes the following expression:

$$\text{IF } x \text{ is } A \text{ THEN } y \text{ is B}, \qquad (6\text{-}1)$$

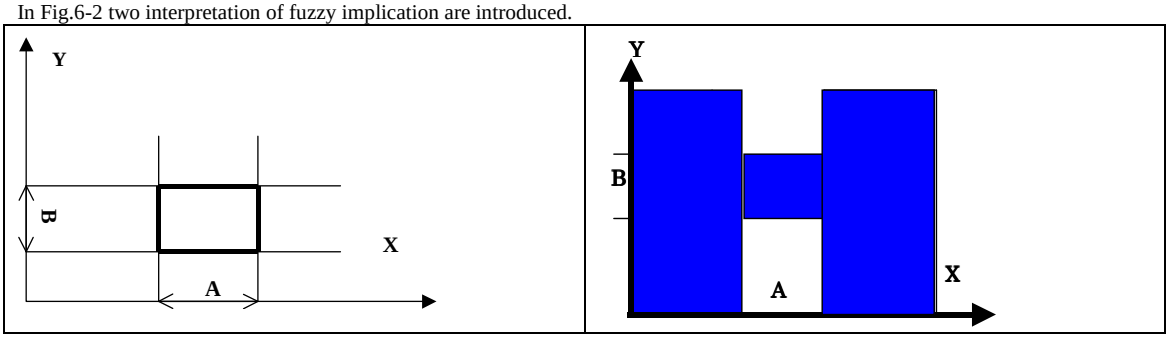
where *A* and *B* are linguistic variables defined by fuzzy sets on universes of discourse *X* and *Y*, respectively. (We will write this rule as $R = A \rightarrow B$).

The part "IF" (x is A) is called the *antecedent, or premise*. The part "THEN" (y is B) is called the *consequence or conclusion*.

From (6-1) you can see that a fuzzy rule can be defined as *binary fuzzy relation R* on the product space $X \times Y$ ($(x, y) \in X \times Y$) with some membership value $\mu_R(x, y)$. So, *R* is a two-dimensional fuzzy set characterized by a two-dimensional membership function $\mu_R(x, y)$.

There are a few ways to interpret fuzzy rule *R*, that is, fuzzy implication.

In Fig.6-2 two interpretation of fuzzy implication are introduced.



(a) (b)

Figure 6-2. Two interpretation of fuzzy implication.

First interpretation (Fig.6-2,a) is as " A coupled with B", the second one (Fig.6-2,b) is as material implication or Boolean logic implication :

$R = A \rightarrow B = \neg A \vee B$.

Further we will use the first interpretation of *R* as " A coupled with B". Then fuzzy set *R* can be described as

$$R = A \rightarrow B = A \times B = \int_{X \times Y} \mu_A(x) * \mu_B(y) / (x, y), \qquad (6\text{-}2)$$

where * is a *fuzzy AND* operator, that is,

$$\mu_R(x, y) = \mu_A(x) \wedge \mu_B(y). \qquad (6\text{-}3)$$

Interpretation of fuzzy implication given in (6-3) is called Mamdani implication.

There are another methods of fuzzy implication interpretations as follows:

Lukasiewicz's implication:

(Copia modificata per una migliore consultazione on-line)

$$\mu_R(x,y) = 1 \wedge (1-\mu_A(x)+\mu_B(y));$$

Algebraic product (or Larsen implication):

$$\mu_R(x,y) = \mu_A(x)\cdot\mu_B(y);$$

Bounded product:

$$\mu_R(x,y) = 0 \vee (\mu_A(x)+\mu_B(y)-1);$$

Zadeh implication:

$$\mu_R(x,y) = (\mu_A(x)\wedge\mu_B(y)) \vee (1-\mu_A(x));$$

Standard implication:

$$\mu_R(x,y) = \begin{cases} 1 & \mu_A(x) \le \mu_B(y) \\ 0 & \mu_A(x) > \mu_B(y) \end{cases};$$

Kleen-Dienes implication:

$$\mu_R(x,y) = (1-(\mu_A(x)) \vee \mu_B(y);$$

Drastic Product:

$$\mu_R(x,y) = \begin{cases} \mu_A(x) & \mu_B(y)=1 \\ \mu_B(y) & \mu_A(x)=1 \\ 0 & \mu_A(x)<1, \quad \mu_B(y)<1 \end{cases};$$

Goedel implication:

$$\mu_R(x,y) = \begin{cases} 1 & \mu_A(x) \le \mu_B(y) \\ \mu_B(y) & \mu_A(x) > \mu_B(y) \end{cases};$$

Gougen implication:

$$\mu_R(x,y) = \begin{cases} 1 & \mu_A(x) \le \mu_B(y) \\ \mu_B(y)/\mu_A(x) & \mu_A(x) > \mu_B(y) \end{cases}.$$

Fuzzy reasoning (also known as approximate reasoning) is an inference procedure used to derive conclusions from a set of fuzzy rules and one or more input conditions.

Generalized Modus Ponens Inference Rule

Traditional *modus ponens* inference rule is formulated by the following way:

premise 1 (fact): x is A
premise 2 (rule): IF x is A THEN y is B

---

consequent (conclusion) : y is B

For example, we have the rule $A \to B$, where *A* is “a tomato is red, and *B* is “a tomato is ripe”. If we define the fact that “the tomato is red”, then we can say (can infer) that “the tomato is ripe”.
Consider now fuzzy reasoning. We have the same rule $A \to B$, where *A* is “ a tomato is red”, and *B* is “a tomato is ripe”, but we know the fact that “the tomato is more or less red” (A'). What we can infer as a conclusion? The answer is “the tomato is more or less ripe”. This can be written as:

Premise 1 (fact): x *is* A' ,
premise 2 (rule): IF x is A THEN y is B

---

consequent (conclusion): y *is* B'

This rule of inference is called *generalized modus ponens inference rule*.
Discuss now: how can we calculate the membership value of a conclusion?
Another words, if we know the membership values of $\mu_{A'}(x)$ and $\mu_R(x,y)$, how can we find the membership value of conclusion $\mu_{B'}(y)$?
Consider different types of calculation of conclusion, that is, different types of fuzzy inference.

Fuzzy Reasoning Based on a Max-Min Composition

Compositional Rule of Fuzzy Inference

Consider the following task:

Let A is a fuzzy set on X and R is a fuzzy relation on X $\times$ *Y*: $R = A \to B$. *Find the resulting fuzzy set B on Y.*

Figure 6-3 (a,b,c,d) shows the design of a resulting fuzzy set [13]. To construct resulting fuzzy set *B* we make the following steps:
(1) construct cylindrical extension (denoted as C(*A*)) with base *A*. This means that we expand the domain *A* from *X* to *X* $\times$ *Y*. (Fig.6-3, a));

(Copia modificata per una migliore consultazione on-line)

(2) find the intersection of C(*A*) with *R* (Fig.6-3, c) that is find C(*A*) $\bigcap$ *R* ;

(3) project this intersection C(*A*) $\bigcap$ *R* onto *y*-axis (Fig.6-3, d). This projection is the resulting fuzzy set *B*.

Write now these steps formally. Let $\mu_A(x)$, $\mu_{C(A)}(x,y)$, $\mu_B(y)$, and $\mu_R(x,y)$ are the membership functions of *A*, C(*A*), *B* and *R* , respectively.

$$\mu_{C(A)}(x,y) = \mu_A(x) \text{ by the definition.}$$

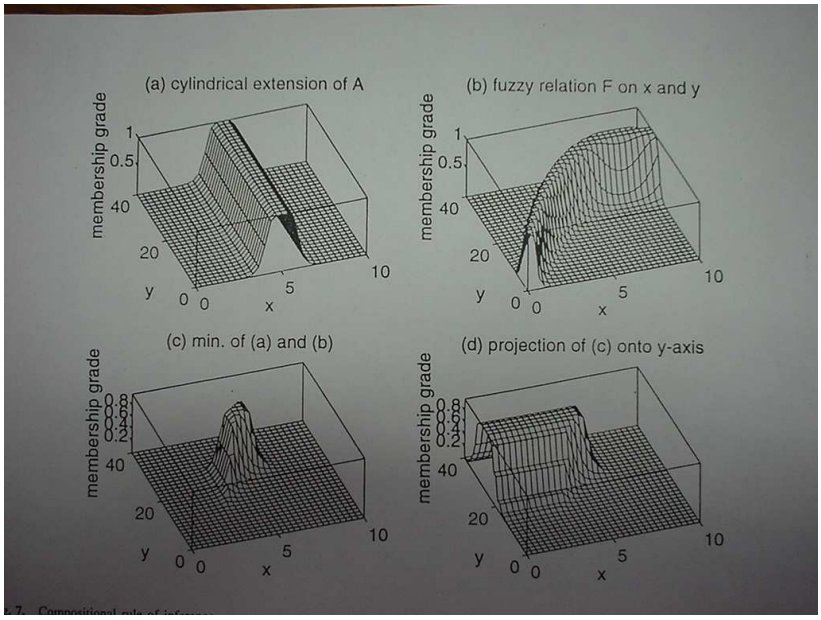


Figure 6-3. Construction of resulting fuzzy set.

Then also by the definition we have

$$\mu_{C(A)\cap R}(x,y) = \min\left[\mu_{C(A)}(x,y), \mu_R(x,y)\right] = \min\left[\mu_A(x), \mu_R(x,y)\right] \qquad (6\text{-}4)$$

By projecting C(*A*) $\bigcap$ *R* onto *y*-axis we have :

$$\mu_B(y) = \max_x \min\left[\mu_A(x), \mu_R(x,y)\right] = \bigvee_x \left[\mu_A(x) \wedge \mu_R(x,y)\right] \qquad (6\text{-}5)$$

Formula (6-5) is called *max-min composition*. Denote this composition by symbol " ∘ ". Then we may write

$$B = A \circ R. \qquad (6\text{-}6)$$

If we choose fuzzy AND operation as the product operation (of membership functions) and fuzzy OR operation as the *max* operation, then we have *max-product composition* and

$$\mu_B(y) = \bigvee_x \left[\mu_A(x) \cdot \mu_R(x,y)\right] \qquad (6\text{-}7)$$

Return now to our fuzzy inference problem:

Premise 1 (fact) : x *is* A' ,

premise 2 (rule): IF x is A THEN y is B

---

consequent (conclusion): y *is* B'

and to the following question: if we know the membership values of $\mu_{A'}(x)$ and $\mu_R(x,y)$, how tofind the membership value of conclusion $\mu_{B'}(y)$?

Now by using the *max-min composition* rule we can write:

$$\mu_{B'}(y) = \max_x \min\left[\mu_{A'}(x), \mu_R(x,y)\right] = \bigvee_x \left[\mu_{A'}(x) \wedge \mu_R(x,y)\right] \qquad (6\text{-}8)$$

or equivalently $B' = A' \circ R$.

Consider now different cases of fuzzy rules.

Fuzzy reasoning with one (single) fuzzy rule with single (one) antecedent

For this case we simplify (6-8) by using formula (6-3):

$$\mu_{B'}(y) = \bigvee_x \left[\mu_{A'}(x) \wedge \mu_R(x,y)\right] = \bigvee_x \left[\mu_{A'}(x) \wedge \{\mu_A(x) \wedge \mu_B(y)\}\right] =$$

$$= \max_x \left[\mu_{A'}(x) \wedge \mu_A(x)\right] \wedge \mu_B(y) = \omega \wedge \mu_B(y). \qquad (6\text{-}9)$$

The mechanism of calculating of formula (6-9) is shown graphically in Fig.6-4.

(Copia modificata per una migliore consultazione on-line)

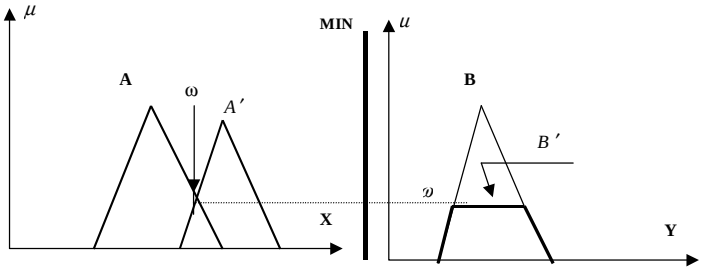


Figure 6-4. Fuzzy reasoning scheme for a single rule with a single antecedent.

We find ω as the maximum of intersection of fuzzy sets *A* and A'. Usually ω is called a *firing strength*. Resulting fuzzy set B' is constructed as cutting the membership function of *B* by ω.

Fuzzy reasoning with one (single) fuzzy rule with two antecedents

A fuzzy rule with two antecedents is usually written as:

"IF x is A AND y is B THEN z *is* C."

So, the inference problem is as follows:

premise 1 (facts) : x *is* A', and y *is* B',

premise 2 (rule): IF x is A AND y is B THEN z *is* C.

---

consequent (conclusion) : z *is* C'

This fuzzy rule represents a ternary fuzzy relation *R* which can be defined by the following membership function:

$$\mu_R(x,y,z) = \mu_{(A\times B)\times C}(x,y,z) = \mu_A(x) \wedge \mu_B(y) \wedge \mu_C(z) \qquad (6\text{-}10)$$

Resulting fuzzy set C' can be represented as:

$$C' = (A' \times B') \circ R \qquad (6\text{-}11)$$

By using (6-10) and extension of (6-5) for the case (6-11), we can calculate $\mu_{C'}(z)$ as:

$$\mu_{C'}(z) = \bigvee_{x,y} \left[(\mu_{A'}(x) \wedge \mu_{B'}(y)) \wedge \mu_R(x,y,z)\right] =$$

$$= \bigvee_{x,y} \left[(\mu_{A'}(x) \wedge \mu_{B'}(y)) \wedge (\mu_A(x) \wedge \mu_B(y) \wedge \mu_C(z))\right] =$$

$$= \bigvee_{x,y} \left[(\mu_{A'}(x) \wedge \mu_{B'}(y) \wedge \mu_A(x) \wedge \mu_B(y))\right] \wedge \mu_C(z) = \qquad (6\text{-}12)$$

$$= \underbrace{\left\{\bigvee_x \left[\mu_{A'}(x) \wedge \mu_A(x)\right]\right\}}_{\omega_1} \wedge \underbrace{\left\{\bigvee_y \left[\mu_{B'}(y) \wedge \mu_B(y)\right]\right\}}_{\omega 2} \wedge \mu_C(z) = \omega_1 \wedge \omega_2 \wedge \mu_C(z)$$

The mechanism of calculating of formula 6-12 is shown graphically in Fig.6-5.

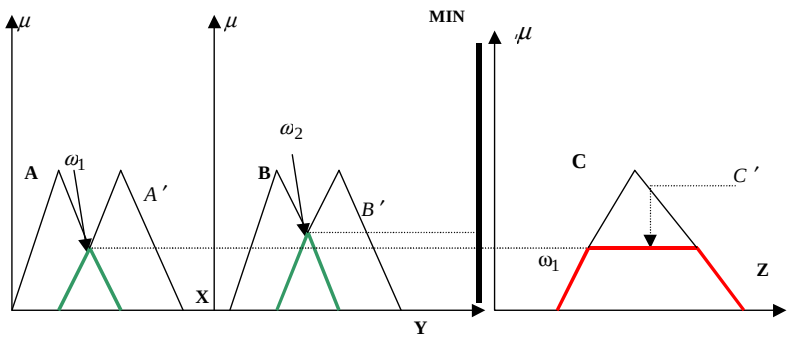


Figure 6-5. Fuzzy reasoning for multiple antecedents.

We find ω_1 and ω_2 (firing strength): ω_1 is the maximum of intersection of fuzzy sets *A* and A', ω_2 is the maximum of intersection of fuzzy sets *B* and B'. Then we give the value of ω as a minimum from the values ω_1 and ω_2.

The resulting fuzzy set C' is constructed by cutting the membership function of *C* by ω.

Remark. Consider another type of fuzzy rule like: "IF x is A OR y is B THEN z *is* C."

(Copia modificata per una migliore consultazione on-line)

In this case the firing strength ω is given as the maximum from ω_1 and ω_2.

Fuzzy reasoning with multiple fuzzy rules with multiple antecedents

Consider the following example of this case.

premise 1 (fact) : $x \ is \ A'$, and $y \ is \ B'$,

premise 2 (rule1): IF $x \ is \ A_1$ AND $y \ is \ B_1$ THEN $z \ is \ C_1$.

premise 3 (rule2): IF $x \ is \ A_2$ AND $y \ is \ B_2$ THEN $z \ is \ C_2$.

---

consequent (conclusion) : $z \ is \ C'$

Here we consider a fuzzy set C' as

$$C' = (A' \times B') \circ (R_1 \cup R_2). \qquad (6\text{-}13)$$

Since the max-min composition operator is distributive over the $\bigcup$ operator we can rewrite the (6-13) formula as:

$$C' = (A' \times B') \circ (R_1 \bigcup R_2)) =$$
$$[(A' \times B') \circ R_1] \bigcup [(A' \times B') \circ R_2] = C_1' \bigcup C_2' \ ,$$

where C_1' and C_2' are the inferred fuzzy sets for rule 1 and rule 2.

So, the final result is constructed as sum (or max) of two inferred fuzzy sets C_1' and C_2'.

In Fig.6-6 the mechanism of calculating of $\mu_{C'}(z)$ is shown graphically. This method of fuzzy inference is called *min-max method*.

Consider now a general case of rules as follows:

Rule l :

$$\text{IF } (x_1 \text{ is } F_1^l) and (x_2 \text{ is } F_2^l) ... and (x_n \text{ is } F_n^l) \text{ THEN } (y \text{ is } G^l), \qquad (6\text{-}14)$$

where $l = 1, ..., M$, M is the number of fuzzy rules.

For the given fuzzy values of input $\tilde{F}(x_i)$ the final conclusion of reasoning is given by

$$\mu(y) = \bigcup_{l=1}^{M} (\omega^l \wedge \mu_{G^l}(y)), \qquad (6\text{-}15)$$

where $$\omega^l = \underbrace{\left\{ \bigvee_{x_1} \left[\mu_{\tilde{F}}(x_1) \wedge \mu_{F_1^l}(x_1) \right] \right\}}_{\omega_1^l} \wedge \cdots \wedge \underbrace{\left\{ \bigvee_{x_n} \left[\mu_{\tilde{F}}(x_1) \wedge \mu_{F_n^l}(x_n) \right] \right\}}_{\omega_n^l}.$$

Remark. Different fuzzy inference techniques are differed by the choice of interpretation of fuzzy AND, fuzzy OR and fuzzy implication operators.

Fuzzy Rules Chaining

We considered simple types of fuzzy rules - *fuzzy rules without chaining*. This means that variables in antecedent parts and variables in consequent parts of rules are different. For example,

IF $x \ is \ A_1$ AND $y \ is \ B_1$ THEN $z \ is \ C_1$

IF $x \ is \ A_2$ AND $y \ is \ B_2$ THEN $z \ is \ C_2$

...

IF $x \ is \ A_n$ AND $y \ is \ B_n$ THEN $z \ is \ C_n$.

In this case variables in the antecedent parts (x and y) and variables in the consequent parts (z) are different.

(Copia modificata per una migliore consultazione on-line)

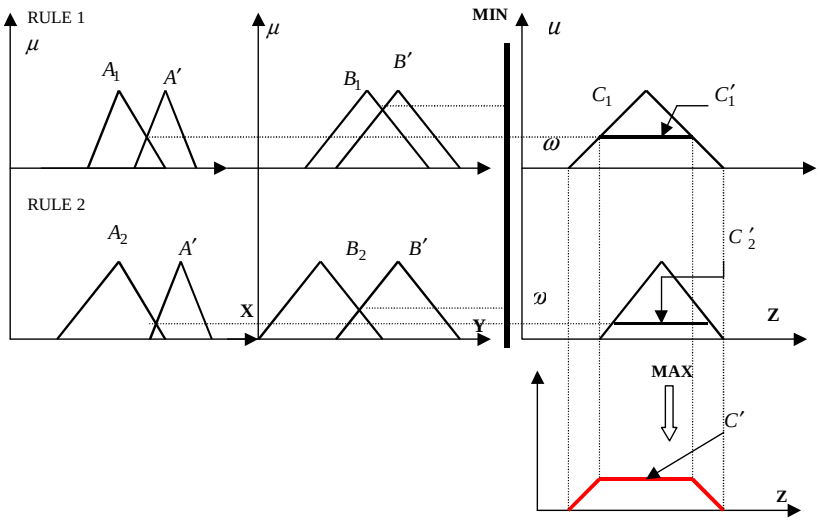


Figure 6-6. Fuzzy Reasoning for multiple rules with multiple antecedents.

Consider now fuzzy rules with chaining. There are two types of *fuzzy rules chaining*: single, and multiple.

In single chaining *of rules, an* intermediate variable *is present in the consequent of* one *rule and in the antecedent of another one. For example,*

$$\text{IF } x \text{ is } A_1 \text{ AND } y \text{ is } B_1 \text{ THEN } p \text{ is } P_1$$
$$\text{IF } p \text{ is } P_2 \text{ AND } y \text{ is } B_2 \text{ THEN } z \text{ is } C_2 .$$

In multiple chaining *of rules, the* intermediate variable *is present in the consequent of* several *rules and in the antecedent of another rule. For example,*

$$\text{IF } x \text{ is } A_1 \text{ AND } y \text{ is } B_1 \text{ THEN } p \text{ is } P_1$$
$$\text{IF } x \text{ is } A_2 \text{ AND } y \text{ is } B_2 \text{ THEN } p \text{ is } P_2$$
$$\text{IF } p \text{ is } P_3 \text{ AND } y \text{ is } B_3 \text{ THEN } z \text{ is } C_1$$
$$\text{IF } p \text{ is } P_4 \text{ AND } y \text{ is } B_4 \text{ THEN } z \text{ is } C_2 .$$

Remark. Formulas for calculation output *z* may be easily written by using Eq. (6-15).

<u>Appendix</u>

Generalized Fuzzy Conjunction , Fuzzy Negation and Fuzzy Disjunction Operations

Consider the generalization of fuzzy conjunction and disjunction operations based on *T- norm* and *T-conorm* concepts [14].
Definition A-1:
T- norm and *T-conorm* are defined as function *T*, *S* : $[0,1]\times[0,1]\to[0,1]$ satisfying the following properties:

(1) $T(x,1)=x,\quad S(x,0)=0$ (boundary conditions);

(2) $T(x,y)\le T(u,v)\ \ and\ \ S(x,y)\le S(u,v)\ \ if\ \ x\le u, y\le v$ (monotonicity);

(3) $T(x,y)=T(y,x),\quad S(x,y)=S(y,x)$ (commutativity);

(4) $T(T(x,y),z)=T(x,T(y,z))\quad S(S(x,y),z)=S(x,S(y,z))$ (associativity).

Definition A-2:
A *negation* is defined as a function *N*: $[0,1]\to[0,1]$ satisfying the following properties:

(1) $N(0)=1,\ N(1)=0$;
$N(x)\le N(y)\ \ if\ \ y\le x$.

Definition A-3:
A negation is called an *involution* if on $[0,1]$ it is fulfilled the involutivity property:

$$N(N(x))=x .$$

Introduce a *parametric class of Sugeno involutive negations* which has the following form:

$$N(x)=\frac{1-x}{1+\lambda x},\lambda>-1 .$$

When $\lambda=0$ we obtain the *negation of Zadeh*: $N(x)=1-x$.

(Copia modificata per una migliore consultazione on-line)

T-norms and *T*-conorms can be obtained one from another as follows:

$$S(x,y) = N(T(N(x),N(y))$$
$$T(x,y) = N(S(N(x),N(y))\text{,}$$

where *N* is an involution.

The simplest examples of T-norms and T-conorms (mutually related by mentioned above relations) for $N(x) = 1 - x$ are shown in Table A-1.

These simplest functions will be used later for the construction of parametric conjuction and disjunction operations.
Generally, for any *T*-norm and *T*-conorm it follows that

$$T_d(x,y) \le T(x,y) \le T_c(x,y) \le S_c(x,y) \le S(x,y) \le S_d(x,y)\,.$$

Table A-1.

$T_c(x,y) = \min\{x,y\}$	$S_c(x,y) = \max\{x,y\}$
$T_p(x,y) = xy$	$S_p(x,y) = x + y - xy$
$T_d = \begin{cases} x, & if \quad y = 1 \\ y, & if \quad x = 1 \\ 0, & otherwise \end{cases}$	$S_d = \begin{cases} x, & if \quad y = 0 \\ y, & if \quad x = 0 \\ 1, & otherwise \end{cases}$
$T_b(x,y) = \max\{0,(x+y-1)\}$	$S_b(x,y) = \min\{1,(x+y)\}$

Hence, *T*-norms T_d and T_c are the minimal and the maximal boundaries for all *T*-norms. Similarly, *T*-conorms S_c and S_d are the minimal and the maximal boundaries for all *T*-conorms.
Remark A-1. These inequalities are very important from a practical point of view because they establish the boundaries of the possible range of operations *T* and *S*.

Consider popular parametric classes of *T*-norms.

- Schweizer-Sklar *T*-norms:

$$T(x,y) = \max\left(0,(x^{-p} + y^{-p} - 1)^{-1/p}\right)$$
$$T(x,y) = 1 - \left[(1-x)^p + (1-y)^p - (1-x)^p(1-y)^p\right]^{1/p};$$

- Yager's *T*-norms:

$$T(x,y) = 1 - \min\left(1,((1-x)^p + (1+y)^p)^{1/p}\right), p > 0;$$

- Dombi's *T*-norms:

$$T(x,y) = \frac{1}{1 + \left[1/(x-1)^p + 1/(y-1)^p - (1-x)^p(1-y)^p\right]^{1/p}}, p > 0.$$

Definition A-4:

A *fuzzy conjunction* operation and a *fuzzy disjunction* operation are functions *T, S* : $[0,1]\times[0,1] \to [0,1]$ satisfying the following properties:

(1) $T(x,1) = T(1,x) = x$;

(2) $S(x,0) = S(0,x) = x$;

(3) $T(x,y) \le T(u,v) \;\; and \;\; S(x,y) \le S(u,v) \;\; if \;\; x \le u, y \le v$

Remark A-2. Naturally, any *T*-norm and *T*-conorm will be a fuzzy conjunction and a fuzzy disjunction with respect to this definition.

Proposition A-1:
Conjuction and disjunction operations satisfy the following properties:

$$T(0,y) = T(x,0) = 0;$$
$$S(1,y) = S(x,1) = 1;$$
$$T_d(x,y) \le T(x,y) \le T_c(x,y) \le S_c(x,y) \le S(x,y) \le S_d(x,y);$$
$$T(0,0) = T(0,1) = T(1,0) = 0\,,\quad T(1,1) = 1;$$
$$S(0,0) = 0,\quad S(0,1) = S(1,0) = S(1,1) = 1\,.$$

<u>Lecture N 7 From Fuzzy Logic to Fuzzy Systems</u>

A Fuzzy System (or a *Fuzzy Model*) is a popular computing framework based on concepts of fuzzy set theory and fuzzy reasoning.
The basic structure of the fuzzy system is shown in Fig.7-1.

(Copia modificata per una migliore consultazione on-line)

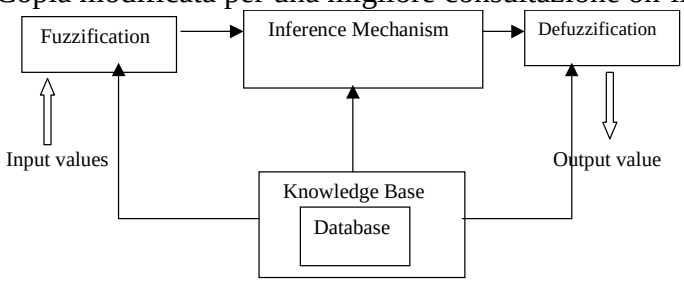


Figure 7-1. The basic structure of a fuzzy system.

Fuzzy system consists of the following components:

- Knowledge Base including Fuzzy Rules Base and Database,
- Reasoning (or Inference) Mechanism, and
- Fuzzification/Defuzzification modules.

Fuzzy Rules Base contains a set of fuzzy rules "if-then". The Database defines membership functions used in the fuzzy rules.
The fuzzification component converts input variable's values (crisp values) to fuzzy set values. The defuzzification component converts fuzzy set values to output crisp value.

Fuzzification

Fuzzification is *the process of finding the membership degree* $\mu_A(a)$ *for an input crisp value* a. The fuzzifier performs a mapping from a crisp point $a \in U$ (where *U* is a universal set) into a fuzzy set *A* in *U*.
There are two possible choices of this mapping:

- *singleton fuzzifier*: in this case fuzzy set *A* is defined as: $\mu_A(x) = 1$ for $x = a$ and $\mu_A(x) = 0$ for all other $x \neq a$;
- *nonsingleton fuzzifier*: in this case $\mu_A(a) = 1$ and $\mu_A(x)$ decreases from 1 as x moves away from a. For example, $\mu_A(x) = \exp\left[-(x-a)^2/\sigma^2\right]$, where σ^2 is a parameter characterizing the shape of $\mu_A(x)$.

In many applications the singleton fuzzifier has been used. A nonsingleton fuzzifier may be useful if the inputs are corrupted by noise.
Remark 1. If we have a partition of a chosen universe of discourse on some parts described by appropriate membership functions, then the fuzzification process may be considered as shown in Fig.7-2.

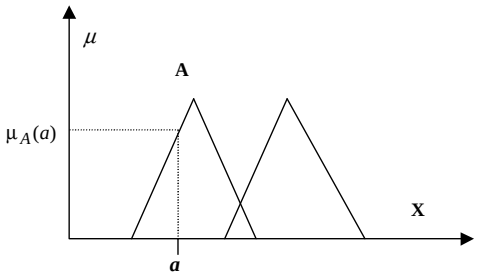


Figure 7-2. Fuzzification example.

Defuzzification

When the input data are crisp and the output value(s) are expected to be crisp too, defuzzification module is used. The purpose of defuzzification process is to obtain a crisp value from inferred membership function $\mu_C(z)$, $z \in Z$.

So, a defuzzifier *performs a mapping from fuzzy set in* Z *to a crisp value* z. There are a few possible choices of that:

- *maximum defuzzifier* defined as
  $z = \arg\sup \mu_C(z)$;
- *center of gravity defuzzifier* defined as

$$z = \frac{\int_Z \mu_C(z) z dz}{\int_Z \mu_C(z) dz} \text{ for continuous case; } z = \frac{\sum_Z \mu_C(z) z}{\sum_Z \mu_C(z)} \text{ for discrete case.}$$

Remark 2. The main idea of a center of gravity defuzzifier is described in Appendix.
Different defuzzification strategies are shown in Fig.7-3.

Example of defuzzification. Consider the following universal set $Z = \{1,2,...8\}$ and let an inferred set *C* be as follows:

$$C = \{0.5/3, 0.8/4, 1/5, 0.5/6, 0.2/7\}.$$

Then the output $\bar{z}$ based on the center of gravity defuzzifier is computed as:

(Copia modificata per una migliore consultazione on-line)

$$\bar{z} = \frac{0.5\times3+0.8\times4+1\times5+0.5\times6,+0.2\times7}{0.5+0.8+1+0.5+0.2} = 4.7$$

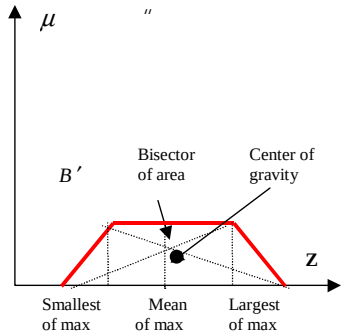


Figure 7-3. Different strategies of defuzzification.

Typical Fuzzy Models

Consider three types of the most commonly used fuzzy models. They are using different fuzzy inference techniques and defuzzification methods.

1. Mamdani Fuzzy Model

The typical fuzzy rule in Mamdani model is following:

Rule i : "IF x *is* A_i AND y *is* B_i THEN z *is* C_i" ($i = 1,2,...,m$).

Mamdani fuzzy inference technique is based on using Mamdani fuzzy implication, *min* and *max* for fuzzy AND and fuzzy OR operations (see Lecture 6). In this case for given fuzzy values of input A' and B' Mamdani Fuzzy Model infers a fuzzy value $\mu_{C'}(z)$ of the output as follows:

$$\mu_{C'}(z) = \bigvee_{i=1}^{m}\left[\underbrace{\left\{\bigvee_x\left[\mu_{A'}(x)\wedge\mu_{A_i}(x)\right]\right\}}_{\omega_1}\wedge\underbrace{\left\{\bigvee_y\left[\mu_{B'}(y)\wedge\mu_{B_i}(y)\right]\right\}}_{\omega 2}\wedge\mu_{C_i}(z)\right] = \bigcup_{i=1}^{m}(\omega_1^i\wedge\omega_2^i\wedge\mu_{C_i}(z))$$

For crisp values of the input x^*, y^* the above expression for the ω_i is simplified: $\omega_1^i = \mu_{A_i}(x^*), \omega_2^i = \mu_{B_i}(y^*)$

and we get the following: $\mu_{C'}(z) = \bigcup_{i=1}^{m}(\omega_1^i\wedge\omega_2^i\wedge\mu_{C_i}(z)$.

In Fig.7-4 a graphical representation of Mamdani fuzzy inference technique based on using *min* and *max* for fuzzy AND and fuzzy OR operations is shown.

In Fig. 7-5 a graphical representation of Mamdani fuzzy model based on using *product* and *max* for fuzzy AND and fuzzy OR is shown. In this case we calculate resulting fuzzy set as follows:

$$\mu_{C'}(z) = \bigcup_{i=1}^{m}(\omega_1^i\times\omega_2^i\times\mu_{C_i}(z)$$

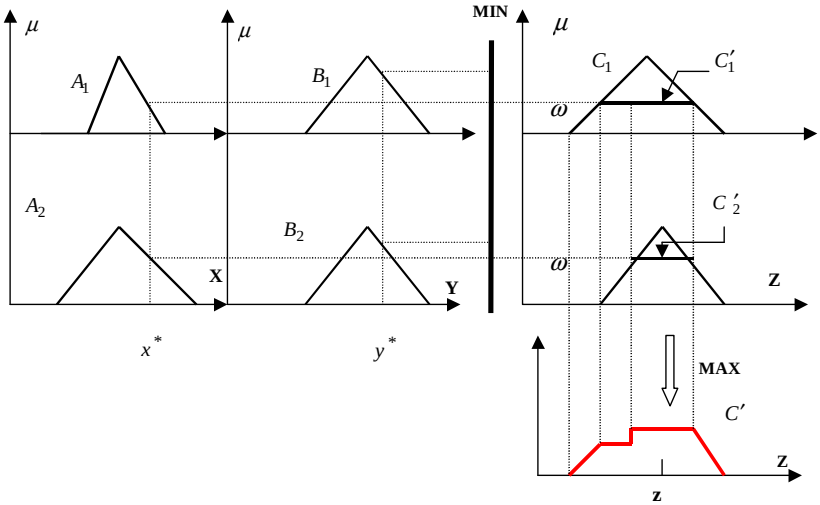


Figure 7-4. Max-min Mamdani fuzzy inference technique.

Here you can see two types of inference: min-max and min-product. Crisp values x^* and y^* are the input, the output is crisp value z .

2. Sugeno Fuzzy Model

This model is also called as Takagi-Sugeno fuzzy model. The typical fuzzy rule in Sugeno model is following:

"if x *is* A and y *is* B then $z = f(x,y)$,

where $z = f(x,y)$ is a crisp function.

(Copia modificata per una migliore consultazione on-line)

Usually $f(x,y)$ is a polynomial function.When $f(x,y)$ is a first order polynomial, then fuzzy model is called *first-order Sugeno fuzzy model*. When $f(x,y)$ is a constant, then fuzzy model is called *zero-order Sugeno fuzzy model*.

When $f(x,y)$ is a first order polynomial, then fuzzy model is called *first-order Sugeno fuzzy model*. When $f(x,y)$ is a constant, then fuzzy model is called *zero-order Sugeno fuzzy model*.

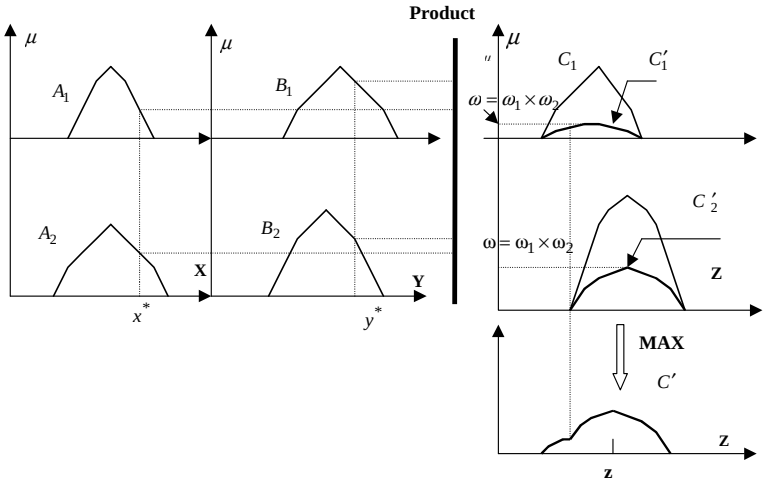


Figure 7-5. Max-product Mamdani fuzzy inference technique.

In Fig.7-6 a graphical representation of Sugeno fuzzy inference technique is shown.

3. Tsukamoto Fuzzy Model

In Tsukamoto fuzzy model the consequent of each fuzzy rule is represented by a fuzzy set with a monotonical membership function as shown in Fig.7-7. Output crisp value is defined as weighted average.

All mentioned above models of fuzzy inference are widely used in Fuzzy Controllers (We will consider these examples in the next lecture).

Example of fuzzy logic application: Fuzzy logic-based driving a car.

Consider the situation of driving a car [11]. Human expert uses some decision making before to make operations of gas pedaling and braking. For example, when a car is climbing a hill, we know that the car will lose a speed, if no action is taken. Therefore, we step on the pedal for *more gas* when the hill is steep, and may be only a *little* when it is not so steep.

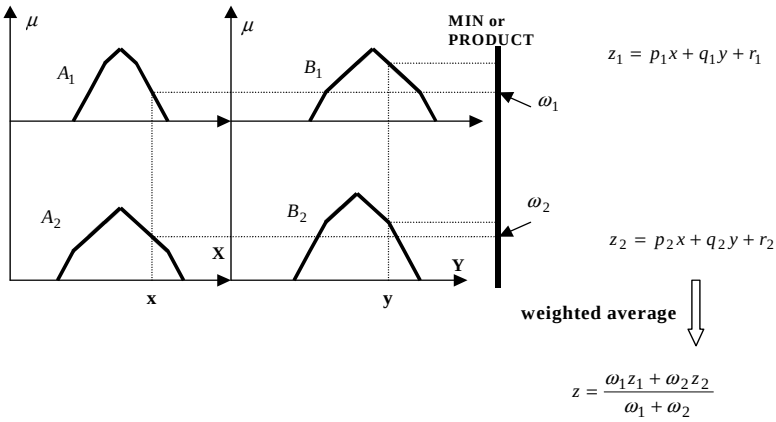


Figure 7-6. A graphical representation of Sugeno fuzzy inference.

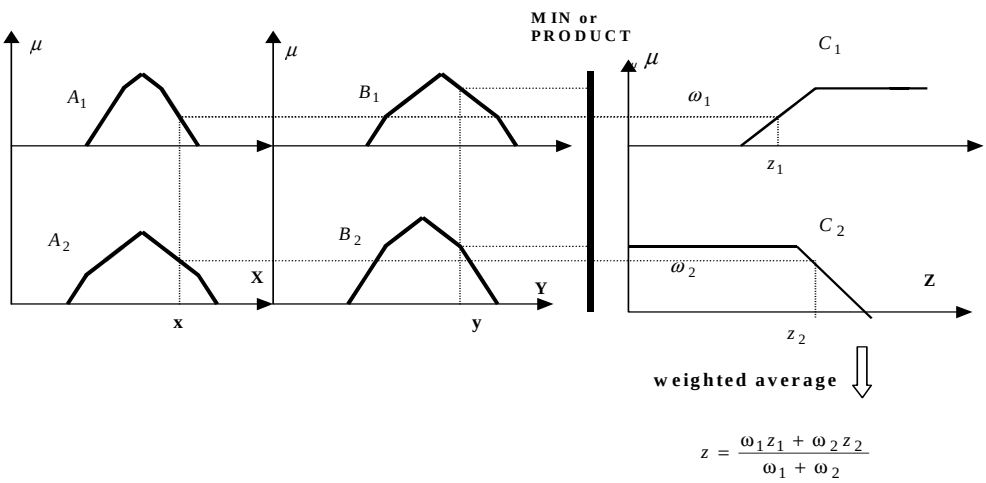


Figure 7-7. A graphical representation of Tsukamoto fuzzy inference.

Consider another driving situation when a car runs on a road and a driver looks another car on the road in front of him. If a distance between cars is *long* and a

speed is *slow*, then the driver holds the gas pedal steady (maintain the speed).

This kind of *human control strategy* is difficult and unnatural to express in mathematical equations. In this case we will use "if-then" rules. Once we establish the fuzzy rules, we can realize the control strategy by fuzzy reasoning. For the situation of driving a car on a road, assume the following rules for this fuzzy reasoning:

Rule 1: IF *distance* between cars is *short* AND *speed* is *slow*, THEN hold the gas pedal steady (*maintain the speed*).
Rule 2: IF *distance* between cars is *short* AND *speed* is *fast*, THEN step on the brake (*reduce the speed*).
Rule 3: IF *distance* between cars is *long* AND *speed* is *slow*, THEN step on the gas pedal (*increase the speed*).
Rule 4: IF *distance* between cars is *long* AND *speed* is *fast*, THEN hold the gas pedal steady (*maintain the speed*).

The preceding rules are written in ordinary words, and we cannot directly apply fuzzy logic. At first, we introduce the following linguistic variables: "distance", "speed", and "acceleration". Then we must describe their values "short", "long", "slow", step on brake" and so on by corresponding fuzzy sets. In our example, these fuzzy sets can be represented by membership functions shown in Fig.7-8.

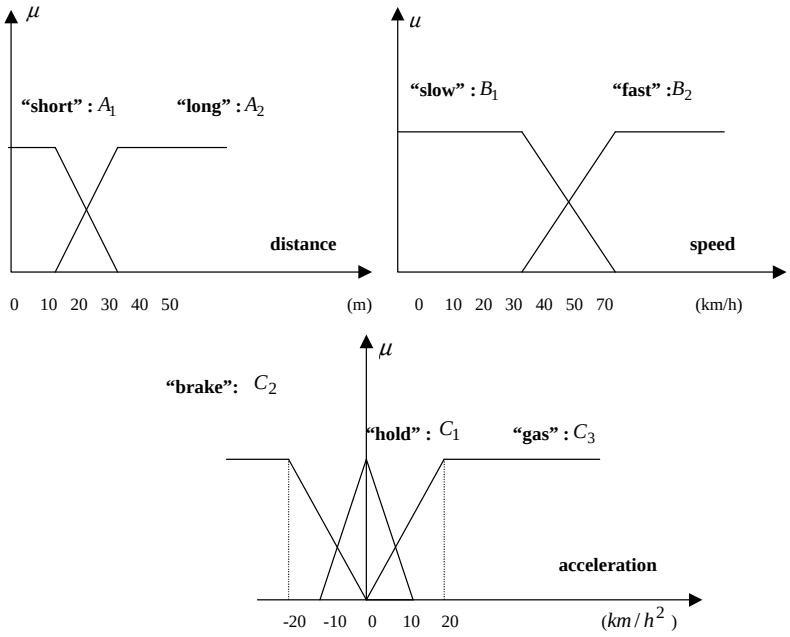


Figure 7-8. Membership functions for example 1.

These membership functions need to be defined appropriately for the situations under consideration. For example, the speed of 70 km/h would be "fast" on a street road but it would be "slow" on a highway.

Let us introduce the following linguistic variables: x is a distance between a car, y is a speed of a car, and z is an acceleration of a car (it allows to adjust the car's speed by gas pedaling or braking operations).

The respective universal sets can be defined as

$X = \{x \mid 0 \le x \le 40\}$ [m]	$Y = \{y \mid 0 \le y \le 100\}$ [km/h]	$Z = \{z \mid -20 \le x \le 20\}$ $[km/h^2]$

Remark 3. The preceding regions can be determined by common sense. For example, the distance between cars and the speed cannot take negative values. There are certain upper and law bounds on the distance between cars and there is a speed limit on a road.

Let us now label each fuzzy set as:

A_1 :"short ; (distance-between- cars) A_2 : "long"; (distance between cars)	B_1 : "slow" (speed); B_2 : "fast" (speed);	C_1 : "maintain" (speed); C_2 : "reduce" (speed); C_3 : "increase" (speed).

Then the preceding rules written in ordinary words can be rewritten in the IF-THEN form such as

Rule 1: IF x *is* A_1 and y *is* B_1 THEN z *is* C_1

Rule 2: IF x *is* A_1 and y *is* B_2 THEN z *is* C_2

Rule 1: IF x *is* A_2 and y *is* B_1 THEN z *is* C_3

Rule 1: IF x *is* A_2 and y *is* B_2 THEN z *is* C_1.

We can also express these rules as the following look-up (or fuzzy rule) table:

		y	
		B_1	B_2
x	A_1	C_1	C_2
	A_2	C_3	C_1

Remark 4. When the number of rules increases, it can become more and more difficult to see individual rules in a sequential order. In such a case, it is better to summarize the rules into a rule.

(Copia modificata per una migliore consultazione on-line)

In Fig.7-9 the reasoning process is shown. Here input crisp values for the distance between cars and the speed are 18 m and 48m/h, respectively. We use Mamdani fuzzy inference model. The final fuzzy set of conclusion may be called as “reduce speed a little”. The output crisp value is calculated as the center of gravity point and is shown by sign “$\downarrow$” on the picture.

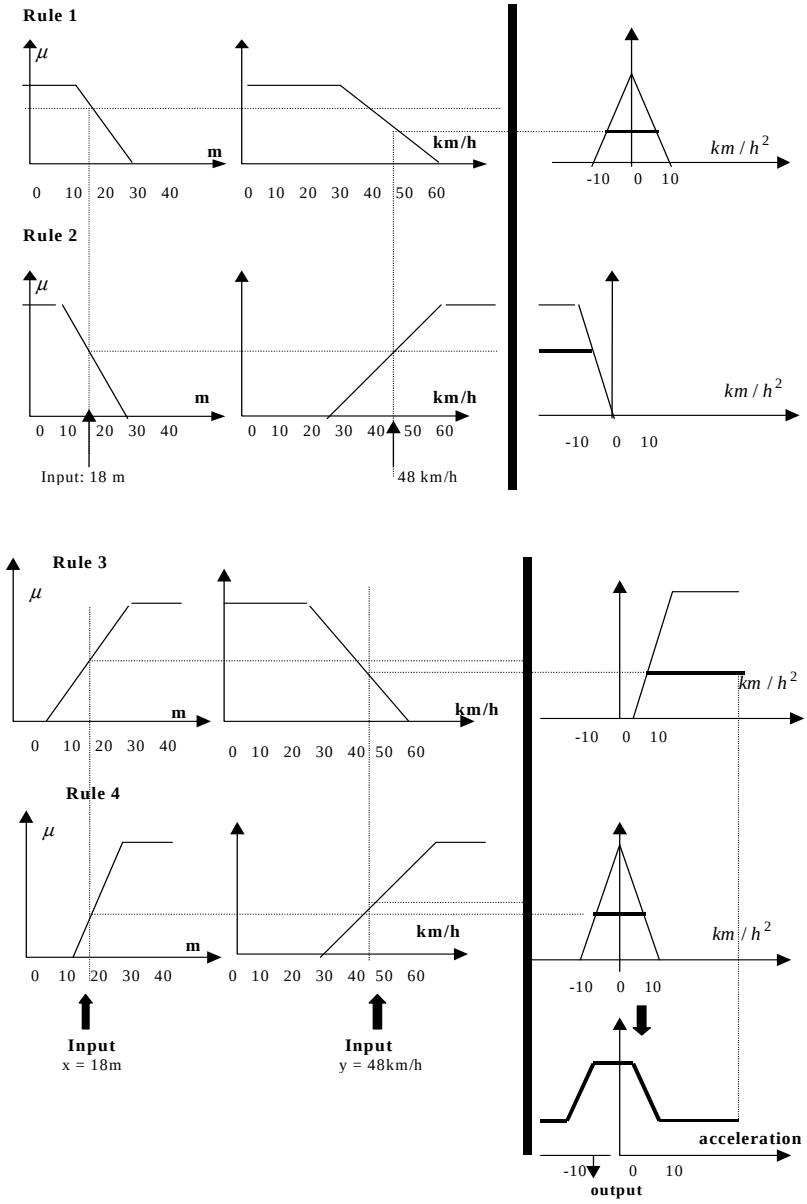


Figure 7-9. Reasoning process for a car driving control.

Finally, let us discuss main properties of fuzzy models.

Properties of Fuzzy Models

Fuzzy Systems as Universal Approximators

Fuzzy systems have very attractive property for control systems design, namely, a fuzzy system can be considered as an universal approximator of systems with unknown dynamic and structure. *What is an approximator*?

A set of fuzzy rules “ if x_i , then y_i “ can be considered as a mapping function $y = f(x)$ such that it approximates to a certain level of accuracy all the data (x_i , y_i). The function f is used to calculate the output value y' for a new input x' possibly not in the data set.

Let us discuss the following question:

Can fuzzy systems be used to achieve arbitrary complex mapping, that is, can fuzzy systems be an universal approximator?

There are two so-called *existence theorems* that proof fuzzy system can be an universal approximator [15].

Theorem 1 (Kosko, 1992 [15]):

An additive fuzzy system uniformly approximates a function $f : x \to y$, if the domain of x is compact (closed and bounded) and f is continuous.

Theorem 2 (Wang, 1994, [16]):

For any given real continuous function f on a compact set $U \subseteq R^n$ and arbitrary ε , there exists a fuzzy logic system F with product fuzzy implication , singleton fuzzifier and COG (center-of-gravity) deffuzifier, and Gaussian membership functions such that

$$|F(x) - f(x)| < \varepsilon .$$

These theorems justify the wide application of fuzzy logic systems to nonlinear control.

Useful Lemmas

(Copia modificata per una migliore consultazione on-line)

Consider a fuzzy system consisting of the following fuzzy rules:

Rule l :

$$\text{IF } (x_1 \text{ is } F_1^l) and (x_2 \text{ is } F_2^l) ... and (x_n \text{ is } F_n^l) \text{ THEN } (y \text{ is } G^l), \quad (7\text{-}1)$$

where $l = 1,...,M$, and *M* is the number of rules of the fuzzy system.

Lemma 1 (Wang, 1994 [16]).

The output of a fuzzy logic system with a *fuzzy rules base* represented as shown in (7-1), with the *singleton* fuzzifier and the *center average* defuzzifier, and with Mamdani max-product inference engine can be calculated by the following:

$$\underbrace{y}_{output-of-fuzzy-system} = f(\bar{x}) = \frac{\sum_{l=1}^{M} \tilde{y}^l (\prod_{i=1}^{n} \mu_{F_i^l}(x_i))}{\sum_{l=1}^{M} (\prod_{i=1}^{n} \mu_{F_i^l}(x_i))},$$

where $\bar{x} = (x_1, x_2, ..., x_n)$ is a crisp input vector to a fuzzy system, $\tilde{y}^l$ is the value at which membership function μ_{G^l} achieves its maximum, i.e. $\mu_{G^l}(\tilde{y}^l) = 1$.

Lemma 2 (Wang, 1994).

The output of a fuzzy logic system with a *fuzzy rules base* represented as shown in Eq.(7-1), with *singleton* fuzzifier, *center average* defuzzifier, with Mamdani max-product inference engine and Gaussian membership functions can be calculated as follows:

$$f(\bar{x}) = \frac{\sum_{l=1}^{M} \tilde{y}^l \left[\prod_{i=1}^{n} a_i^l \exp\left(-\left(\frac{x_i - \bar{x}_i^l}{\sigma_i^l} \right)^2 \right) \right]}{\sum_{l=1}^{N} \left[\prod_{i=1}^{n} a_i^l \exp\left(-\left(\frac{x_i - \bar{x}_i^l}{\sigma_i^l} \right)^2 \right) \right]},$$

where $\bar{x} = (x_1, x_2, ..., x_n)$ is a crisp input vector to a fuzzy system, $\tilde{y}^l$ is the value at which membership function μ_{G^l} achieves its maximum, i.e. $\mu_{G^l}(\tilde{y}^l) = 1$; and membership functions are given as: $\mu_{F_i^l}(x_i) = a_i^l \cdot e^{-\frac{(x_i - \bar{x}_i^l)^2}{\sigma_i^{l\,2}}}$.

Finally, let us define an adaptive fuzzy system *as a fuzzy logic system equipped with a training algorithm, where the training algorithm adjusts parameters of the fuzzy logic system.*

Different training algorithms we will consider in following lectures.

Appendix

Main idea of defuzzification

As a result of applying fuzzy reasoning technique, we get a fuzzy set $\mu_C(z)$ that describes for example for each possible control value z , how reasonable it is to use this particular value. In other words, for every possible control value *z*, we get truth value $\mu_C(z)$ that describes to what extent this value *z* is reasonable to use. In automatic control applications, we want to transform this fuzzy information into a single value $\bar{z}$ of the control that will actually be applied.

So, we want to select a crisp value of control $\bar{z}$ that, on average, would lead to the smallest error. If we choose $\bar{z}$, and the best control is *z*, then the control error is $\bar{z} - z$. Thus, to determine $\bar{z}$, we can use the least squares method. As weights for each square $(\bar{z} - z)^2$, we can take the grade of truth $\mu_C(z)$ with which *z* is a reasonable control value. As a result, we get the following formula for determining $\bar{z}$:

$\int \mu_C(z)(\bar{z} - z)^2 dz \to \min$. Differentiating the minimized function with respect to the unknown $\bar{z}$ and equating the derivative to 0, we get the formula

$$\bar{z} = \frac{\int z \cdot \mu_C(z) dz}{\int \mu_C(z) dz}.$$

This formula is called *centroid defuzzification*.

Remark 5. In many real situations of control the knowledge base consists of compatible “if-then” rules. In such situations, the above defuzzification methodology leads to a good control. But, there exist real situations in which some rules are incompatible. For example, let us consider a following case. Suppose we have an automatic controller for a car. If the car is moving on an empty road, and there is an obstacle straight ahead of the car (e.g., a box that fell from a truck), then a reasonable decision is to swerve to avoid this obstacle. Since the road is empty, there are two possibilities:

- swerve to the right; and

- swerve to the left.

For swerving, the control variable *z* is the angle to which we steer the wheel. Based on the distance to the obstacle and on the speed of the car, an experienced driver can describe a reasonable amount of steering z_0 . In reality, z_0 will probably be a fuzzy value, but, for simplicity, we can assume that z_0 is precisely known.

As result of formalizing expert knowledge, we conclude that for the case of empty road, there are two possible control values: z_0 and $-z_0$, both with grade of truth 1. For this

$\mu(z)$, centroid defuzzification formula leads to $\bar{z}$ = 0, i.e., the car will run directly into the box.

Once we have detected such a situation, how can we repair it? There are two possible ways of doing it:

- we should carefully analyze the rules, find out which of them are inconsistent, and modify them so as to avoid inconsistency;
- if it is impossible to analyze the rules, then we must use heuristic approaches, e.g., we can use special, more complicated, defuzzification methods.

Lecture N 8 From Fuzzy Systems to Fuzzy Control

Before considering of fuzzy control systems let us refresh main ideas of control problem at all, and discuss some questions related to generally control systems, and more particularly to control systems (controllers) using artificial intelligence to control a nonlinear plant. The general scheme of a *feedback* (or *closed loop*) *control system* is shown in Fig. 8-1:

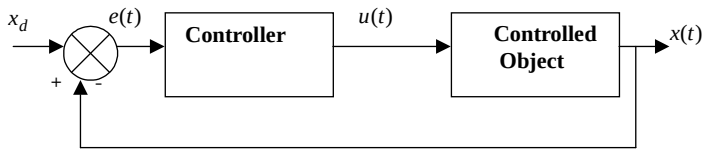


Figure 8-1. Generic structure of a closed loop control system

A *central component* in the feedback control system is a *controlled object*, otherwise known as a process "*plant*", whose output variables are to be controlled.

The task of a *feedback control system* is to maintain the output of a plant at a desired value x_d in spite of *external disturbance forces* that would make the output away from desired value.

In Fig.8-1, $x(t)$ is a state vector describing behavior of a controlled object (a plant), x_d is desired state of the controlled object, called also as a *reference signal*; $e(t)$ is an *error* calculated as $e(t) = x_d - x(t)$, and $u(t)$ is a control vector.

Consider simple example of a feedback control system: a *household furnace* controlled by a *thermostat*.

The thermostat continuously measures the air temperature of the house, and when the temperature falls below a desired minimum temperature the thermostat turns the furnace on. When the furnace has warmed the air above the desired minimum temperature, then the thermostat turns the furnace off. The thermostat-furnace system maintains the household temperature at a constant value in spite of external disturbances such as a drop in the outside air temperature. Similar types of feedback control are used in many applications.

In this example, the *controlled object* is the house, the output variable is the air temperature of the house, and the disturbance is the flow of heat through the walls of the house. The *control system* is the thermostat in combination with the furnace.

The thermostat-furnace system uses *simple on-off feedback control system* to maintain the temperature of the house.

In many control environments, for example motor speed control systems, simple on-off feedback control is *insufficient*.

More advanced classical control systems rely on combinations of proportional feedback control, integral feedback control, and derivative feedback control.

Feedback control that is the sum of *proportional* plus *integral* plus *derivative* feedback is often called as PID control (see Fig.8-2).

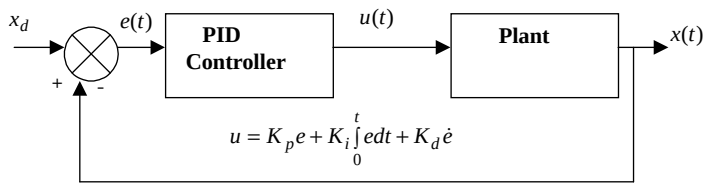


Figure 8-2 . Block diagram of a classical PID control system

In Fig.8-2, control signal $u(t)$ is calculated as

$$u = K_p e + K_i \int_o^t e dt + K_d \dot{e} ,$$

where e is the error signal, $\dot{e}$ is the derivative of error, and K_p, K_i, K_d denote proportional gain, the integral gain and the derivative gain, respectively.

The *PID control system is a linear control system that is based on a linear dynamic model of the plant.* In classical control systems, a linear dynamic model is obtained in the form of dynamic equations, usually ordinary differential equations as follows:

$$\dot{x} = Ax + Bu ,$$

where *x* is a state vector of a plant, and $u = K \cdot x$ is the control vector. Matrix *K* is called a gain matrix, matrices *A*, *B* are constant.

A typical behavior of a closed loop control system is shown in Fig.8-3.

(Copia modificata per una migliore consultazione on-line)

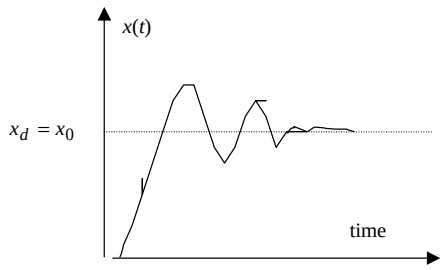


Figure 8-3. Typical response of a closed loop control system.

The plant is assumed to be relatively linear, time invariant, and stable. However, many real-world plants are time varying, highly nonlinear, and unstable. For example, the dynamic model may contain parameters (e.g., masses, inductances, aerodynamics coefficients, etc.) which are either poorly known or depend on a changing environment.

If the parameter variation is small and the dynamic model is stable, then the PID controller may be sufficient. However, if the parameter variation is large, or if the dynamic model is unstable, then this kind of control system is incapable to perform the control task.

Evaluating the motion characteristics of a nonlinear plant is often difficult, in part due to the lack of a general analysis method.

Conventionally, when controlling a plant with nonlinear motion characteristics, it is common to find certain equilibrium points of the plant and the motion characteristics of the plant are linearized in a vicinity of an equilibrium point. Control is then based on evaluating the pseudo (linearized) motion characteristics near the equilibrium point. This technique works poorly, if at all, for plants described by models that are unstable or dissipative.

Classical control methods are based on mathematical formulas and solving a set of differential equations. But there are some difficulties in using classical control methods.

First difficulty:

Solving a set of differential equations might be too slow for a fast real-type control process.

Second difficulty:

A slight change in the parameters of the object (for example, the resistance of air) requires a new set of differential equations to be defined.

In order to avoid all mentioned above and other problems new intelligent (AI-based) control systems are needed.

What is an intelligent control?

Intelligent control is the realization of human control strategies and adaptation/ optimization skills in a control system.

Human control strategies are implemented in Fuzzy Control Systems (called also as *Fuzzy Controllers*) by using a fuzzy logic approach and human adaptation/ optimization skills are implemented in *Optimizers* by using genetic algorithms and neural networks-based training.

So, new types of *self-organizing AI control systems* adapted to control a nonlinear plant use *a fuzzy controller together with an optimizer*.

Consider examples of intelligent control based on a fuzzy logic approach.

Fuzzy Controllers Design Problems

Fuzzy systems are used widely in control application for Fuzzy Controllers (FC) design. The structure of a fuzzy controller is shown in Fig.8-4.

Design of Fuzzy Controllers consists of four main steps:

1. construction of control (fuzzy) rules ;
2. selection of inference technique;
3. parameters tuning to determine fuzzy sets;
4. validation and revision of control rules.

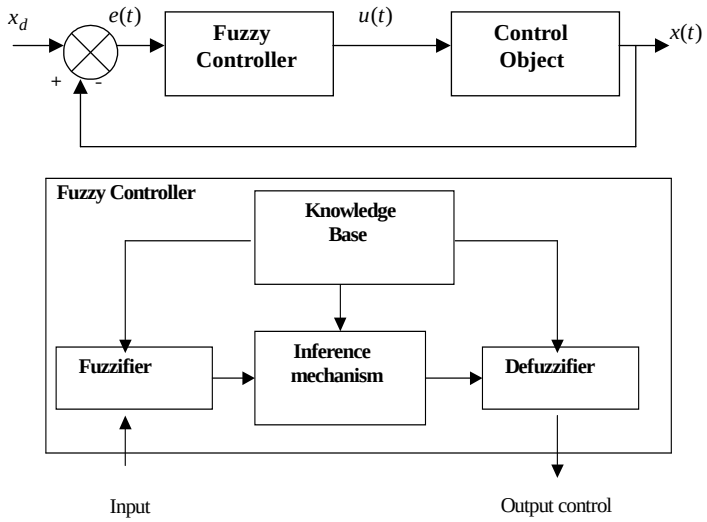


Figure 8-4. The Fuzzy Controller structure.

Once we establish fuzzy rules, we can realize the control strategy by fuzzy reasoning. Therefore, the structure of the Fuzzy Controller is the structure of fuzzy reasoning itself (see Fig.8-4).

There are a few important questions in the design of FC.

1. How do we choose an appropriate inference technique?
2. How is a fuzzifier chosen?
3. How is a defuzzifier chosen?

(Copia modificata per una migliore consultazione on-line)

4. How do we determine functional forms for the membership functions?
5. How to tune parameters of fuzzy sets?

In general, we may use the following criteria for these questions.
Empirical Fit: fuzzy logic systems are used to incorporate linguistic information, it is important that the choices generate appropriate models of real-system behavior.
Computational Efficiency: for large problems or limited computing power, we may have to select simpler fuzzy logic systems.
Easy for adaptation: because we will develop training algorithms for the fuzzy logic system, the selection must result in systems, which are easy to adapt.

Consider examples of fuzzy controller design for practical applications.
Example 1: Elevator Fuzzy Controller.

Elevator controller must perform a control of a lift motion. For this task we will not use classical mechanical modeling based on an equation of motion, we will use fuzzy modeling. Let us discuss design problems of the elevator fuzzy controller.

The first step in FC designing is defining of a set of fuzzy rules (usually given by a human expert). For example, let the following rules are given by some human expert:

1) IF the elevator is moving upward with a slow speed and its position is far below the desired position, THEN make a motor move it rapidly upward.
2) IF the elevator is moving upward with a medium speed and it is a little below the desired position, THEN make a motor move it slowly upward

These rules can be expressed in conventional form by introducing the following linguistic variables: *position* and *velocity*. For our example we consider the following values of linguistic variables:
SP – small positive; MP – medium positive; LP – large positive; SN – small negative; MN – medium negative; LN – large negative.
Rewrite now the above rules as follows:

1) IF *velocity is SP (small positive)*, AND *position is LN (large negative)*, THEN *motor voltage is LP (large positive)*.
2) IF *velocity is MP (medium positive)*, AND *position is SN (small negative)*, THEN *motor voltage is SP (small positive)*.

The task of FC is to process these two rules (and possibly others) when specific values of position and velocity have been determined (by sensors).
In Fig.8-5 a fuzzy reasoning process in the elevator control system based on the *min-max* method of inference is shown.
You can see how Fuzzy Controller would process two rules when specific values of position and velocity have been determined by sensors.
Discuss briefly this reasoning process. Following input values are determined by the sensors:

- current velocity input is equal 0.1 *ft/sec*, and
- current position input is equal -10 *ft* from the desired level.

The velocity input of 0.1 *ft/sec* is considered by SP membership function of *Rule*1. Parameter ω_1 is found equal to 0.6.

The position input of -10 *ft* is considered by LN membership function of *Rule* 1, and parameter ω_2 is found equal to 0.8.

The output fuzzy set's membership function is constructed by cutting the membership function of consequence by the $\omega = min\{0.6, 0.8\} = 0.6$. (set C_1')
By the same way the membership function of consequence of *Rule* 2 is constructed. The velocity input of 0.1 *ft/sec* is considered by MP membership function of *Rule* 2. Parameter ω_1 is found equal to 0.3.

The position input of -10 *ft* is considered by SN membership function of *Rule* 2, and parameter ω_2 is found equal to 0.2.

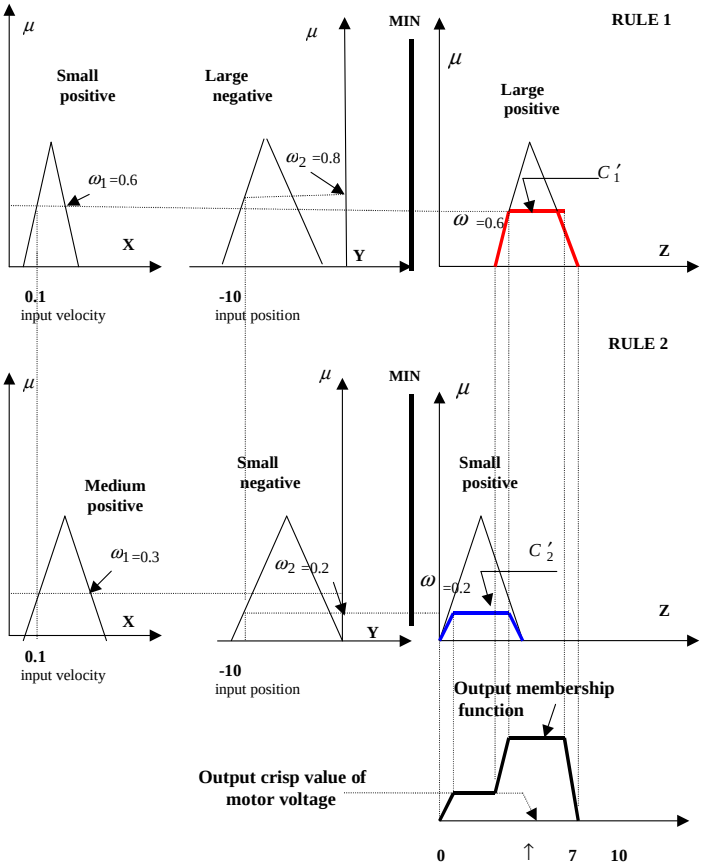


Figure 8-5. Min-Max Fuzzy reasoning for Elevator Control.

(Copia modificata per una migliore consultazione on-line)

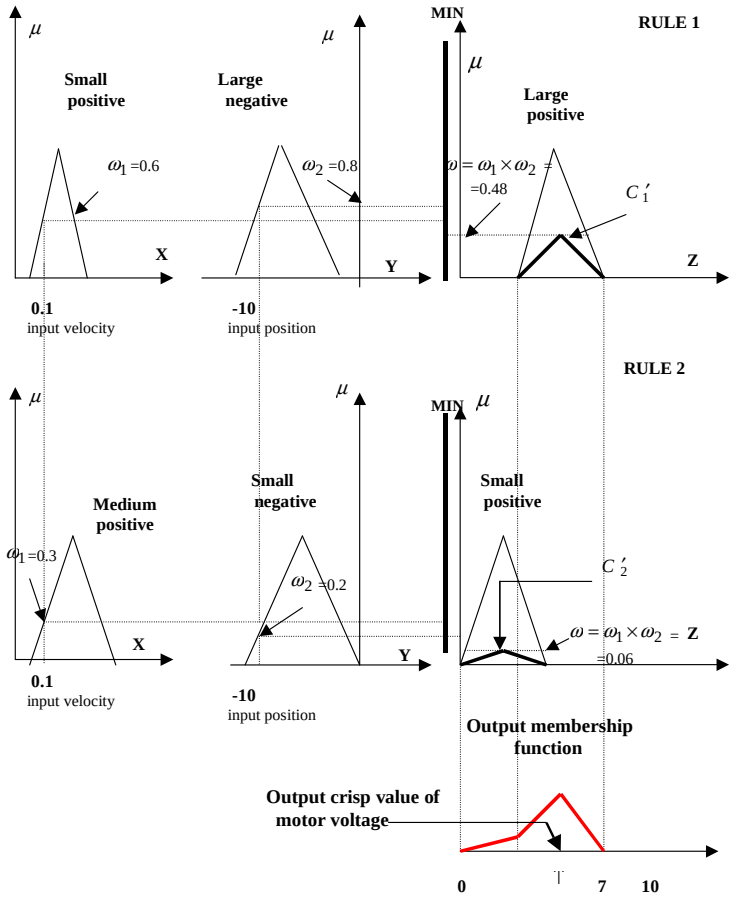


Figure 8-6. Product-Max Fuzzy reasoning for Elevator Control

The output fuzzy set's membership function is constructed by cutting the membership function of consequence by the ω = *min*{ 0.3,0.2 }= 0.2 (the set C_2').

The next step of inference is to combine the output values of the 2 rules as sum (or max) of two inferred fuzzy sets C_1' and C_2'.

The final step is finding a crisp value of the output. The crisp value of an output motor voltage is defined as x -component of a *centroid point* (or *center of gravity point*).

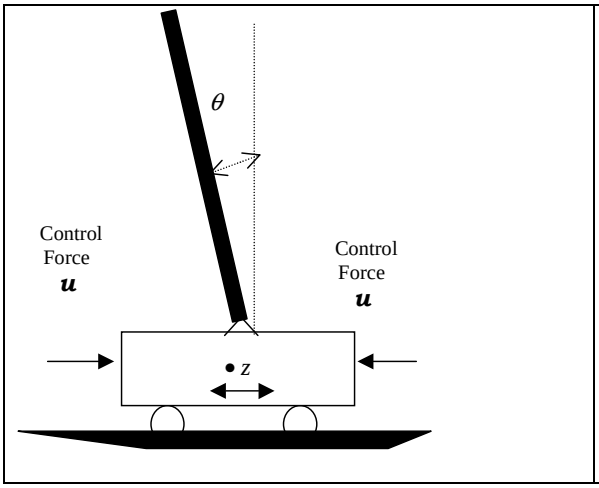
In Fig.8-6 the inference technique based on *product-max method* of inference is shown.

The output fuzzy set's membership function of Rule1 is constructed as the product of the membership function of consequence and ω = 0.6 $\times$ 0.8 = 0.48. (set C_1')

The output fuzzy set' membership function of Rule2 is constructed as the product of the membership function of consequence and ω = 0.3 $\times$ 0.2 = 0.06. (set C_2')

The crisp value of a motor voltage is defined as x -component of a *centroid point* (or *center of gravity point*).

Example 2: Inverted pendulum control problem.
Consider the task of a fuzzy controller development maintaining an inverted pendulum in the vertical position despite external disturbances.
This is a classical control problem. In Fig. 8-7 a graphical representation of the problem is shown.



System Conditions:
A pendulum is fixed to a cart which can move in two directions: *forward* and *backward*.

Control problem:
keep the pendulum standing vertical.

Figure 8-7. The inverted pendulum system.

(Copia modificata per una migliore consultazione on-line)

To keep the pendulum (or pole) balanced, a force u should be applied to the cart to move forward or backward continuously according to the current state of pendulum.

There are two inputs to the control system: θ (Theta) is the angle between the pendulum and vertical axis; $\dot{\theta} = \Delta\theta$ (Dtheta) is the rate of change of the Theta angle (angular velocity).
The output of the control system is the value of Motor Current.

Classical nonlinear control method is based on mathematical formulas to calculate the values of the output control. The inverted pendulum problem control is described by second-order differential equations for calculating the force to be used for moving the cart:

$$\ddot{\theta} = \frac{g\sin\theta + \cos\theta\left(\dfrac{-u/ml\dot{\theta}^2\sin\theta}{m_c+m}\right)}{l\left(\dfrac{4}{3} - \dfrac{m\cos^2\theta}{m_c+m}\right)}, \quad \ddot{z} = \frac{u + ml(\dot{\theta}^2\sin\theta - \ddot{\theta}\cos\theta)}{m_c+m}, \quad (8\text{-}1)$$

where g is the acceleration due to gravity (usually $9.8\, m/\sec^2$), m_c is the mass of the cart, m is the mass of the pendulum, l is the half-length of the pendulum, and u is the applied force in Newtons.

By defining the state vector $[x_1 x_2 x_3 x_4]^T = [\theta\ \dot{\theta}\ z\ \dot{z}]^T$, we can put the preceding equations (8-1) into standard format for state equations:

$$\dot{x} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix}^T = f(x,u) = \begin{bmatrix} x_2 \\ \dfrac{g\sin\theta + \cos\theta\left(\dfrac{-u/ml\dot{\theta}^2\sin\theta}{m_c+m}\right)}{l\left(\dfrac{4}{3} - \dfrac{m\cos^2\theta}{m_c+m}\right)} \\ x_4 \\ \dfrac{u + ml(\dot{\theta}^2\sin\theta - \ddot{\theta}\cos\theta)}{m_c+m} \end{bmatrix} \qquad (8\text{-}2)$$

(Note that $x_1 = \theta$ and $\dot{x}_1 = \dot{\theta}$. Also, $x_2 = \dot{\theta}$ by definition, so we have $\dot{x}_1 = x_2$. Similarly, $\dot{x}_3 = x_4$.)

The task of control is to find the control law $u = \phi(x)$ of a controller that maps a state vector x (or an error signal $x_d - x$) into an appropriate force u, such that a control goal can be achieved in a satisfactory manner.

Usual control goals for the inverted pendulum system include the following:
- Keep the pole balanced, regardless of the cart position;
- Keep the pole tracking a desired signal, regardless of the cart position.

Above we discussed the difficulties in using classical control method based on mathematical formulas. In order to avoid them, we will use fuzzy logic–based approach to control. *So, instead of classical modeling we will use fuzzy modeling.*
A fuzzy controller (FC) of the inverted pendulum is much easier to develop. Discuss main steps of FC- design (Fig. 8-8).

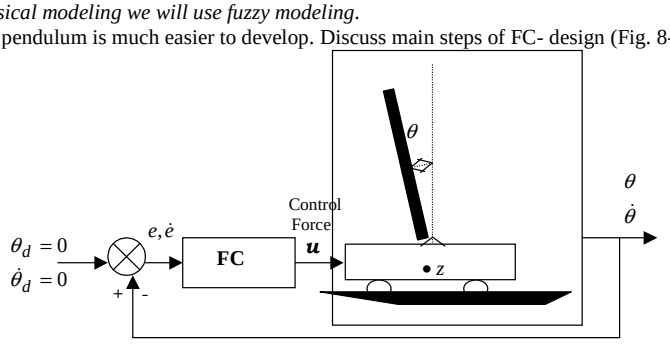
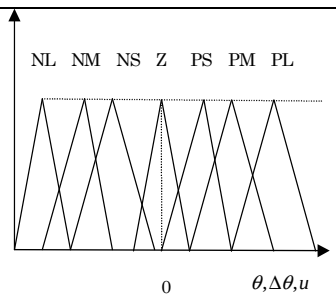


Figure 8-8. The structure of the Fuzzy Controller.

We will describe input-output parameters (θ (Theta) and $\dot{\theta} = \Delta\theta$ (Dtheta)) by the following membership function:
Z - Zero, NS - Negative Small, NM - Negative Medium, PS - Positive Small, PM - Positive Medium.
Membership functions and fuzzy rules are shown in Fig.8-9.



(a)

IF θ is PM and $\Delta\theta$ is Z	THEN u is PM
IF θ is PS and $\Delta\theta$ is PS	THEN u is PS
IF θ is PS and $\Delta\theta$ is NS	THEN u is Z
IF θ is NM and $\Delta\theta$ is Z	THEN u is NM
IF θ is NS and $\Delta\theta$ is NS	THEN u is NS
IF θ is NS and $\Delta\theta$ is PS	THEN u is Z
IF θ is Z and $\Delta\theta$ is Z	THEN u is Z

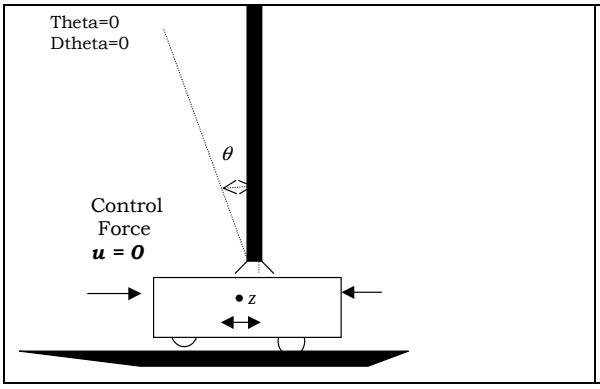
(b)

Figure 8-9. Input/Output membership functions (a) and fuzzy rules for given control task (b).

(Copia modificata per una migliore consultazione on-line)

Remark 1. Note that in this example the membership functions for Theta, DTheta and Motor Current are the same, although in general case they may be differ.

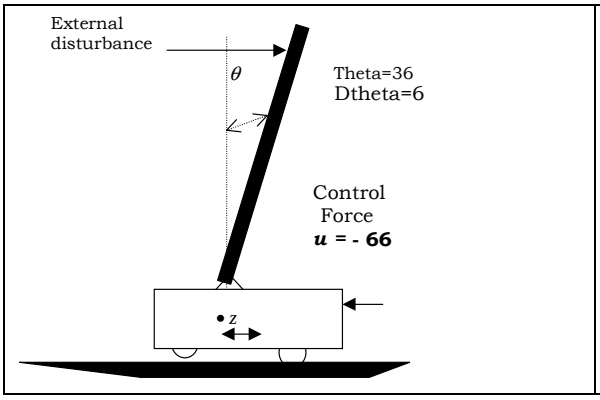
In Fig.8-10 the simulation system of the inverted pendulum control is shown. Four control situations are considered: a) fuzzy control in equilibrium; b) fuzzy control when some impulse to right (external disturbance) is given; c) fuzzy control after the impulse.



	NM	NS	Z	PS	PM	
NM			PM			
NS			PS	Z		
Z	PM	PS	Z	NS	NM	
PS		Z	NS			
PM			NM			

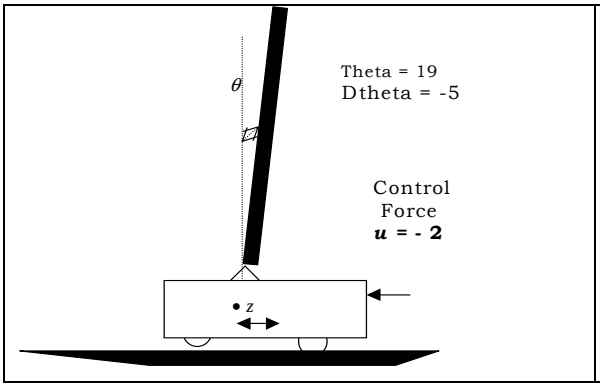
Look-up-table of FC.

(a) Fuzzy logic controller in equilibrium.



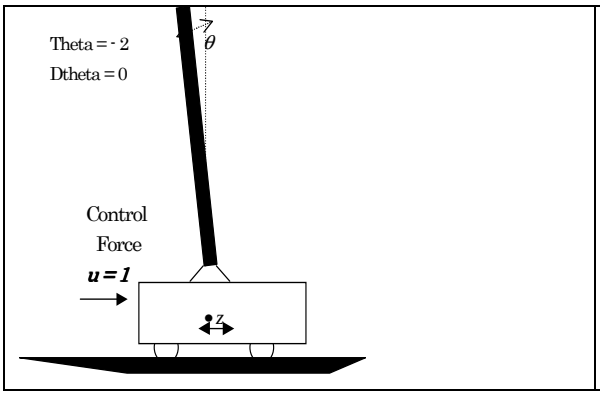
	NM	NS	Z	PS	PM	
NM			PM			
NS			PS	Z		
Z	PM	PS	Z	NS	NM	
PS		Z	NS			
PM			NM			

(b) Fuzzy Control after impulse to right (external disturbance).



	NM	NS	Z	PS	PM	
NM			PM			
NS			PS	Z		
Z	PM	PS	Z	NS	NM	
PS		Z	NS			
PM			NM			

(c) Fuzzy Control recovering from impulse.



	NM	NS	Z	PS	PM	
NM			PM			
NS			PS	Z		
Z	PM	PS	Z	NS	NM	
PS		Z	NS			
PM			NM			

(d) Fuzzy control near an equilibrium

Figure 8-10. Fuzzy Controller for inverted pendulum control.

Comment of the simulation results. In the right part of the pictures (a,b,c,d) *look-up table* of FC is shown. In the look-up table input membership functions associated with Theta and Dtheta are located along the horizontal and vertical axes, respectively. Interior squares of the table represent consequents of fuzzy rules ("then" parts).

In the case (a) the Fuzzy Control in an equilibrium is described. In this case both Theta and DTheta are zero, and the rule with zero Motor Current is active (see light square).

The output value is equal 0.

(Copia modificata per una migliore consultazione on-line)

In the case (b) the Fuzzy Control after impulse is described. The pendulum is moving to the right. In this case Theta =36, DTheta = 26. So, both PS and PM membership functions are activated for the Theta, and two rules for output are active (see light squares, where output NS and NM membership functions for Motor Current are shown). The output value of Motor Current is equal -66.

In the case (c) the pendulum is moving back. In this case three Motor Current membership functions are active: PS, NS and Z. The output value is equal -2.

In the case (d) the pendulum is passed its center position and is moving slightly to the left. In this case two rules are active with PS and Z Motor Current membership functions. The output value is equal +1.

Example 3: Fuzzy control of a double inverted pendulum

Consider a control of a double inverted pendulum (Fig. 8-11) based on two fuzzy controllers.

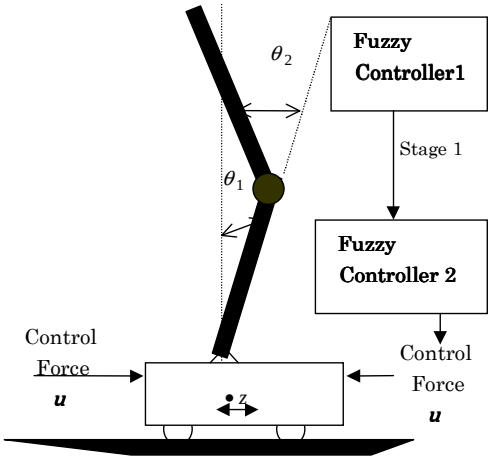


Figure 8-11. A double inverted pendulum

	$\dot{\theta}_2$	NL	NM	Z	PM	PL
θ_2						
NL		NL	NL	NL	NM	NS
NM		NL	NL	NM	NS	PS
NS		NL	NM	NS	PS	PM
PS		NM	NS	PS	PM	PL
PM		NS	PS	PM	PL	PL
PL		PS	PM	PL	PL	PL

(a) Output: Stage 1

	$\dot{\theta}_1$	NL	NM	Z	PM	PL
θ_1						
NL		NL	NL	NL	NM	NS
NM		NL	NL	NM	NS	PS
NS		NL	NM	NS	PS	PM
PS		NM	NS	PS	PM	PL
PM		NS	PS	PM	PL	PL
PL		PS	PM	PL	PL	PL

(b) For the input "Stage" = Z; Output: Motor Current

Figure 8-12. Look-up Tables for Fuzzy Controller 1 (a) and Fuzzy Controller 2 (b).

Fuzzy Controller 1 has two inputs: angle θ_2 and angular velocity $\dot{\theta}_2$. The output is called "Stage 1" and is used as input variable to Fuzzy Controller 2.

Fuzzy Controller 2 (FC 2) takes as input three variables: "Stage 1" and angle θ_1 and angular velocity $\dot{\theta}_1$. The output of FC 2 is the value of a Control Force (Motor Current).

For each output value of the variable "Stage 1" a separate look-up table is used to represent the rules for FC 2. The look-up tables for FC 1 and FC 2 for Stage = Z (zero) are given in Fig.8-12.

Fuzzy PID Control

The structure of a Fuzzy PID-control system is shown in Fig.8-13.

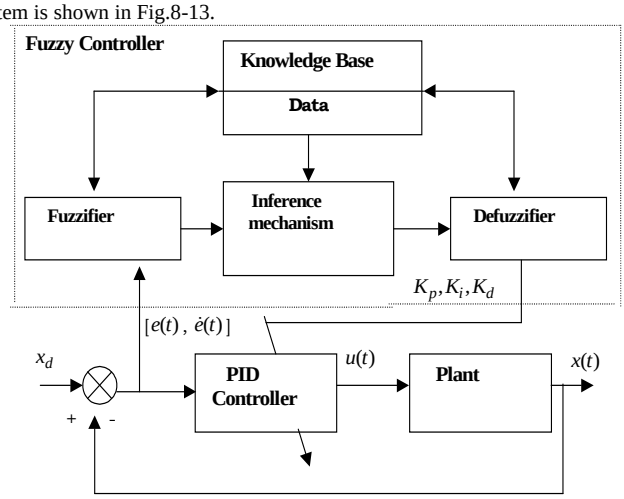


Figure 8-13. Block diagram of Fuzzy PID control system.

(Copia modificata per una migliore consultazione on-line)

The input-output relation of the PID controller is expressed as

$$u = K_p e + K_i \int_o^t e dt + K_d \dot{e},$$

where u is the control signal, e is the error signal, $\dot{e}$ is the derivative of error, and K_p, K_i, K_d denote proportional gain, the integral gain and the derivative gain, respectively.

If different values of K_p, K_i, K_d are chosen, then it is obvious that various responses of the plant will be obtained.

From Fig.8-13, it is follows that *input variables* for fuzzy reasoning are the *error and derivative of error,* and *output variables* of fuzzy reasoning are K_p, K_i, K_d *parameters*.

If n_1 and n_2 are the numbers of membership functions for the error and its derivative, then there will be $n_1 \times n_2$ fuzzy rules expressed as:

$$\begin{aligned}
&\text{If } e \text{ is } A_1 \text{ and } \dot{e} \text{ is } B_1 \text{, then } K_p = C_{11}, K_i = D_{11}, K_d = E_{11} \\
&\qquad \vdots \\
&\qquad \text{If } e \text{ is } A_1 \text{ and } \dot{e} \text{ is } B_{n_2} \text{, then } K_p = C_{1n_2}, K_i = D_{1n_2}, K_d = E_{1n_2} \\
&\text{If } e \text{ is } A_2 \text{ and } \dot{e} \text{ is } B_1 \text{, then } K_p = C_{21}, K_i = D_{21}, K_d = E_{21} \\
&\qquad \vdots \\
&\qquad \text{If } e \text{ is } A_2 \text{ and } \dot{e} \text{ is } B_{n_2} \text{, then } K_p = C_{2n_2}, K_i = D_{2n_2}, K_d = E_{2n_2} \\
&\qquad \vdots \\
&\qquad \text{If } e \text{ is } A_{n_1} \text{ and } \dot{e} \text{ is } B_{n_2} \text{, then } K_p = C_{n_1 n_2}, K_i = D_{n_1 n_2}, K_d = E_{n_1 n_2},
\end{aligned} \tag{8-3}$$

where $A_1, A_2, ..., A_{n_1}$ and $B_1, B_2, ..., B_{n_2}$ are membership functions of e and $\dot{e}$, and $C_{11}, ..., C_{n_1 n_2}$, $D_{11}, ..., D_{n_1 n_2}$ and $E_{11}, ..., E_{n_1 n_2}$ are real numbers that satisfy

$$\begin{aligned}
K_{p,\min} \le C_{ij} \le K_{p,\max}, 1 \le i \le n_1, 1 \le j \le n_2 \\
K_{i,\min} \le D_{ij} \le K_{i,\max}, 1 \le i \le n_1, 1 \le j \le n_2 \\
K_{d,\min} \le E_{ij} \le K_{d,\max}, 1 \le i \le n_1, 1 \le j \le n_2.
\end{aligned}$$

According to Sugeno fuzzy inference model and fuzzy rules (8-3), the outputs will be as follows:

$$K_p = \frac{\left(\sum_{i=1}^{n_1}\sum_{j=1}^{n_2} \omega_{ij} C_{ij}\right)}{\left(\sum_{i=1}^{n_1}\sum_{j=1}^{n_2} \omega_{ij}\right)}, \quad K_i = \frac{\left(\sum_{i=1}^{n_1}\sum_{j=1}^{n_2} \omega_{ij} D_{ij}\right)}{\left(\sum_{i=1}^{n_1}\sum_{j=1}^{n_2} \omega_{ij}\right)}, \quad K_d = \frac{\left(\sum_{i=1}^{n_1}\sum_{j=1}^{n_2} \omega_{ij} E_{ij}\right)}{\left(\sum_{i=1}^{n_1}\sum_{j=1}^{n_2} \omega_{ij}\right)} \tag{8-4}$$

where $\omega_{ij} = A_i(e) \cdot B_j(\dot{e})$.

Example 4: Fuzzy PID control of a ball and beam system.

Consider the fuzzy PID control of the ball and beam system shown in Fig. 8-14.

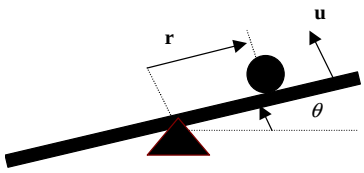


Figure 8-14. A ball and beam system.

We assume that there are five membership functions for e and $\dot{e}$, represented as shown in Fig.8-15.

(Copia modificata per una migliore consultazione on-line)

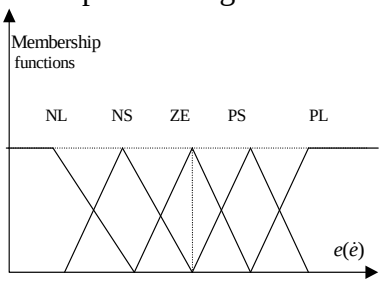


Figure 8-15. Membership functions for e and $\dot{e}$.

For the ball and beam system shown above, the mathematical model is given by the following equation:

$$0 = \left(\frac{J_b}{R^2} + M\right)\ddot{r} + MG\sin\theta - Mr\dot{\theta}^2$$

$$\tau = (Mr^2 + J + J_b)\ddot{\theta} + 2Mr\dot{r}\dot{\theta} + MGr\cos\theta,$$

where τ is the torque applied to the beam, θ, J are the angle and the moment of inertial of the beam, M, J_b, R, r are the mass, the moment of inertial, the radius and the position of the ball, respectively, and G is the acceleration of gravity.

If $r, \dot{r}, \theta, \dot{\theta}$ are chosen as state variables, then by defining

$$B = \frac{M}{\left(\frac{J_b}{R^2} + M\right)}, \quad C = -(Mr^2 + J + J_b)$$

state equations of the system will be expressed as

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} x_2 \\ B(x_1 x_4^2 - G\sin x_3) \\ x_4 \\ (2Mx_1x_2x_3 + MGx_1\cos x_3)/C \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ -1/C \end{bmatrix}\tau, \qquad (8\text{-}5)$$

where $[x_1 x_2 x_3 x_4] = [r\ \dot{r}\ \theta\ \dot{\theta}]$.

In classical PID control the *control problem* for this task is as follows:

Based on mathematical model (8-5), generate the desired torque (control) such that the ball and beam system can be driven from initial state $[r\ \dot{r}\ \theta\ \dot{\theta}] = [0.3m\ \ 0\ \ -0.6rad\ \ 0]$ to the zero final state $[r\ \dot{r}\ \theta\ \dot{\theta}] = [0\ 0\ 0\ 0]$.

Since both the state variables r and θ are to be driven to zero, the input-output relation of the PID controller will be given as (here $\tau = u$)

$$u = K_{p_1}e_1 + K_{i_1}\int_o^t e_1 dt + K_{d_1}\dot{e}_1 + K_{p_2}e_2 + K_{i_2}\int_o^t e_2 dt + K_{d_2}\dot{e}_2,$$

where $e_1 = r_d - r, \quad e_2 = \theta_d - \theta$.

We assume that $K_{p_1}, K_{i_1}, K_{d_1}$ are determined by $e_1, \dot{e}_1$ and $K_{p_2}, K_{i_2}, K_{d_2}$ are determined by $e_2, \dot{e}_2$, respectively.

In fuzzy control, we select these parameters by using fuzzy modeling including expert fuzzy rules base design *(8-3)* and fuzzy computation *(8-4)*.

Fuzzy rules for defining $K_{p_1}, K_{i_1}, K_{d_1}$ and $K_{p_2}, K_{i_2}, K_{d_2}$ parameters for PID controller of the ball and beam system are shown in Table 8-1 (a,b).

Table 8-1. Fuzzy rules for selecting parameters of the PID controller.

(a)

If e_1	*and* $\dot{e}_1$	*then* K_{P_1}	K_{I_1}	K_{D_1}
NL	*NL*	30.0000	0.0012	5.2500
NL	*NS*	64.0000	0.0071	7.6000
⋮	⋮	⋮	⋮	⋮
NS	*ZE*	4.0000	0.0006	0.3375
ZE	*ZE*	55.0000	0.0019	10.4844
ZE	*PS*	9.0000	0.0180	0.1688
PS	*PL*	32.0000	0.0064	5.800
⋮	⋮	⋮	⋮	⋮
PL	*PL*	16.0000	0.0007	3.0500…

(b)

If e_2	*and* $\dot{e}_2$	*then* K_{P_2}	K_{I_2}	K_{D2}
NL	*NL*	40.0000	0.0019	0.4375
NL	*NS*	15.0000	0.0014	0.1172
⋮	⋮	⋮	⋮	⋮
NS	*ZE*	38.0000	0.0033	0.5344
ZE	*ZE*	14.0000	0.0007	0.8531
ZE	*PS*	24.0000	0.0008	2.2500
PS	*PL*	26.0000	0.0260	1.4625
⋮	⋮	⋮	⋮	⋮
PL	*PL*	17.0000	0.0007	1.1688…

Remark. In next lectures we will discuss how to construct automatically these look-up tables for fuzzy PID control.

(Copia modificata per una migliore consultazione on-line)

Example 5: Fuzzy Control in Autonomous Robot Navigation Task

The development of techniques for autonomous navigation in real-world environments is one of the major trends in current research in intelligent robotics. Important problem in autonomous robot's navigation is the need to cope with the large amount of uncertainty that is inherent for natural environment.

Real-world environments typically have complex and unpredictable dynamics: objects can move, other agents can modify the environment, and relatively stable features may change with time (for example, seasonal variations). Prior knowledge about environment is, in general, incomplete, uncertain, and approximate. For example, maps typically omit some details and temporary features, spatial relations between objects may have changed, and the metric information may be imprecise and inaccurate.

Fuzzy logic has features that make it adequate tools for the task of autonomous navigation.

The robot uses two kinds of sensors: *external* and *internal*. By using external sensors, like a CCD-camera and sonar or infrared sensors, robot observes the state of environment. By using internal sensors, like compass and shaft encoders on the wheels, robot has information about the state of its own body.

Autonomous navigation task consists of two levels: *modeling and planning* and *execution*. Planning module constructs a plan of actions that will perform the given task in the given environment. The execution layer performs this plan. Performance of each action is decomposed into small independent decision-making processes, or *behaviors*. Each behavior implements a control strategy for one specific subtask, like following a path, avoiding sensed obstacles, or opening and crossing a door, etc. Consider some of these tasks and their realization based on fuzzy logic techniques.

Consider example of behavior called as *reactive navigation*: following a given path while avoiding unforeseen obstacles in real time. This behavior consists of two parts.

Path tracking behavior (or control): bring the robot from initial position to a target position following a planned path.

This behavior may be described by the following general rule:

$$IF\ \{path-condition_1\}\ THEN\ \{command_1\}$$

Obstacle avoidance behavior (or control): avoid unforeseen obstacles in real time.

This behavior may be described by the following general rule:

$$IF\ \{obstacle-condition_1\}\ THEN\{command_1\}$$

Behavior coordination problem

The problem of behavior coordination is one of the major open issues in behavior-based approaches to robotics. This problems consists of two different subproblems:

1) how to decide which behavior(s) should be activated at each time moment;
2) how to combine the results from different behaviors into one command to be sent to the robot's effectors (possibly, taking weights into account).

First problem is called *behavior arbitration* problem, second one is called *command fusion* problem.

Command fusion

The simplest way to fuse the commands from different behaviors is to use a switching scheme: the output from one behavior is selected for execution, and all the others are ignored. Which output is selected depends on the arbitration strategy.

Consider example of behavior called as *reactive navigation*: following a given path while avoiding unforeseen obstacles in real time.

A general rule of command fusion for this type of behavior can be described as follows:

$$\begin{array}{l} IF\ \ (obstacle\ is\ close)\ THEN\ \{Avoid\text{ - }obstacle\} \\ IF\ \ (obstacle\ \ is\ \ far)\ THEN\ \{Go\text{ - }to\text{ - }Target\} \end{array}.$$

Reactive Obstacle avoidance control

A typical fuzzy control rule for this task looks as:

IF (*obstacle-right-distance* is *close*) AND (*obstacle-left-distance* is *far*) THEN {*turn left*}

IF (*right-distance* is *close*) AND NOT(*obstacle-left*) THEN {*turn left*},

where "*obstacle-right-distance*"," *obstacle-left-distance*", "*obstacle-left*" are linguistic variables described by fuzzy sets.

Remark. The rich bibliography of considered problem is contained in [17].

So, we considered a few examples of intelligent control system design based on ideas of fuzzy logic that are closer to human control strategies. Finally, discuss the following important for control theory problems.

Robustness, Stability and Controllability of Control Systems

Control system is *robust*, if changing the parameters of control object does not necessarily require changing the control system.

Fuzzy Controller is robust control system, if changing of the parameters of the control object does not change the set of fuzzy rules.

For example, FC developed for inverted pendulum control is a robust fuzzy control of inverted pendulum. Consider the following case. Let us put a glass of wine on the top of pendulum. Experiments with given fuzzy controller demonstrated that without changing the set of rules, the pendulum with the glass of wine was balanced.

Let us replace the glass of wine with a live mouse, for example, and so on. Must we also change the set of fuzzy rules?

There is a very important question: *how much the parameters may be changed without the need for changing the fuzzy rules*. This problem is connected with the *stability problem* of a control system. The feedback control system must maintain stability when the control object is subjected different external disturbances or changes.

If for every bonded control signal input, the output state of a controlled object is bounded, then the control system is called *stable*. An *unstable system* has an output that grows without bound (exponentially, for example).

The question of stability is clearly defined for linear dynamic systems. In nonlinear dynamic systems the problem of stability is more complex, and different methods for stability analyzing of nonlinear systems are developed, for example, *Lyapunov's stability methods* and *entropy production-based method* [18].

Stability of linear systems depends only on the system's parameters, and doesn't depend from initial conditions. But the stability of nonlinear systems depends on the initial conditions as well as system's parameters. Nonlinear system can exhibit a stable response (output) to one type of input, and unstable response to another type of input.

The concept of controllability is also very important in control theory. A system is completely *controllable* if there exists a control which transfers every initial state (input) at $t = t_0$ to any final state (output) at $t = T$ for all t_0 and T.

Controllability property means that by a control system we can stabilize a controlled object (that is, the controlled object can reach an equilibrium state).

(Copia modificata per una migliore consultazione on-line)

Lecture N 9 Genetic Algorithms: Theoretical Backgrounds and Applications

Let us discuss how to improve the capabilities of fuzzy controllers based on a fuzzy logic approach.

Fuzzy logic approach enables us to translate qualitative knowledge about the problem into a reasoning system capable of performing approximate pattern matching and interpolation. But, in fuzzy logic based technology the generation of membership functions (MF) and fuzzy rules (FR) is a task mainly done by human expert. Human expert also solves the task of refining (or tuning) of knowledge base.

Fuzzy logic approach does not have adaptation and learning capabilities for self-constructing (automatic constructing) of MF's and FR's for fuzzy controller.

These tasks can be successfully solved by using *Genetic Algorithms* (GA) and *Artificial Neural Networks* (ANN), another Soft Computing technology.

So, a parameter's tuning step in a fuzzy controller's design can be performed on the base of GA technology.

Before GA definition, let us refresh traditional schemes of finding solution in optimization task and compare them with a human like optimization.

Main Schemes of Optimization

1. *Calculus-based optimization* is based on gradient descent methods. In this case we seek optimum of a given function by using a gradient of the function and have to solve a nonlinear differential equation.
2. *Enumerative* schemes investigate every point in a search space, and find better. This method requires a lot of time, and not efficient for large dimension tasks.
3. *Random search algorithms* investigate randomly all space of solution, and find a solution. They are also not effective on large dimensions of solutions space.

Let us discuss now how people solve optimization tasks.

4. *Human like optimization*

Using conventional search algorithms we ask every time: does the method reach an optimum or no? In other words, we must prove the convergence to the optimum. So, the goal of classical optimization is the achievement of the optimum itself. Consider human-like optimization behavior. In our common life, we often try to do something better and better relative to the past performance. But, convergence to the best is not an issue in a human optimization behavior. The goal is to find a "good enough" solution. So, in the human like optimization *the most important goal of optimization is improvement.* GAs use this idea.

What is Genetic Algorithm?

John Holland was the first researcher who in the 1970 developed and analyzed genetic algorithms [19]. Genetic Algorithms are global optimum search procedures with a probabilistic component based on the *mechanism of natural evolution.*

What is a natural evolution?

Natural evolution is a process that operates on *chromosomes*, which are special organic devices.

Chromosomes encode the structure of living beings (individuals). The set of individuals is called a *population.*

The population evolves over time through competition: *survival of the fittest.* This is the biological law of evolution.

Three main processes characterize the natural evolution: *selection, recombination and mutation.*

Natural selection processes are as follows:

1) link chromosomes with the performance (*fitness*) of their decoded structure ;
2) define the chromosomes that encode *successful structures* (which have a maximum values of fitness) and
3) cause those better chromosomes to reproduce more often than a chromosomes encoding not successful structures.

Recombination process creates different chromosomes in children by combining material from the chromosomes of the two parents.

Mutation process causes the chromosomes of children to be different from the chromosomes of their parents.

GA incorporates these features of natural evolution in computer algorithms to solve difficult problems of optimization through evolution.

Consider main idea of GA by using a simple example.

Let we define two point A and B in some space, and we want to find an optimal path from A to B in this space (for example, the path with a minimum of time). In this case GA manipulates with a set of all potential solution of the problem. GA considers all path's from A to B, and by using stochastic (random) mechanism finds a subset of good path's so that an optimal solution is contained in this subset.

Main paradigm of GA:

GA investigates a search space, finds and maintains a population of individual's structures that represent candidates of optimal solution to the current problem.

GA can be applied to a wide range of optimization and learning problems, including routing and scheduling, machine vision, control systems design and so on.

We consider the using of GA for the task of fuzzy rules and members hip function design for fuzzy controllers.

Let us discuss the basic structure and mechanism of GA.

Basic Structure and Mechanisms of GA

Step 0. Coding

GA operates on a coding of the parameters of the problem. Thus, at first, the parameters of the problem must be encoded in finite length strings (like chromosomes).

A chromosome can be considered as a vector x consisting of l genes:

$$x = (a_1 a_2 \ldots a_l),\ a_i \in A_i,$$

where l is the length of the chromosome , A_i is the alphabet . Commonly, all A_i is the same, that is $A_1 = A_2 = \ldots = A_l = A$.

If A = {0,1} then chromosome is represented by *binary genes*.

If A = *R* {real numbers} then chromosome is represented by *real-valued genes*. Further the following steps are performed by GA.

Step 1: *Initial population construction*

The initial population can be initialized using whatever knowledge is available about possible solutions. In the absence of such knowledge, the initial population should represent a random sample of search space.

Randomly generate an initial population $X(0) := (x_1, x_2, \ldots, x_n)$.

Step 2: *Fitness evaluation*

Compute a fitness $f(x_i)$ of each individual x_i in the current population $X(t)$.

In the step 2 each member of population is evaluated and assigned a measure of its *fitness* as a solution. Fitness can be measured by using some fitness function (called also as objective function, or evaluation function).

When each individual in the population has been evaluated, a new population of individuals is formed in two steps (step3 and step4).

Step 3: *Selection*

(Copia modificata per una migliore consultazione on-line)

Generate an intermediate population $X_r(t)$ (called also as a *Mating pool* or a set of possible parents) applying the *reproduction (selection) operator*.

In this step individuals in the current population are selected for replication (reproduction, copy) based on their relative fitness. Individuals with high relative fitness ("good" individuals) might be chosen several times for replication, while individuals with low relative fitness ("bad" individuals) might not be chosen at all.

The probability $p(x_i)$ that an individual x_i will be copied into the next generation depends upon of the ration of its fitness value $f(x_i)$ to the total fitness, *F*, of all individual in the population. This ratio ($f(x_i)$ / *F*) is called a *relative fitness*.

The reproduction is done by conducting a series of random trials in which each string is copied to the intermediate population a number of times that is proportional to the value of its relative fitness. This random procedure can be, for example, like a *Monte Carlo random procedure*, called also as "wheel of fortune" (or Roulette wheel) (see Fig.9-1).

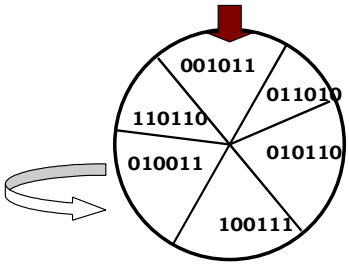


Figure 9-1. Roulette wheel procedure (or "Wheel of fortune").

Each chromosome (in the roulette procedure) occupies an area that proportional to its relative fitness.

Expected number of times ($E_{select}(x)$) that chromosome x will be selected is given as follows:

$$E_{select}(x) = NP_{select}(x), \qquad (9\text{-}1)$$

where *N* is the population size; and

$$P_{select}(x) = \frac{f(x)}{\sum_{i=1}^{N} f(x_i)}. \qquad (9\text{-}2)$$

(We approximate the value of (9-1) to the nearest integer.)

Remark 1. Besides the "roulette wheel" procedure, may be another methods of selections. For example:

- *uniform selection*: each chromosome has an equal chance of selection regardless of its fitness;
- *tournament selection*: small number of chromosomes is uniformly chosen, after they compete with each other on the basis of their fitness.
- *Selection with elitism*: the fittest chromosome is transferred to the next generation without change. Then random selection is performed over last chromosomes.

In the absence of any other mechanism, the resulting selective procedure would cause the best individuals to occupy a larger and larger proportion in the population over time.

Step 4: *Crossover and Mutation*

Generate the population $X(t+1)$ applying genetic operators to the $X_r(t)$.

In this step the selected individuals (from $X_r(t)$)) are altered using a *genetic operators* to form a new set of individuals for evaluation. Consider two main genetic operators: *crossover* and *mutation*.

Crossover Operation

The primary genetic search operator is the *crossover operator*, which performs the following functions: 1) selects two parent's individuals, 2) mates (or combines) the features of these two parents and 3) forms two similar *offspring (children)*.

There are many possible forms of crossover. The simplest is the following. Pairs of parents are selected randomly. For each pair of parents a *point of crossover* (one or two, or a few) is selected also randomly. This point indicates how many bits on the right end of each string should be interchanged.

For example, if the parents are represented by the lists

$(a_1 a_2 \updownarrow a_3 a_4 a_5)$ and $(b_1 b_2 \updownarrow b_3 b_4 b_5)$,

and the point of crossover is shown by the symbol "$\updownarrow$",

then the crossover-operator produces the following offspring:

$$(a_1 a_2 b_3 b_4 b_5) \text{ and } (b_1 b_2 a_3 a_4 a_5).$$

You can see that the <u>*contiguous groups of bits at the right end of two strings are interchanged their values*</u>.

Remark 2. There are another types of crossover, for example, those shown below.

a) *Uniform crossover*:

parents :	*offspring* :
$(A_1 B_1 \updownarrow C_1 D_1 \updownarrow E_1 \updownarrow F_1)$	$(A_1 B_1 C_2 D_2 E_1 F_2)$
(001101) - *a crossover mask*)	
$(A_2 B_2 \updownarrow C_2 D_2 \updownarrow E_2 \updownarrow F_2)$	$(A_2 B_2 C_1 D_1 E_2 F_1)$

In the uniform crossover, a bit string called a *crossover mask* is used to generalize the rossover process. 1 bit in this mask indicates that corresponding bits in the parents are to be interchanged; bit 0 indicates no bit interchange.

b) *Linear interpolation 2-point crossover*:

(Copia modificata per una migliore consultazione on-line)

parents : **offspring :**

$$(A_1 B_1 C_1 \updownarrow D_1 E_1 \updownarrow F_1) \Rightarrow \left(\frac{2A_1 + A_2}{3} \; \frac{2B_1 + B_2}{3} \; \frac{2C_1 + C_2}{3} \; \frac{D_1 + 2D_2}{3} \; \frac{E_1 + 2E_2}{3} \; \frac{2F_1 + F_2}{3}\right)$$

$$(A_2 B_2 C_2 \updownarrow D_2 E_2 \updownarrow F_2) \Rightarrow \left(\frac{A_1 + 2A_2}{3} \; \frac{B_1 + 2B_2}{3} \; \frac{C_1 + 2C_2}{3} \; \frac{2D_1 + D_2}{3} \; \frac{2E_1 + E_2}{3} \; \frac{F_1 + 2F_2}{3}\right)$$

In generating new individuals for testing, the crossover operator usually based only on the information present in the structures of the current individuals. If a specific information is missing, due to storage limitations or loss incurred during the selection process of a previous iteration, then *crossover can not produce new structures* that contain it.

Mutation

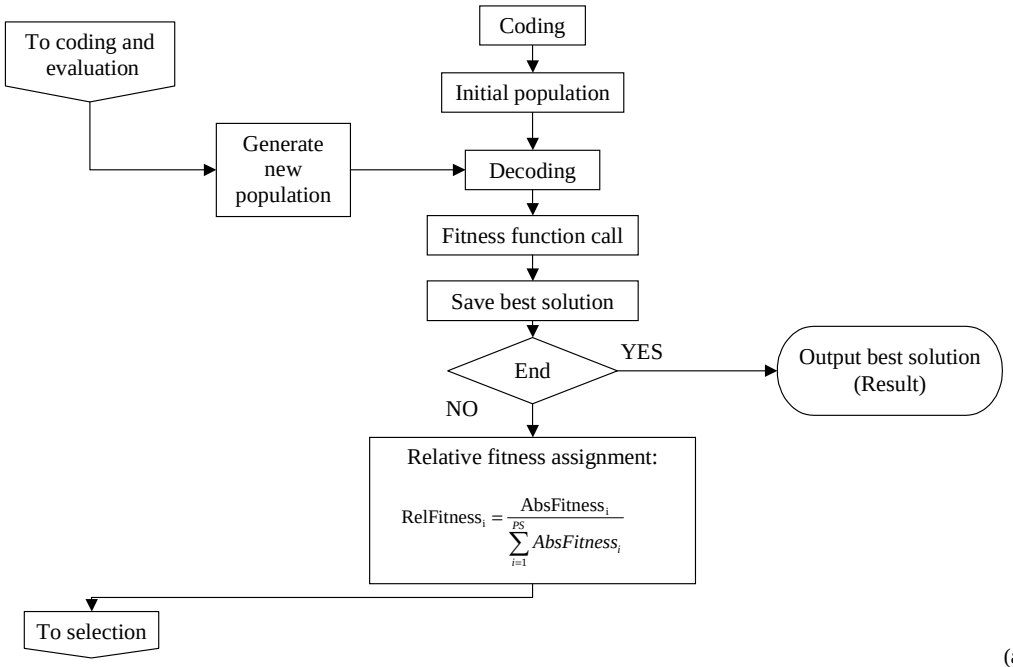
A *mutation operator* which alters one or more components of a selected structure, provides the way to introduce new information into the population.
A wide range of mutation operators have been proposed, ranging from completely random alterations to more heuristically motivated local search operators.
Mutation operator serves as secondary search operator that allows GA to investigate all points in the search space.
The resulting offspring are then evaluated and inserted back into population, replacing older members.
Specific decisions about how many members are replaced during each iteration, and how members are selected for replacement, define a range of alternative implementation.
Step 5: *Checking of End_Test*
$t = t+1$; IF NOT (End_Test) THEN go to Step2 ; else stop.
The "End_Test" describes the condition of finishing (termination) of GA. It is a stopping criteria. The "End_Test" is usually given by the number of generation (for example, t = 1,2,...,100), or by the time-length of work of GA (for example, 3 hours) , or may be some special convergence criteria.

So, final generation of individuals represents a solution of a given optimization task. In this case we say that GA converges to the optimal solution.
Convergence of GA means the situation when all of the chromosomes in the final population have the same gene values.

We have seen that even simple GA exhibits a sophisticated information processing capabilities.

Block-diagrams of GA are shown below in Fig.9-2 (a,b,c,d).

Coding and Evaluation

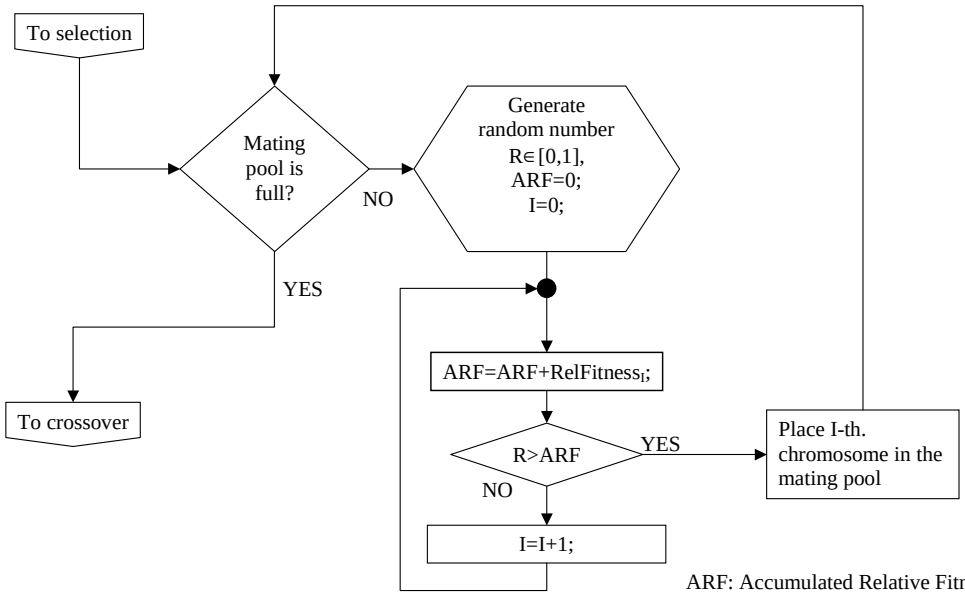


(a)

Figure 9-2, a. Coding and Evaluation steps of GA.

(Copia modificata per una migliore consultazione on-line)

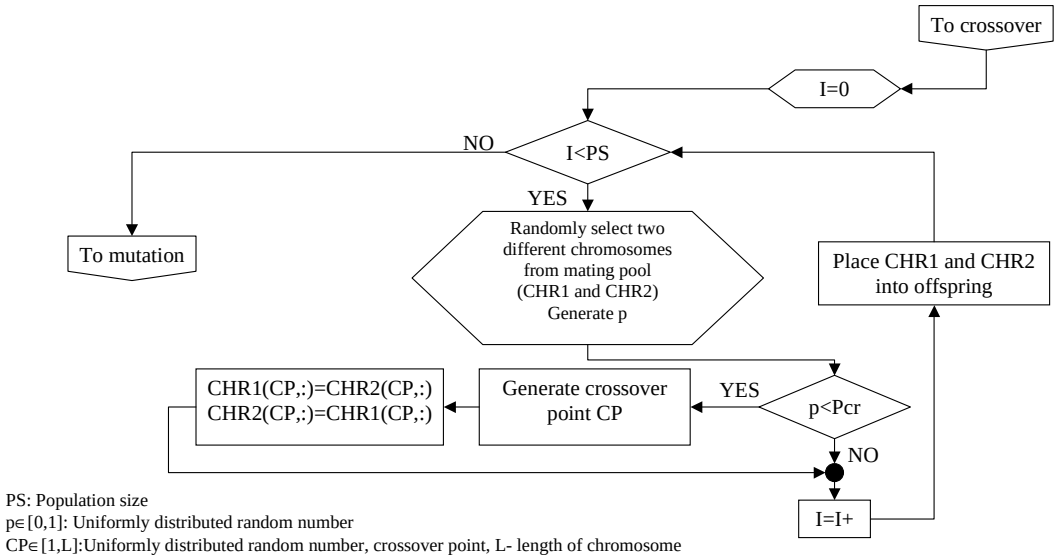
Selection



ARF: Accumulated Relative Fitness

(b)

Crossover

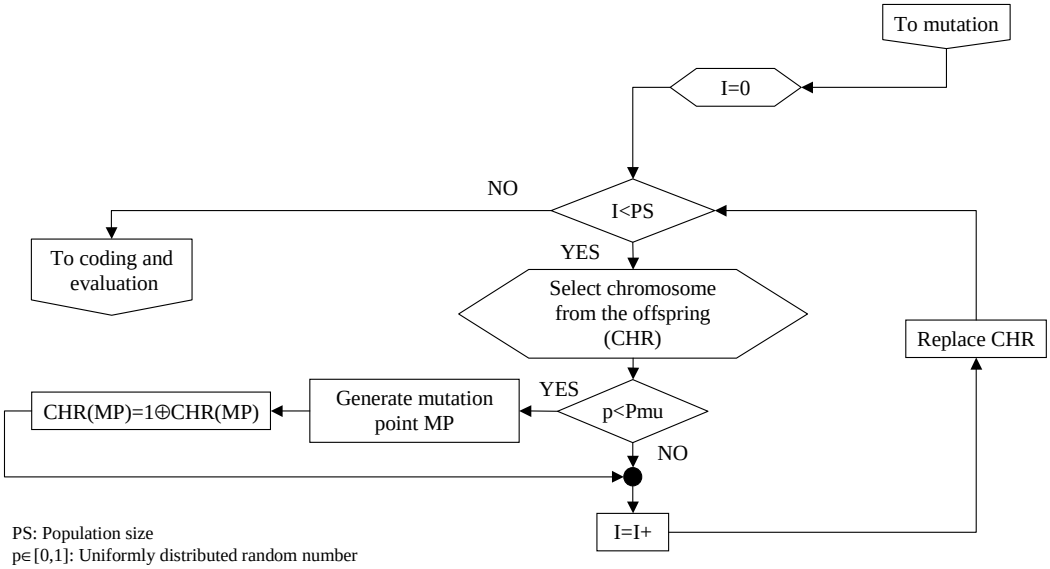


PS: Population size
p∈[0,1]: Uniformly distributed random number
CP∈[1,L]:Uniformly distributed random number, crossover point, L- length of chromosome

(c)

Figure 9-2. Selection and Crossover steps of GA.

Mutation



PS: Population size
p∈[0,1]: Uniformly distributed random number
MP∈[1,L]:Uniformly distributed random number, mutation point, L- length of chromosome

(d)

Figure 9-2. Block-diagrams of GA.

(Copia modificata per una migliore consultazione on-line)

Example of GA-based optimization

Consider now a simple example in order to understand how a genetic algorithm works. Let us consider the following task: optimize the function $f(x) = x^2$ over the interval [0-31] by using GA search mechanism. Thus, the problem for GA is to find the value of x which gives a maximum value of the given function $f(x)$ on the given interval.

At first we consider the set of parameters of the given task which must be optimized. It is a set of x from interval [0-31].
GA operates on a coding of the parameters. Thus, the parameters of the problem must be encoded in finite length strings. We will use a binary coding and consider a five-bit strings: { 00000 - 11111}.

Remark 3. A binary string of length n can represent 2^n individuals. So, if $n = 5$, then we can represent 1,2,3,...32 numbers ($2^5 = 32$) . Therefore in order to code all numbers from interval [0-31] a five-bit strings are enough ($n = 5$).

Let us consider initial population by using random procedure. Let the initial population pool (set) be a set with four individuals as shown in Table 9-1.

Table 9-1. Initial population.

initial population	x	$f(x)$ **(fitness)**	**relative fitness (percent of total)**
01101	13	169	14.4
11000	24	576	49.2
01000	8	64	5.5
10011	19	361	30.9
Sum - fitness		**1170**	**(100.0%)**

Remark 4. In Table 9-1 you can see a binary and decimal representations of numbers. The rule of transforming a binary number to a decimal number is shown below:

$$\underbrace{abcde}_{01234\ -\ positions\ -\ from\ -left\ -to\ -right} = a2^4 + b2^3 + c2^2 + d2^1 + e2^0 .$$

For example, $01101 = 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 8 + 4 + 1 = 13$.

Our next step is reproduction. Reproduction operator evaluates and selects pairs of strings for mating pool by the following way: one copy of string 01101, two copies of 11000, and one copy of 10011 are selected by using a roulette wheel method.
Next we apply the crossover operator as illustrated in Table 9-2. The crossover point is determined by a random number generator.

Table 9-2. Mating pool strings and crossover.

Mating pool	**Mates**	**Swapping**	**New population**
01101	1	0110 [1]	01100
11000	2	1100 [0]	11001
11000	2	11[000]	11011
10011	4	10 [011]	10000

After crossover the mutation operator are applied which may modify a random bit in the given string. Special probabilistic functions for determining the probability of mutation are used.
In Table 9-3 a new generation of string after mutation is shown (here you can see, that there is no mutation). Then the end-test is applied in order to define: continue the GA work or stop.
You see here that GA achieves very quickly (only by two generation) a solution which is very close to final (optimal) solution.

Table 9- 3. Next generation of strings.

initial population	x	$f(x)$ **(fitness)**	**relative fitness (percent of total)**
01100	12	144	8.2
11001	25	625	35.6
11011	27	729	41.5
10000	16	256	14.7
Sum - fitness		**1754**	**(100.0%)**

Finally, discuss the following main properties of GA.

Differences between Classical Methods of Optimization and GA

1) Conventional (classical) optimization techniques are based on finding of the parameters of the *mathematical model* of a problem which produce the global maximum (or minimum) of some given *objective function* for this problem.
   Genetic Algorithm itself *doesn't use a mathematical model*, it investigates populations of solutions of a problem, and operates by maintaining and modifying the characteristics of a population. GA can find a population of good solutions of the problem that contains an optimum solution of the problem.
2) Classical optimization techniques represent a class of *derivative - based methods*. This means that for optimal solution finding an objective function's derivative information is needed.
   GA represents derivative-free optimization. This means that for optimal solution's searching an objective function's derivative information is not needed. In this case GA can be used both for continuos and discrete optimization problems.
3) Classical methods operate with parameters of a given problem. *GA operates on a coding of the parameters.* Thus, the parameters of the problem must be encoded in finite length strings. The string may be a sequence of any symbols. The binary symbols "0" and "1" are often used in GA. A selection (choice) of a coding method is very important.

(Copia modificata per una migliore consultazione on-line)

4) GA optimization is performed on a set of string by using probabilistic mechanism and fitness measure for a string.
5) GA can operate without any knowledge of a search space.
6) GA provides a means to search an optimal solution in poorly understood and irregular spaces.

Theoretical Foundation of GA

Let us define simple genetic algorithms (SGA) formally.
Definition *9-1 : (*simple genetic algorithm*)*
SGA is defined as following 8-tuple:

$< C, F, P^0, \mu, \Omega, \Gamma, \Delta, \Psi >$,

where *C* is the genetic coding scheme of individual for a given problem;
F is the fitness (or evaluation, or objective) function to compute the fitness value of individual;
P^0 is the initial population;
μ is the size of population;
Ω is the selection operator;
Γ is the crossover operator;
Δ is the mutation operator;
Ψ is termination conditions (End_Test).

SGA usually apply a binary coded strings . We will say, that such strings are produced from the alphabet A = {0,1}. Each symbol in a string is identified by its bit position, where *bit position equal* 1 *means the first left symbol in a string*.

Definition 9-2: (a schema)
A schema S (the plural is schemata) is a string that is constructed from alphabet A' = {0,1,*} , where * indicates " don't care" .

For example, the following string is the schema:

S = * 1 * 0 * * 0 0. (9-3)

Definition 9-3: (a representative of a schema)
A string which matches a schema in all of its definite bit position (0 or 1) is called the representative of given schema.

The following strings A, B, and C represent schema S given in (9-3):
A = 1 1 0 0 1 0 0 0
B = 0 1 1 0 0 0 0 0
C = 1 1 1 0 1 1 0 0.
Definition 9-4: (an order of a schema)
The order of a schema , *o*(S), is the number of definite bit positions .
For the schema given in (9-3), *o*(S) = 4.

Definition 9-5: (a defining length of a schema)
The defining length of a schema S, δ (S) is the distance between the leftmost (b_{left}) and rightmost (b_{right}) bit positions holding either a 0 or a 1.

δ (S) is calculated as $b_{right} - b_{left}$.

For our example (9-3) , δ (S) = $b_{right} - b_{left}$ = 8-2= 6.

Implicit Parallelism of GA

Suppose that the strings are length l . Each string represents 2^l schemata. For example, the string 1101 has length equal 4, and therefore represents 2^4 =16 following schemata:
1100, 110*, 11*0, 11**, 1*00, 1*0*, 1**0, 1***, *100, *10*, *1*0, *1**, **00, **0*, ***0, 0***, ****.
If there is a population of *n* string, the total number of schemata, N_S is

$$2^l \le N_S \le n2^l .$$

Since each string can represent many schemata, it means that GA operations defined on a population of strings process a much larger number of schemata in parallel. This property is called *implicit parallelism of GA.*

Consider the following task [20].
Let some schema S has $n(S,t)$ representative strings in a population at time *t* . We will calculate how many representatives of the given schema will appear in the next generation:

$$n(S,t+1) = ?$$

This number depends upon the operations of reproduction, crossover, and mutation. The effects of each will be considered separately, and then combined.

Reproduction

According to the formula (9-2) the probability $p(x_i)$ that an individual x_i will be copied into the next generation depends upon of the ration of its fitness value $f(x_i)$ to the total fitness *F* of all individual in the population :

$$p(x_i) = f(x_i) / F .$$

Thus, the probability, $p(S_i)$, of a string S_i which is a representative of the schema *S* being copied to the next generation is defined also as:

(Copia modificata per una migliore consultazione on-line)

$$p(S_i) = f(S_i) / F .$$

where $f(S_i)$ is a fitness value of string S_i, F is a total fitness of all strings in the population.

According to the formula (9-1), each representative S_i of schema S is copied to a Mating Pool $Nf(S_i)/F$ times. Hence,

$$n(S,t+1) = \sum_{i=1}^{n} Nf(S_i)/F . \qquad (9\text{-}4)$$

Consider the average fitness of schema S which is as follows:

$$f(S) = \frac{\sum_{i=1}^{n} f(S_i)}{n} . \qquad (9\text{-}5)$$

From (9-5) we have:

$$\sum_{i=1}^{n} f(S_i) = f(S)n . \qquad (9\text{-}6)$$

By using (9-6) we may now rewrite (9-4) as follows:

$$n(S,t+1) = n(S,t)\, N\, f(S)/F \qquad (9\text{-}7)$$

where $f(S)$ is the average of the fitness functions of schema S; n is the number of nonoverlapping representatives strings; F is the total of all fitness functions taken across all strings (N) in the population.

Consider $f(P) = F/N$ which is the average of the fitness over all string in the population. Then we can write (9-7) as

$$n(S,t+1) = n(S,t)\, f(S)/\, f(P) \qquad (9\text{-}8)$$

This shows that *the number of representatives of a schema grows from generation to generation at a rate that depends upon the ratio of their average fitness to the average fitness of the population. Schemata with large fitness get increasing numbers of representative. Schemata with small fitness decreases.*

If we have a schema, which has fitness greater than the average of a population, for example, as follows

$$f(S) = (1+k)\, f(P) = f(P) + kf(P) , \qquad (9\text{-}10)$$

then

$$n(S,t+1) = n(S,t)\left[f(P) + kf(P)\right]/\, f(P) = n(S,t)(1+k) \qquad (9\text{-}11)$$

After n generations the number of representatives of S will be:

$$n(S,t+n) = n(S,t)(1+k)^{n+1} \qquad (9\text{-}12)$$

Equation (9-12) is the well known Holland's theorem 1.

Holland's theorem 1:

The number of representatives of schemata with above average fitness ($f(S) > f(P)$) grows in the population exponentially, while those schemata which have below average fitness ($f(S) < f(P)$) decrease exponentially.

Crossover

Despite the effectiveness of reproduction in increasing the percentage of "good" representatives, the procedure is essentially sterile. It can not create new and better strings.

Crossover and mutation operators perform this function.

Consider the following example (Table 9-4). Let the population consists of a string A and string B. String A is the representative of schemata S_1 and S_2, and string B represents neither.

Table 9-4. Strings and Schemata

$S_1 = * * 0 * \mid * * * 1$	
$S_2 = * * * * \mid 01 * *$	
$A = 0100 \mid 0101$	(S_1, S_2)
$B = 1010 \mid 1000$	**(no** S_1, **no** S_2**)**

Consider a crossover operator which mates random pairs of strings. It cuts the strings at a randomly determined position, and interchanges all bits to the right of the cut point. Doing so with strings A and B crossover produces the following new strings A' and B':

$A' = 0100 \mid 1000$ (**no** S_1 **no** S_2)

$B' = 1010 \mid 0101$ (S_2)

Discuss the results of crossover operation. Now the string A' does not represent schemata S_1 and S_2, but the string B' represents S_2. This fact shows that *crossover operator can reduce the number of representative of a schema*.

The probability p_{lost} of schema S losing a representative from a randomly chosen cut position (a crossover point), may be expressed as:

$$p_{lost} = p_c \delta(S)/(l-1) ,$$

where p_c is the crossover probability; l is a length of a string.

(Copia modificata per una migliore consultazione on-line)

If we define $p_{retaining}$ as the probability of schema S retaining a representative, then

$$p_{retaining} = 1 - p_{lost} = 1 - p_c \delta(S)/(l-1) \qquad (9\text{-}13)$$

Combining (9-7) and (9-13) we obtain

$$n(S,t+1) \geq [n(S,t) f(S)/ f(P)][1 - p_c \delta(S)/(l-1)] \qquad (9\text{-}14)$$

Equation (9-14) is the following theorem 2 of Holland.

Holland's theorem 2:

The growth of a schema depends upon the value of the fitness functions of its representatives, as well as its defining length.

Thus, a *schema with high fitness function and short defining length is more prolific.*

Mutation

Each bit in a string has a chance of mutation to its opposite value. Let p_m be the probability of mutation in any bit position. Then, the probability that a bit does not change is (1- p_m).

If a string represent a schema that has *o*(S) bits that are either 0 or 1, then the probability that all corresponding *o*(S) bits in the string do not mutate is $(1-p_m)^{o(S)}$. In other words, the probability that a string remains a representation of a schema after mutation is:

$$(1-p_m)^{o(S)} \qquad (9\text{-}15)$$

Since p_m is usually small then formula (9-15) can be simplified as:

$$(1 - o(S) p_m) \qquad (9\text{-}16)$$

We now can write the combined effects of reproduction, crossover and mutation. Combining the formulae (9-14) and (9-16) we have the following schema theorem of Holland.

Schema Theorem: Fundamental Theorem of GA.

$$n(S,t+1) \geq [n(S,t) f(S)/ f(P)][1 - p_c \delta(S)/(l-1)][1 - o(S) p_m] \qquad (9\text{-}17)$$

Schemata grow exponentially if they have high fitness values, short defining length, and low order.

In (9-17) following designations are used:

$n(S,t)$ is the number of representatives of schema *S* at time *t* ;

$f(S)$ is the average fitness function for schema *S*;

$f(P)$ is the average fitness function over the population;

p_c is the crossover probability; l is a length of a string;

$\delta(S)$ is the defining length of schema *S*;

$o(S)$ is the order of *S*;

p_m is the probability of mutation at any bit position.

Final discussions

Coding problem

So far we have dealt with a simple coding scheme using a binary alphabet. How do we know that this binary coding is optimal, or effective? May be another way of coding will be better? Only way to answer on "what coding method is better" through simulation.

Experimental observations shown that GA are robust, that is they produce good results despite the difference in the coding. But, we must give careful attention to the coding method because it can have a significant effect on the accuracy and efficiency of the GA.

Goldberg (1989) [21] proposes two principles for effective coding:

1) The *principle of minimal alphabets*;
2) The *principle of meaningful building blocks*: "a user should select coding so that short defining length and low-order schemata are relevant to the given problem".

Why is needed first principle? A small alphabet produces a long string. In this case the efficiency of "implicit parallelism" property is increases. Goldberg shows that: a binary alphabet yields the maximum number of schemata for a string of a given length, making the binary alphabet the optimal choice of coding in GA.

But, binary coding is not a requirement. Another coding methods are used in GA. For example, real-coded chromosomes consisting of real or integer numbers.

Other Evolutionary Algorithms

The literature contains a variety of evolutionary techniques, some of which show promise in solving problems that are difficult for the simple GA. For example, in the biological reality most high creatures propagate through sexual reproduction. So, *sexual reproduction-based GA* uses two classes of chromosomes: females and males. This representation allows to refine and improve the selection process. But little research has been reported where sexual reproduction is simulated, perhaps because benefits are not clear.

Another example of evolutionary algorithms is a bacterial evolutionary algorithm (BEA) [22].

The BEA incorporates another operations analog to the direct transfer of strands of genes from host cells (a set of chromosomes) to other cells. The operator *of bacterial mutation* is applied to each chromosome one by one by the following way: first chromosome is chosen and it is reproduced in *m* clones. Each

chromosomes is divided on parts: part1, part2,…, part *p*. One part (i) is randomly chosen, and genes within the *i*th part of *m*-1 clones are mutated. At this stage the *i*-th part of all the clones is replaced by the *i*th part of the selected chromosome.
The *gene transfer operation* substitutes the crossover operation in a simple GA. It works with two classes of chromosomes in a population: with better fitness and with bad fitness. It propagates "good" parts from better chromosomes into bad one (see details in [22]).

Lecture N 10 GA Application in Intelligent Control Systems Design

GAs have been used to devise control rules for a variety of dynamic systems. Discuss GA application for tuning membership functions and fuzzy rules of fuzzy controllers. Consider following examples.

Example 1. Cart-centering control problem: centering and stopping a cart located on a one-dimensional track.

In this case we have the following *Task* 1 (Fig.10-1): *design a fuzzy controller which for given initial velocity and position on the track will bring the cart to zero velocity and zero location in minimum time.*

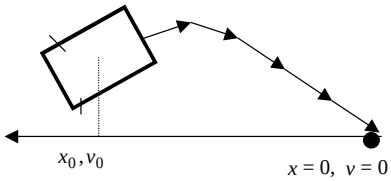


Figure 10-1. Cart centering problem.

The input variables for this problem is the location of the cart - x , and the velocity of the cart - v . The output variable is the force *F* applied to the cart. The equations of motion for the cart are:

$$x(t+\tau) = x(t) + \tau v(t)$$
$$v(t+\tau) = v(t) + \tau F(t)/m ,$$

where τ is the time step, *m* is a mass of the cart.

Example 2. Truck-Backing Control Problem.

In this case (Fig.10-2) we have the following *Task* 2:
design a fuzzy controller which is used to control a truck from an arbitrary initial position to the special final position (called "loading dock") in minimum time.
We will describe a truck location as a point on 100 $\times$ 100 grid at a given angle to the horizontal, ϕ .

The goal is to control the truck so that in a minimum time it goes from the initial location (x_0, y_0, ϕ_0) to the location of "loading dock" (x = 50 m, *y* = 100 m , ϕ =90).

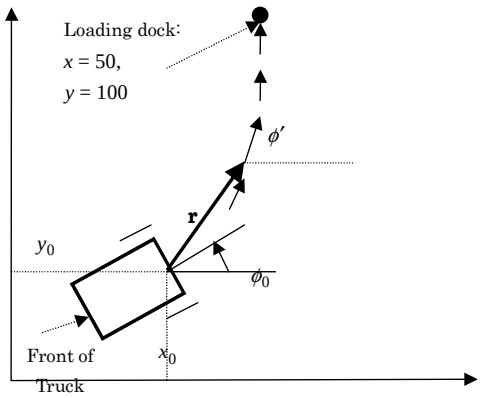


Figure 10-2. Truck-Backing Control problem.

The controller must provide a turning angle θ that moves the wheels and turn a truck every time step.
For the truck motion we have the following relationships:

$$\phi' = \phi + \theta ,\ x' = x + r\cos\phi' ,\ y' = y + r\sin\phi' ,$$

where x', y', ϕ' are the new values of the truck position and angle after each time step.

Consider the application of a simple genetic algorithm (SGA) method for these tasks.
At first let us remember what is needed to design a fuzzy controller?
The following steps are needed:
- description of input-output parameters and its values spaces (Table 10-1 for the task 1 and Table 10-2 for the task 2);
- division input-output spaces into different partition sizes ;
- description of these partitions by membership functions (MF) (Fig.10- 3);
- construction of a set of fuzzy rules (FR).

So, different fuzzy controllers have different MFs and FRs, and we want to find optimal parameters of MF's and optimal set of FR's so that a given fuzzy controller performs the control task in a minimum time.

(Copia modificata per una migliore consultazione on-line)

Table 10-1. Cart Control problem

variable	value
m	$20kg$
τ	$0.02\,\text{sec}$
x	$-2\ \text{to}+2m$
v	$-2\ \text{to}+2m/s$
F	$-150\ \text{to}+150N$

Table 10-2. Truck Control problem

variable	value
r	$2m$
θ	$-30^\circ\ \text{to}+30^\circ$
ϕ	$-90^\circ\ \text{to}+270^\circ$
x	$0\ \text{to}\,100\text{m}$
y	$0\ \text{to}\,100m$

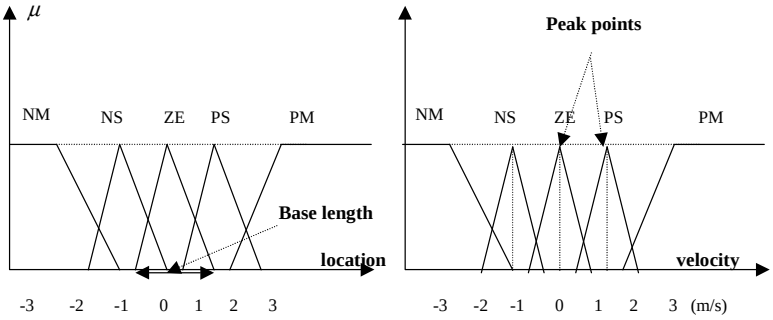


Figure 10-3. Membership functions.

Let us consider GA application for tuning parameters of the cart fuzzy controller (Task 1).

In this task five fuzzy sets with triangular MFs are used to partition of the input and output spaces (Fig.10-3) : NM - Negative Medium, NS - Negative Small, ZE - Zero,
PS - Positive Small, PM - Positive Medium.

For a fuzzy control of the cart we will consider the following type of fuzzy rules:

IF X is { NM,NS,ZE,PS,or PM} and V is { NM,NS,ZE,PS,or PM}) THEN { *output*}.

The task of GA is to find optimal set of these fuzzy rules.

The first step in GA application is a choice of a coding method of potential solutions.

Coding for the optimization problem of the cart controller design

We must create a string representing our potential solutions. In this case, what is our potential solution? It is a set of fuzzy rules. For fuzzy rules coding we will use an integer-based coding method by the following way. Let us introduce the integer numbers for the five fuzzy sets as:
1 - NM, 2 - NS, 3 - ZE, 4 - PS, 5 - PM. Then the set of fuzzy rules can be described by the look-up-table shown in the Fig.10-4.

position

velocity	*NM*	*NS*	*ZE*	*PS*	*PM*
NM	1	4	3	2	1
NS	5	2	4	3	2
ZE	1	2	4	5	1
PS	4	3	1	2	2
PM	1	1	3	4	5

Figure 10-4. Look-up table for the Cart Fuzzy Controller

We can describe this set of rules by the following string (consisting of 25 positions):

{1432152432124514312211345}.

So, we will use such a kind of chromosomes in our GA-based controller design.
The task of GA is to find an optimal solution, that is, to generate the population containing optimal fuzzy rules for our control task.

Let us remember the definition of our task: we must construct a fuzzy controller which for given initial position and velocity on the track will bring the cart to zero velocity and zero location in a minimum time.

If the coding method and a fitness function (see below) are defined, then SGA software can be used in order to find optimal solution. The SGA program allows the user to define the values of population size; of maximum number of generations, of probabilities of crossover and mutation. For our Task 1 the following values are chosen:
100, 100, 0.7, 0.03.

In order to select the individuals for the next generation, a special selection method called “tournament selection” is used. In the “tournament selection”, two or more individuals of population are selected at random and their fitness compared. The individuals with the highest fitness are copied to the next generation.

For the search of optimal solution of our task 1 the GA process is divided into 2 stages : *stage1* (called *evolution stage*) and *stage2* (called *refinement stage*). In the first stage GA is used to find satisfactory solutions (controllers) which are solved the task 1 with a tolerant error at some time (not necessary minimal time). In the second stage, GA attempted to minimize the amount of time needed to bring the cart’s location and velocity to zero.

Stage 1.
In this stage GA performed 30 generations. The fitness function rewarded an individuals of the population according to how well it come to the tolerance value equal $\pm$ 0.5 for both location and velocity. The tolerance value shows the admissible value of error (a difference between real final location and velocity of cart and desired final location and velocity equal zero).
The number of generations equal 30, at which the evolution stage ended, was obtained experimentally.

(Copia modificata per una migliore consultazione on-line)

A main problem in GA application is the definition of an appropriate fitness function for the given problem. In the previous lecture we discussed the properties of GA, and pointed that GA do not require a mathematical model. However, from another side, GA must have a method to evaluate the controller's performance. In this case the mathematical model can be used to evaluate the fitness of a given individual of a population.
Consider the fitness function for this stage.

Fitness Function for Evolution Stage.

(*if* ($|x|$ < 0.5) and ($|v|$ < 0.5) *then* fitness = $8 \cdot 175 / time$

{ x and v are within tolerance values }

else (*if* ($time$ = 175) *then*

{ simulation times out before x and v are within tolerance values }

(*if* ($|x|$ < 1.0) and ($|v|$ < 1.0) *then* fitness = $3.5 / \sqrt{x^2 + y^2}$ *else*

fitness = -1)))

if (($|x|$ > 5.0) and ($|v|$ > 5.0) *then* fitness = -7).

Let us comment this fitness function.

If the controller succeeded in bringing and it is within the tolerance, then it is evaluated by the fitness relative to the time it took. The minimum fitness in this case is 8 (since 175 is the time limit).

If the controller "timed out" (that is did not converge by 175 time steps) , it was slightly punished with negative fitness, or slightly rewarded depending on location and velocity.

If the controller has the value of location or velocity greater than 5.0, the fitness was given a larger negative value.

Stage 2.

The second stage, from generation 31 to generation 100, finds the solution based on a time criteria.
The fitness function is as follows:

Fitness Function for Refinement Stage.

(*if* ($|x|$ < 0.5) and ($|v|$ < 0.5) *then* fitness = $0.3 \cdot (175 - time)$

{ x and v are within tolerance values }

else

(*if* ($time$ = 175) *then*

{ simulation times out before x and v are within tolerance values }

then fitness = $-42\sqrt{x^2 + y^2}$))

if (($|x|$ > 5.0) and ($|v|$ > 5.0) *then* fitness = -300).

Comment.

If the controller reached the tolerance values, it was rewarded according to how short a time it took. If the controller "timed out", it was punished according to the error values. If the controller has the value of location or velocity greater than 5.0, the fitness was given a larger negative value.
Following pictures illustrate a fitness evaluation process of each individual (chromosome) in a current population.

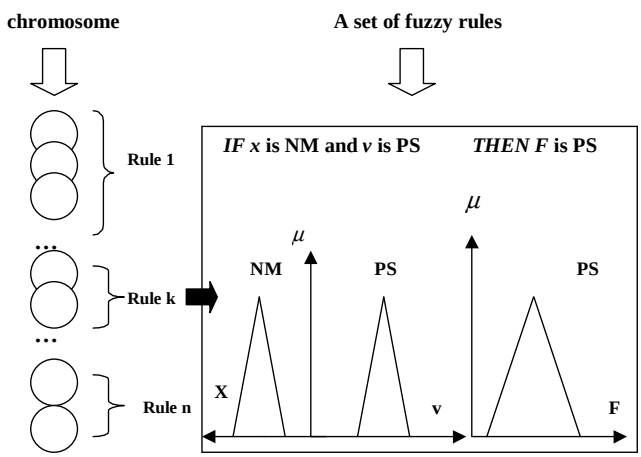


Figure 10-5. Decoding structure of a chromosome for GA-based optimization of a fuzzy controller .

(Copia modificata per una migliore consultazione on-line)

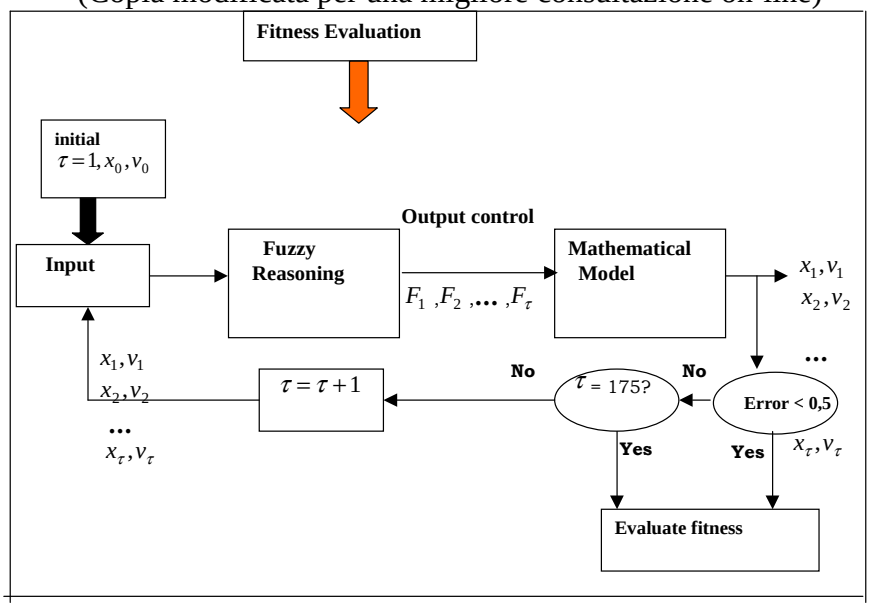


Figure 10-6. Fitness evaluation process in GA-based optimization of a fuzzy controller.

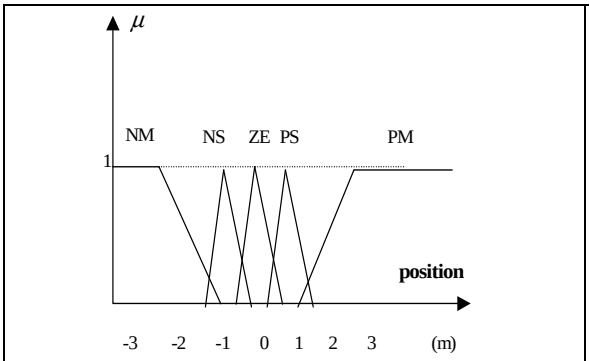
Consider now some problems connected to initial conditions.
A potential optimal controller must be able to operate over the entire range of initial conditions. How must GA work in this case? This can be done by using multiple initial conditions in the evaluation of each individual of a population.

If a single initial conditions were used, for example x_0 = 0.7m and v_0 = - 0.5m/s, then GA would find a controller which works well around that particular point, but may fail elsewhere. In order to avoid this disadvantage, in evaluating of each individual of population, the total fitness of the individual is used. The total fitness is calculated as a sum of the fitness at each initial condition. Different initial conditions are shown in the Table 10-3.

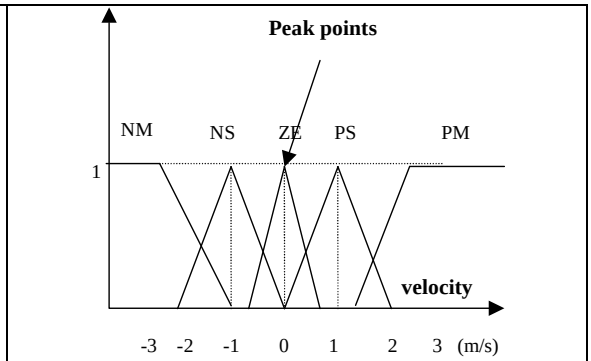
Table 10-3. Initial conditions for cart controller.

Controller	$Initial-conditions(x_0, v_0)$
333	(−2,−2)(−2,2)(0,0)(2,−2)(2,2)
555	(−2,−2)(−2,2)(−1,−1)(−1,1)(0,0)(1,−1)(1,1)(2,−2)(2,2)
557	(−2,−2)(−2,2)(−1,−1)(−1,1)(0,0)(1,−1)(1,1)(2,−2)(2,2)

Remark 1. Controller 333 (555,557) means that 3 (5 and 5, respectively) membership functions are used for the location input variable and 3 (5 and 5, respectively) for the velocity description, and 3(5 and 7, respectively) membership functions are used for output description.
In Fig.10-7 the result of GA optimization for the Task 1 is shown.



(a)

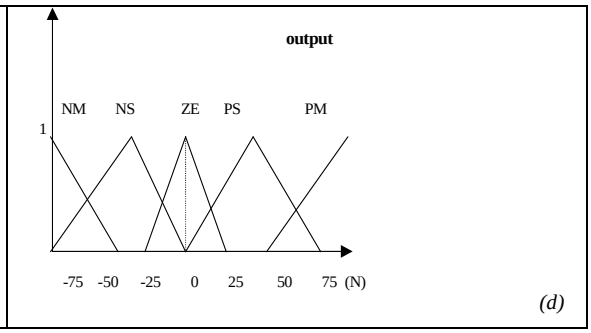


(b)

position

velocity	*NM*	*NS*	*ZE*	*PS*	*PM*
NM	5	4	4	4	2
NS	5	5	5	1	1
ZE	5	5	3	1	1
PS	5	5	1	1	1
PM	4	2	2	2	1

(c)



(d)

Figure 10-7. The optimal fuzzy cart controller.

Remark *2. To simplify understanding of our example we considered only the optimization problem for the set of fuzzy rules. If we want to consider the optimization problem for membership function's parameters too, we must introduce this information in chromosomes. Two input parameters (position, velocity) and one output parameter are described by 15 MF's (five fuzzy sets NM,NS,ZE,PS,PM - for* x*, five fuzzy sets for* v*, and five for output). Triangular MFs can be described by its base length values (or their end points) (see Fig.10-3). We will find by SGA optimal values of end points for each MF.*

(Copia modificata per una migliore consultazione on-line)

Let the end point's value is described by m *bits, then 15* m *bits are needed in order to encode all membership functions for input /output parameters. So, to represent a set of FRs and end points of input MFs a binary string of (25*$\times 3$ *+15* m *) bits is needed.*

Remark *3. 1) The fitness functions considered for the Tasks 1 and 2 are obtained experimentally. In Fig.10-7 results (a), (b),(c) are obtained by GA; part (d), output MF's, was fixed.*

Consider now Task 2, a Truck-Backing Problem.

Truck-Backing Problem

Let us state this problem as follows: by using GA design a fuzzy controller which is used to control a truck from an arbitrary initial position to the final position (x = 50 m, y = 100 m , ϕ =90) in a minimum time.

For this task also two stage are used. In the first stage working controllers are found, in the second stage the optimal controller is determined based on time minimization. The fitness function for this task is defined by the following way:

Fitness Function for Evolution Stage:

(*if* $(|x_{err}| < |x_{tol}|)$, $(|y_{err}| < |y_{tol}|)$ and $(|\phi_{err}| < |\phi_{tol}|)$ *then*

fitness = 2.0 *else* fitness = - 5) .

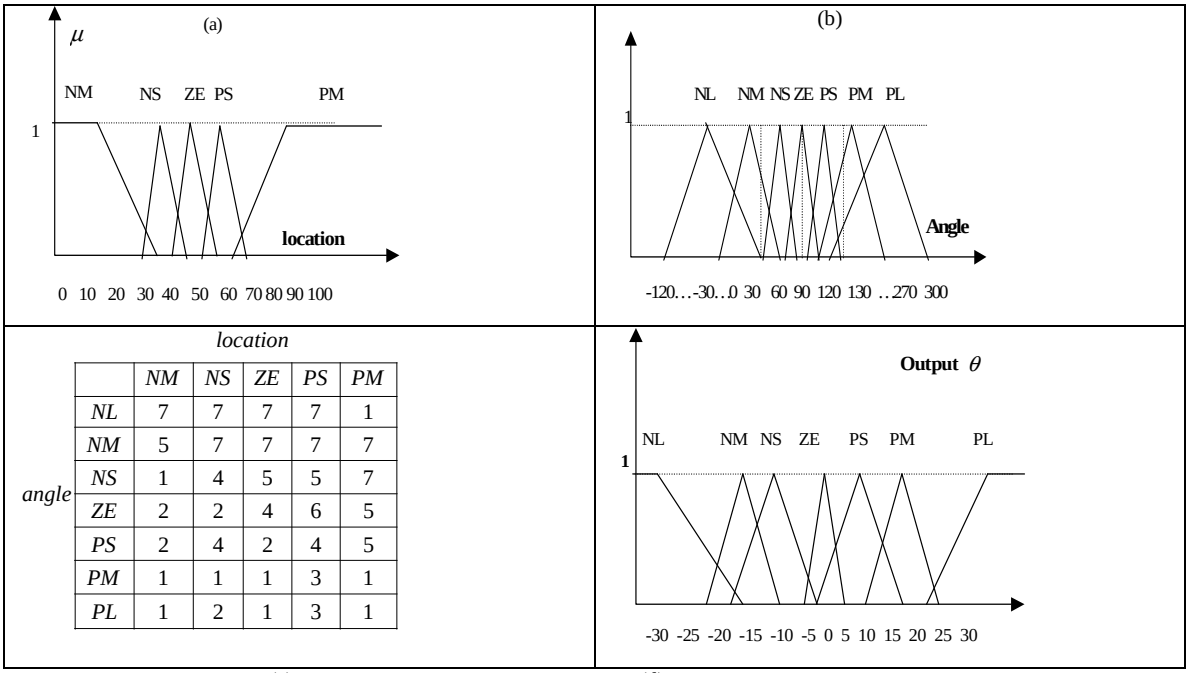
Fitness Function for Refinement Stage:

(*if* $(|x_{err}| < |x_{tol}|)$, $(|y_{err}| < |y_{tol}|)$ and $(|\phi_{err}| < |\phi_{tol}|)$ *then*

fitness = (100 - *time*) *else* fitness = - 50) ,

where x_{err} , y_{err} , ϕ_{err} are the difference between the real final location and angle the truck and the desired final location and angle x_{tol} , y_{tol} , ϕ_{tol} are the tolerance values.

In Fig.10-8 (a,b,c,d) the result of GA-based optimization for the Task 2 is shown.



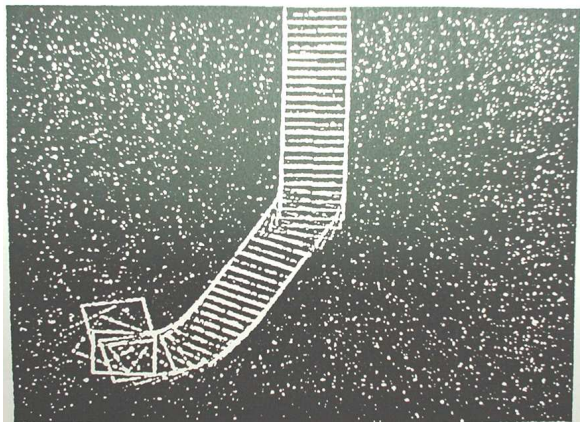
location

angle	NM	NS	ZE	PS	PM
NL	7	7	7	7	1
NM	5	7	7	7	7
NS	1	4	5	5	7
ZE	2	2	4	6	5
PS	2	4	2	4	5
PM	1	1	1	3	1
PL	1	2	1	3	1

(c) (d)

Figure 10-8. 757 fuzzy truck controller.

In Fig.10-9 simulation results of the truck motion under fuzzy control is shown.



Motion of truck from x_0 = 20 ϕ_0 = -80 degrees

Figure 10-9. Simulation of the truck motion

(Copia modificata per una migliore consultazione on-line)

In general, a classical PID-control system can be described as shown in Fig.10-10.

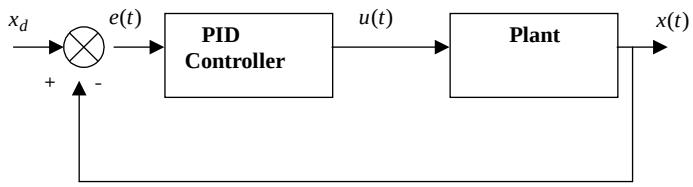


Figure 10-10 . Block diagram of a classical PID control system

The input-output relation of the PID controller is expressed as

$$u = K_p e + K_i \int_o^t e dt + K_d \dot{e},$$

where u is the control signal, e is the error signal, $\dot{e}$ is the derivative of error, and K_p, K_i, K_d denote proportional gain, the integral gain and the derivative gain, respectively.

If different values of K_p, K_i, K_d are chosen, then it is obvious that various responses of the plant will be obtained. Therefore, the <u>parameters tuning problem</u> of a PID controller can be considered as *selecting the three parameters* K_p, K_i, K_d *such that the response of the plant will be desired.* We will use GA for this task. The block diagram of GA-based tuning is shown in Fig. 10-11.

Input variables for fuzzy reasoning are the error and derivative of error, and output variables of fuzzy reasoning are the PID parameters. If there are n_1 and n_2 membership functions for the error and its derivative, then there will be $n_1 \times n_2$ fuzzy rules expressed as:

If $e \text{ is } A_1$ and $\dot{e} \text{ is } B_1$, then $K_p = C_{11}, K_i = D_{11}, K_d = E_{11}$

$\vdots$

If $e \text{ is } A_1$ and $\dot{e} \text{ is } B_{n_2}$, then $K_p = C_{1n_2}, K_i = D_{1n_2}, K_d = E_{1n_2}$

If $e \text{ is } A_2$ and $\dot{e} \text{ is } B_1$, then $K_p = C_{21}, K_i = D_{21}, K_d = E_{21}$

$\vdots$ (10-1)

If $e \text{ is } A_2$ and $\dot{e} \text{ is } B_{n_2}$, then $K_p = C_{2n_2}, K_i = D_{2n_2}, K_d = E_{2n_2}$

$\vdots$

If $e \text{ is } A_{n_1}$ and $\dot{e} \text{ is } B_{n_2}$, then $K_p = C_{n_1 n_2}, K_i = D_{n_1 n_2}, K_d = E_{n_1 n_2}$,

where $A_1, A_2, ..., A_{n_1}$ and $B_1, B_2, ..., B_{n_2}$ are membership functions of e and $\dot{e}$, and

$C_{11}, ..., C_{n_1 n_2}$, $D_{11}, ..., D_{n_1 n_2}$ and $E_{11}, ..., E_{n_1 n_2}$ are real numbers that satisfy

$$K_{p,\min} \le C_{ij} \le K_{p,\max}, 1 \le i \le n_1, 1 \le j \le n_2$$

$$K_{i,\min} \le D_{ij} \le K_{i,\max}, 1 \le i \le n_1, 1 \le j \le n_2 \qquad (10\text{-}2)$$

$$K_{d,\min} \le E_{ij} \le K_{d,\max}, 1 \le i \le n_1, 1 \le j \le n_2 .$$

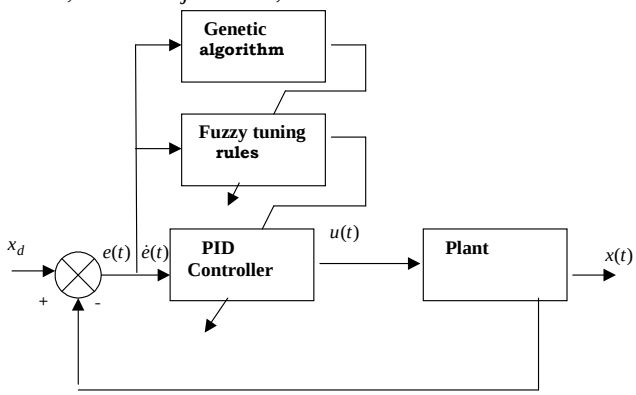


Figure 10-11. Block-diagram of GA-based tuning of a fuzzy-PID controller.

According to Sugeno fuzzy inference model and fuzzy rules (10-1) , the outputs will be

(Copia modificata per una migliore consultazione on-line)

$$K_p = \frac{\left(\sum_{i=1}^{n_1}\sum_{j=1}^{n_2}\omega_{ij}C_{ij}\right)}{\left(\sum_{i=1}^{n_1}\sum_{j=1}^{n_2}\omega_{ij}\right)}, \quad K_i = \frac{\left(\sum_{i=1}^{n_1}\sum_{j=1}^{n_2}\omega_{ij}D_{ij}\right)}{\left(\sum_{i=1}^{n_1}\sum_{j=1}^{n_2}\omega_{ij}\right)}, \quad K_d = \frac{\left(\sum_{i=1}^{n_1}\sum_{j=1}^{n_2}\omega_{ij}E_{ij}\right)}{\left(\sum_{i=1}^{n_1}\sum_{j=1}^{n_2}\omega_{ij}\right)} \qquad (10\text{-}3)$$

where $\omega_{ij} = A_i(e)\cdot B_j(\dot{e})$.

We have the following optimization task :

By using GA define optimal values of K_p, K_i, K_d that is define optimal values of C_{ij}, D_{ij}, E_{ij} corresponding to each rule in (10-1).

In order to apply GA we must also define fitness function for our task. We introduce the following the fitness as the reciprocal of the squared error:

$$\text{Fitness} = \frac{1}{\int_0^t e^2 dt} \qquad (10\text{-}4)$$

In this case larger values of the fitness will correspond to a lower total error and , hence, a better performance of PID.
Since GA works with a coding of parameters, let us describe a coding method.

Without any loss of generality, we can assume that there are N_1, N_2, N_3 bits for each value of C_{ij}, D_{ij}, E_{ij} respectively. In this case, the size of chromosome S representing set of fuzzy rules (10-1) is defined as $(n_1 \times n_2)\times(N_1 + N_2 + N_3)$ bits.

Let us express S as $S = S_1 S_2 \ldots S_{n1\times n2}$, where $S_j = S_j^1 S_j^2 S_j^3$, $j = 1,2,\ldots,(n_1 \times n_2)$ are strings of $(N_1 + N_2 + N_3)$ bits.

For example, if

$$S_1^1 = \underbrace{00\ldots001}_{N_1 bits},\ S_1^2 = \underbrace{00\ldots010}_{N_2 bits},\ \text{and}\ S_1^3 = \underbrace{00\ldots011}_{N_3 bits},$$

These binary strings represent decimal numbers 1,2,3, respectively. Then the values of $C_{11}, D_{11} E_{11}$ (see Appendix) that correspond to the first fuzzy rule in (10-1) will be:

$$C_{11} = \frac{K_{p,\max} - K_{p,\min}}{2^{N_1} - 1}\cdot 1 + K_{p,\min}$$

$$D_{11} = \frac{K_{i,\max} - K_{i,\min}}{2^{N_2} - 1}\cdot 2 + K_{1,\min}$$

$$E_{11} = \frac{K_{d,\max} - K_{d,\min}}{2^{N_3} - 1}\cdot 3 + K_{d,\min}.$$

When the coding method, the fitness function and important genetic parameters such as the population size, the crossover and mutation rates, and the number of generations are defined, then we may use GA-based searching procedure.
So, the PID parameters tuning procedure based on GA can be summarized as follows:

1. Give the membership functions of e and $\dot{e}$.
2. Define the fitness measure as shown in Eq.(10-4).
3. Determine the population size, number of generations, crossover and mutation rates.
4. Produce an initial generation of binary strings in a random way.
5. For each string in the generation decode the binary parts of string into the corresponding values of C_{ij}, D_{ij}, E_{ij}, respectively, for all ($i = 1,2,\ldots,(n_1 \times n_2)$, $j = 1,2,\ldots,(n_1 \times n_2)$.
6. Evaluate the fitness of the decoded values of C_{ij}, D_{ij}, E_{ij}.
7. Reproduce an intermediate generation (mating pool) by the roulette wheel selection.
8. Perform crossover and mutation in the mating pool.
9. Define a new generation from old members and offspring.
10. Repeat steps 5-10 iteratively until the number of generations reaches a prescribed value.

Example 3: Simulation example: a ball and beam system.
To illustrate the proposed above tuning method consider a ball and beam system shown in Fig. 10-12.

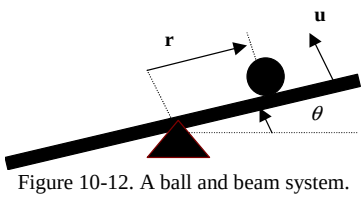


Figure 10-12. A ball and beam system.

We assume that there are five membership functions for e and $\dot{e}$, and the values of N_1, N_2, N_3 are all chosen to be 6. We also assume that membership functions for e and $\dot{e}$ are represented as shown in Fig.10-13.

(Copia modificata per una migliore consultazione on-line)

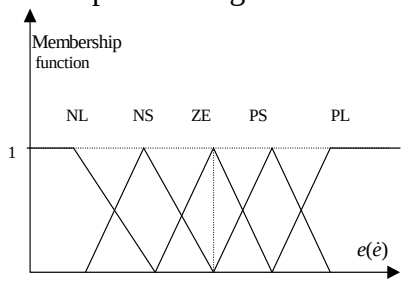


Figure 10-13. Membership functions for e and $\dot{e}$.

In applying the mentioned above GA the population size, the crossover rate, the mutation rate and generation number are chosen to be 100, 0.98, 0.01 and 50, respectively.
For the ball and beam system shown above, the mathematical model is given by the following equation:

$$0 = \left(\frac{J_b}{R^2} + M\right)\ddot{r} + MG\sin\theta - Mr\dot{\theta}^2$$

$$\tau = (Mr^2 + J + J_b)\ddot{\theta} + 2Mr\dot{r}\dot{\theta} + MGr\cos\theta,$$

where τ is the torque applied to the beam, θ, J are the angle and the moment of inertial of the beam, M, J_b, R, r are the mass, the moment of inertial, the radius and the position of the ball, respectively, and G is the acceleration of gravity.

If $r, \dot{r}, \theta, \dot{\theta}$ are chosen as state variables, then by defining

$$B = \frac{M}{\left(\frac{J_b}{R^2} + M\right)}, \quad C = -(Mr^2 + J + J_b)$$

the state equations of the system will be expressed as

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} x_2 \\ B(x_1 x_4^2 - G\sin x_3) \\ x_4 \\ (2Mx_1x_2x_3 + MGx_1\cos x_3)/C \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ -1/C \end{bmatrix}\tau, \qquad (10\text{-}5)$$

where $[x_1 x_2 x_3 x_4] = [r\ \dot{r}\ \theta\ \dot{\theta}]$.

Based on Eq.(10-5), the control problem in this example is to use a PID controller to generate the desired torque such that the ball and beam system can be driven from initial state $[r\ \dot{r}\ \theta\ \dot{\theta}] = [0.3m\ \ 0\ \ -0.6rad\ \ 0]$ to the zero final state.

For the simulation process, the values of the parameters are chosen as follows:

$$J = 0.036\,kg\cdot m^2, M = 0.5\,kg, J_b = 2\times10^{-5}\,kg\cdot m^2, R = 0.01m, G = 9.8m/s^2.$$

Since both the state variables r and θ are to be driven to zero, the input-output relation of the PID controller will be given as (here $\tau = u$)

$$u = K_{p_1}e_1 + K_{i_1}\int_o^t e_1 dt + K_{d_1}\dot{e}_1 + K_{p_2}e_2 + K_{i_2}\int_o^t e_2 dt + K_{d_2}\dot{e}_2,$$

where $e_1 = r_d - r, \quad e_2 = \theta_d - \theta$.

Then by assuming that $K_{p_1}, K_{i_1}, K_{d_1}$ are determined by $e_1, \dot{e}_1$ and $K_{p_2}, K_{i_2}, K_{d_2}$ are determined by $e_2, \dot{e}_2$, respectively, the fuzzy tuning rules of the PID controller for the ball and beam system are generated on the base of GA (described above). In Table 10-4 (a,b) results of GA-based fuzzy rules determination are shown.

Table 9-4. Fuzzy tuning rules for parameters of the PID controller.

(a)

If e_1	*and* $\dot{e}_1$	*then* K_{P_1}	K_{I_1}	K_{D_1}
NL	*NL*	30.0000	0.0012	5.2500
NL	*NS*	64.0000	0.0071	7.6000
⋮	⋮	⋮	⋮	⋮
NS	*ZE*	4.0000	0.0006	0.3375
ZE	*ZE*	55.0000	0.0019	10.4844
ZE	*PS*	9.0000	0.0180	0.1688
PS	*PL*	32.0000	0.0064	5.800
⋮	⋮	⋮	⋮	⋮
PL	*PL*	16.0000	0.0007	3.0500…

(b)

If e_2	*and* $\dot{e}_2$	*then* K_{P_2}	K_{I_2}	K_{D2}
NL	*NL*	40.0000	0.0019	0.4375
NL	*NS*	15.0000	0.0014	0.1172
⋮	⋮	⋮	⋮	⋮
NS	*ZE*	38.0000	0.0033	0.5344
ZE	*ZE*	14.0000	0.0007	0.8531
ZE	*PS*	24.0000	0.0008	2.2500
PS	*PL*	26.0000	0.0260	1.4625
⋮	⋮	⋮	⋮	⋮
PL	*PL*	17.0000	0.0007	1.1688…

Simulation of the ball-beam system motion under PID control (with optimal gains founded by GA)) is shown in Fig.10-14.

(Copia modificata per una migliore consultazione on-line)

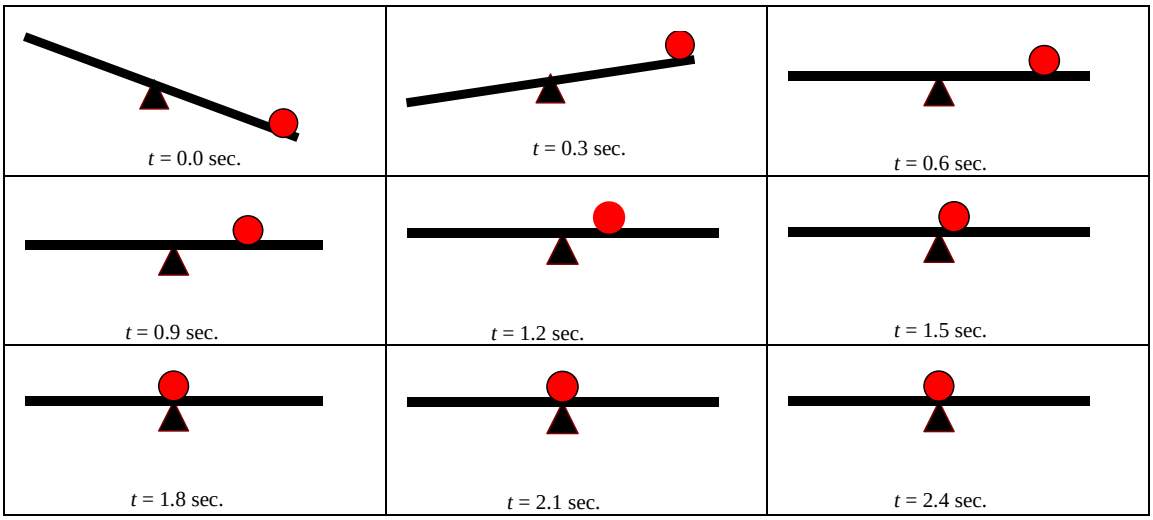


Figure 10-14. Simulation of ball-beam system movement.

We studied main ideas of GA methodology and its application and considered its effectiveness to solve optimization problems. *This methodology allows to design self-constructing adaptive fuzzy controllers for different application areas.*

Fuzzy Inference System Tuning Method Based on GA

In this lecture we discuss the GA-based method to automatically and simultaneously select appropriate rule base and membership functions of a fuzzy system structure [23]. Consider the following task definition.

Task definition:
Develop the GA capable to tune a Mamdani type fuzzy inference-based control system. Tuning is based on the training patterns represented as a table of in-out pairs.

Fuzzy system structure
Consider the following fuzzy sets used usually in a fuzzy rule base of a fuzzy system: “Negative-Large”, “Negative-Medium”, “Negative-Small”, “Zero”, ”Positive-Small”, ”Positive-Medium”, ”Positive-Large”, and so on.
Let us introduce special notations for the designation of these sets.
We will use the index set $I_{m_i} = \{-m_i, -m_i+1, \ldots, -1, 0, 1, \ldots, m_i-1, m_i\}$, $m_i \in N$ (set of natural numbers), to describe ($2m_i+1$) linguistic values for a linguistic variable x_i. For example, let we have $F_j(x_i)$ and $j \in I_3$. $F_j(x_i)$ defines the j-th fuzzy set for the x_i-th linguistic variable. Thus, there are the following fuzzy sets for the x_i-th linguistic variable:

$$F_{-3}, F_{-2}, F_{-1}, F_0, F_1, F_2, F_3,$$

which may be called as “Negative-Large”, “Negative-Medium”, “Negative-Small”, “Zero”, ”Positive-Small”, ”Positive-Medium”, ”Positive-Large”.
Mamdani type Fuzzy Inference System consists of a fuzzy rules base of the following type:

IF x_1 is $F_{j_1}(x_1)$ $AND \cdots AND$ x_n is $F_{j_n}(x_n)$ THEN y is O_k, $j_i \in I_{m_i}$, $-m \le k \le m$,

where x_i and y stand for input and output linguistic variables, respectively. $F_{j_i}(x_i)$ and O_k are fuzzy sets characterized by membership functions.

Let input fuzzy variables are described by Gaussian membership functions:

$$\mu F_{j_i}(x_i) = 1 \cdot e^{-\frac{(x_i - j_i D_i)^2}{s_i^2 D_i^2}}, \quad j_i \in \{-m_i, \ldots, m_i\},$$

and output fuzzy variables are described as follows:

$$O_k(y) = \begin{cases} 1, & if \quad y = kD \\ 0, & if \quad y \ne kD \end{cases}, \quad k \in \{-m, .., m\}.$$

Consider central values (*Cv*) of $\mu F_{j_i}(x_i)$ and O_k as follows (see Fig.10-15):

$Cv(\mu F_{j_i}(x)) = j_i D_i$ and $Cv(O_k) = kD$, $k \in \{-m, .., m\}$.

(Copia modificata per una migliore consultazione on-line)

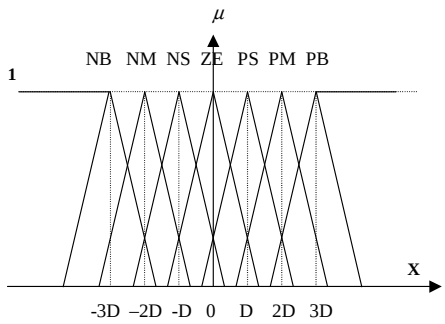


Figure 10-15. The membership functions of a fuzzy rule.

So, the membership functions are defined by the parameters m_i, m, D_i, s_i, s, D.
Let us write a general form of a fuzzy rule by the following way:

Rule *l* (R^l):

$$\textit{If AND}\ (x_i \text{ is } \mu^l_{F_{j_i}}(x_i))\ \textit{Then}\ (y^l \text{ is } \mu^l{}_y),\ j_i \in I_{m_i}, l \in \{1,...,M\}.$$

According the Lemma 1 (see Lecture 6), we can calculate the total output of given fuzzy system as:

$$y = f(x_1,...,x_n) = \frac{\sum_{l=1}^{M} \bar{y}^l \prod_{i=1}^{n} \mu^l{}_{F_{j_i}}(x_i)}{\sum_{l=1}^{M} \prod_{i=1}^{n} \mu^l{}_{F_{j_i}}(x_i)} = \frac{\sum_{l=1}^{M} \bar{y}^l z^l}{\sum_{l=1}^{M} z^l}, \text{where } z^l = \prod_{i=1}^{n} \mu^l{}_{F_{j_i}}(x_i) \qquad (10\text{-}6)$$

and $\bar{y}^l$ is the point of maximum value (called here as a central value) of $\mu_y{}^l(y)$.

Consider now the following task:
By using GA and training input-output patterns (*p*) update numbers of membership functions for input-output linguistic variables, membership functions parameters, and the set of fuzzy rules so that the total error measure E is minimal.

Define E^p as: $E^p = 1/2(f(x^p) - d^p)^2$ and $E = \sum_p E^p$, where x^p and d^p are input-output pattern.

We will consider a few stages of GA based (evolutional) tuning.
First stage: evolutional updating of fuzzy rules and membership function's parameters, while the number of membership functions is fixed.
Second stage: minimization of fuzzy rules.
Third stage: evolutional updating of numbers of membership functions for input-output linguistic variables and membership function's parameters.
In order to apply GA to perform tasks mentioned above, we must define our searching space. The searching space consists of a set of parameters

$$S = (m_1,...,m_n, m, D_1,...,D_n, s_1,...,s_n, s, D, R^1,..., R^M)$$

and we will select an appropriate S .
So, first we must code S as a finite-length string (chromosome) and then search for a better solution in the searching space by using chosen fitness function .The tuning ranges of given parameters are defined according to given application area.

Evolutional updating of fuzzy rules

The set of fuzzy rules described above can be considered as a mapping:

$$f : \{-m_1, -m_1+1,...,-1,0,1,...,m_1-1,m_1\} \times \cdots \times$$
$$\{-m_n, -m_n+1,...,-1,0,1,...,m_n-1,m_n\} \to \{-m\ ,-m\ +1,...,-1,0,1,...,m\ -1,m\ \}$$

So, you can see that in order to approximate some unknown function *f* by the set of fuzzy rules we can approximate the set of rules by an index function $f(j_1,..., j_n)$.

We may consider $f(j_1,..., j_n)$ as $f(j_1,..., j_n) = \langle a_1 j_1 + a_2 j_2 + ... + a_n j_n \rangle = \langle b \rangle$, where $\langle b \rangle$ denotes the integer nearest to the value b.

$$\langle b \rangle = \begin{cases} -m & \textit{if } b \le -m + 1/2, \\ \text{int}(b - 1/2) & \textit{if } -m + 1/2 < b \le m - 1/2, \\ \text{int}(b - 1/2) + sign(b - 1/2) & \textit{if } 0 < b \le m - 1/2, \\ m & \textit{if } \ b > m - 1/2 \end{cases}$$

There are $(2m+1)$ fuzzy sets O_k for the output variable. So, combining the same consequent fuzzy set, we can represent the complete control rules by the following $(2m+1)$ fuzzy control rules:

(Copia modificata per una migliore consultazione on-line)

Rule Ξ_k :

IF OR {[x_1 is $F_{j_1}(x_1)$] $AND \cdots AND$ [x_n is $F_{j_n}(x_n)$] $\left| \langle a_1 j_1 + a_2 j_2 + ... + a_n j_n \rangle = k \right.$ }

THEN y is O_k , $j_i \in I_{m_i}$, $-m \le k \le m$. (10-7)

We will interpret the OR -operator as Lukasievich OR operator (called as *LOR*) and the *AND*-operator by the following way:

$LOR(p_1,...,p_n) = \min(\sum_{i=1}^{n} p_i, 1), \forall p_i \in [0,1]$	$AND(p_1, p_2,..., p_n) = \prod_{i=1}^{n} p_i$

Now we modify our formula (10-6) into the following:

$$y = \frac{\sum_{k=-m}^{m} \Delta_k cv(O_k)}{\sum_{k=-m}^{m} \Delta_k} \text{ , where } \Delta_k = \underbrace{LOR}_{\substack{\langle a_1 j_1 + ... + a_n j_n \rangle = k \\ j_i \in I_{m_i}}} \left\{ \prod_{i=1}^{i=n} \mu^l_{F_{j_i}}(x_i) \right\}$$

and $cv(O_k)$ are the central values of fuzzy sets O_k .

Observing the fuzzy rules (10-7), we find that the fuzzy rules base is defined by the parameters $\langle a_1, a_2,..., a_n \rangle$, and the membership functions are defined by the parameters $(m_1,..., m_n, m, D_1,..., D_n, s_1,..., s_n, s, D)$.

So, the control performance depends mainly on the selected $\langle a_1, a_2,..., a_n \rangle$ and $(m_1,..., m_n, m, D_1,..., D_n, s_1,..., s_n, s, D)$. Therefore, GA may be used in order to find optimal parameters.

Remark. In practical applications, where there are large amount of input-output variables and large amount of membership functions, the number of fuzzy rules is very big. In general case it equals to $(2m_1 + 1) \times ... \times (2m_n + 1)$ rules. So, in order to find appropriate fuzzy rules base by GA, we must consider chromosomes of $N(2m_1 + 1) \times ... \times (2m_n + 1)$ length that results in extremely large time of GA work. In considered method the fuzzy rules base is defined by the parameters $\langle a_1, a_2,..., a_n \rangle$, hence a chromosome length is proportional to n .

Appendix

Coding and decoding of real –valued parameters

Consider a vector of parameters $\vec{x} = \{x_i\}, \ i = 1,..., n$, where each $x_i \in [u_i, v_i] \subset R, u_i < v_i$.

We will code this vector by a bit string $\vec{a} = (a_1,..., a_l) \in IB^l$ such that $l = nl_x$. So, each segment $a_{i1},..., a_{il_x} \in IB^{l_x}$ of $\vec{a}$ encodes the corresponding variable x_i . This process is illustrated in Fig.10-A1.

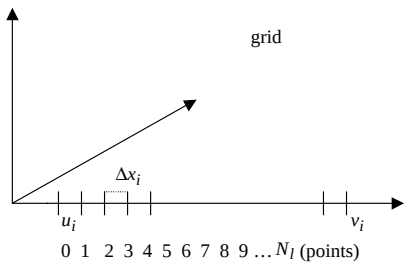


Figure 10-A1. Coding of variable x_i .

Decoding of a segment works by decoding the bits into corresponding integer value between 0 and $2^{l_x} - 1$ and then linearly mapping that integer to the real interval $[u_i, v_i]$.

Simple decoding of binary (base two) numbers to decimal (base ten) numbers takes the form:

$$Y^i(a_{i1},..., a_{il_x}) = u_i + \frac{v_i - u_i}{2^{l_x} - 1} \cdot \left(\sum_{j=0}^{l_x - 1} a_{i(l_x - j)} \cdot 2^j \right). \qquad (10\text{-A1})$$

This mechanism implies that in the continuous space of parameters of the original problem with some fitness f only a search on grid points is performed. In this case, instead of the true global optimum point it can be expected to find that grid point whose fitness function value is the closest to optimal value.

The number l_x of bits for coding one variable x_i , $x_i \in [u_i, v_i] \subset R, u_i < v_i$, determines the distance Δx_i between two neighbor points in the grid according to

(Copia modificata per una migliore consultazione on-line)

$$\Delta x_i = \frac{v_i - u_i}{2^{l_x} - 1}, \qquad (10\text{-A2})$$

i.e. accuracy of the results can be improved by increasing l_x.

If the distance of discretization is given Δx_i, then from (10-A2) we may obtain

$$l_x = \left\{ \underbrace{Fix}_{\text{int}eger-part} \left(\log \frac{v_i - u_i}{\Delta x_i} \right\} + 1. \qquad (10\text{-A3})$$

Lecture N 11 Artificial Neural Networks: Background and Application

Consider now another soft computing methodology for simulation self-learning and adaptation mechanisms in intelligent systems based on artificial neural networks.

Artificial neural networks (ANN) are *parallel computational models comprised of densely interconnected adaptive processing units*.

Two key features of ANN define very broad applications of this model.

1) A very important feature of these networks is their *adaptive nature* where “learning by examples” replaces “programming” in solving problem.

2) *intrinsic parallel architecture* of ANN allows for fast computation of solutions in given problem area.

These properties of ANN make possible their use for:
- function approximation, when a set of data is presented;
- pattern association and recognition;
- data clastering and classification;
- learning statistical parameters;
- accumulating knowledge through training;
- prediction and forecasting;
- optimization;
- associative memory;
- non linear modeling and control.

The networks are “neural” in the sense that they have been inspired by neuroscience, the study of human brain and nervous system. Artificial neurons used are thought to be very simple models of real biological neurons. However, this does not mean that they are faithful models of biological neural or cognitive phenomena – those are of a much more complex nature.

What is Artificial Neural Network?

The history of ANN research begins with the works of McCuloch and Pitts (1943), Hebb (1949), Rosenblatt (1958,1961), Rochester (1956) , Widrow (1960), and continued by Kohonen (1972) ,Hopfiled (1982), Cohen and Grossberg (1983) and others [24].
ANN researches were inspired by biological neural networks investigations dealing with organization and functions of a human brain.
Let us look at brain organization at first.
The human brain contains about 10^{11} neurons and has about 10^{15} interconnections (links) between them. This is very big number, which is approximately equal the number of stars in the Milky Way Galaxy. The presence of such a high number of links determines a high level of massive parallelism of information processing which is specific for brain mechanism.
The human brain is a complicated communication system. The brain consists of different types of neurons. They differ in shape and in their specialized functions.
The massive parallelism property explains brain’s ability in a comparatively short time to analyze complex problems and to react adequately to unknown situations.
The main human brain’s ability is a learning ability. Modeling of this ability is the goal of ANN’s research and applications.

Biological neuron model

In Fig.11-1 a structure of a typical biological neuron is shown.

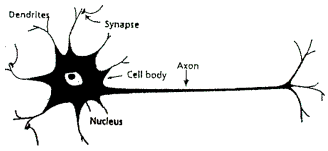


Figure 11-1. The structure of a biological neuron.

The neuron is composed of a *cell* (or cellular) body, also called the soma , ant its one or several branches.
The branches conducting information (stimulus) into a cell are called dendrites.
The branches conducting information (reaction) out of the cell are called axon.
An activation of a neuron, called an action potential, is transmitted to other neurons through its axon. A signal (called also spike) emitted from a neuron is characterized by frequency, duration, and amplitude.
The interaction between neurons takes place at strictly determined points of contact called synapses.
ANN’s are simplified models of the central nervous system. They are networks of highly interconnected neural computing elements that have the ability to respond to input stimuli and learn to adapt to the environment.

Artificial neuron model

(Copia modificata per una migliore consultazione on-line)

A mathematical model of neuron proposed by McCulloch and Pitts is shown in Fig.11-2.

Mathematical neuron computes a weighted sum of its n input signals, x_j, $j = 1,2,..., n$, and generates an output *y* equal 1 if this sum is above a certain threshold *u*. Otherwise, an output is 0. So, mathematically an *activation function* is

$$y = f(\sum_{j=1}^{n} w_j x_j - u) \qquad (11\text{-}1)$$

where f is a unit step function and w_j is the *synapse weight* associated with the *j* input. Positive weights correspond to *excitatory synapses*, while negative weights model *inhibitory* ones. Often formula (11-1) is written also as following:

$$y = f(net), \qquad (11\text{-}2)$$

where $net = (w_0 x_0 + \sum_{j=1}^{n} w_j x_j) = \sum_{j=0}^{n} w_j x_j$ and $w_0 = -u, x_0 = 1$.

The input x_0 is called a bias. In this case we can consider (*n*+1) input signals.

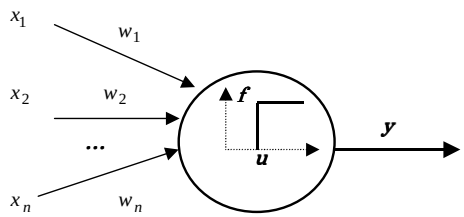


Figure 11-2. Artificial neuron model.

This model has been generalized in many ways. One way is to use different activation function, other than threshold function, such as sigmoid, or Gaussian (Fig.11-3).

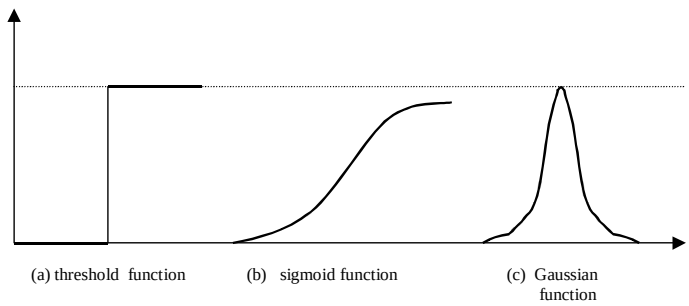


Figure 11-3. Different activation functions of ANN.

From neuron to a system of neurons

Artificial neural networks can be viewed as *weighted directed graphs in which artificial neurons are nodes and directed edges (with weights) are connections between neuron outputs and neuron inputs.*

Today too much variety of neural network's models is developed. They differ in the following points:

- the type of neuron used, and the type of calculation;
- the mathematical model used for representing and processing of information in the network including methods of training;
- the class of problems they can solve.

ANN is defined by three parameters:

1) type of neurons (also called nodes , because a neural networks described by the graph formalism);
2) neural architecture, called also as *connectionist (connection) architecture* . It means the organization of the connections (links) between neurons (nodes);
3) Learning algorithm.

Based on the connection architecture ANNs can be grouped into two classes (Fig.11-4):

- Feed-forward networks, in which there are no loops; and
- Recurrent (or Feedback) networks, in which loops occur because of feedback connections.

(Copia modificata per una migliore consultazione on-line)

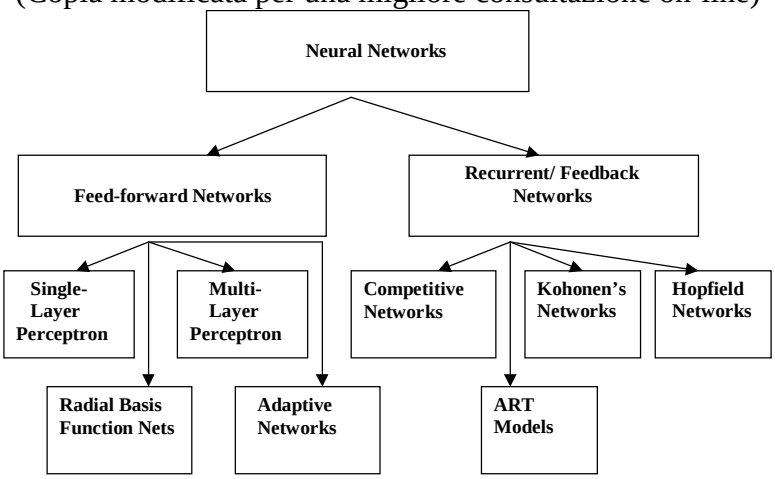


Figure 11-4. Classification of ANN's architectures.

Different types of connections yield different network behavior. The response of feed-forward networks to an input is independent of the previous network state. Recurrent networks are dynamic systems because of the feedback path the inputs to each neuron can be modified depends of previous states, and then the neuron output is computed.

Basic types of neural networks

Perceptron

The perceptron consists of a single neuron with adjustable weights $w_j \quad j = 1,2,...,n$ and threshold u, as shown in Fig.11-5.

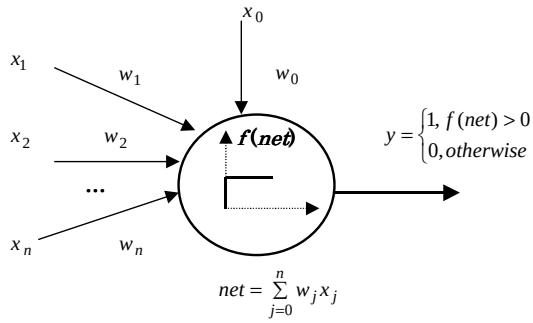


Figure 11-5. An *n* input, one output perceptron.

Given an input vector $x = (x_1, x_2, ..., x_n)^t$, the *net input* to the neuron is as follows:

$y = f(net)$, where $net = (w_0 x_0 + \sum_{j=1}^{n} w_j x_j) = \sum_{j=0}^{n} w_j x_j$ and $w_0 = -u, x_0 = 1$ and

f is a unit step function. The output y of the perceptron is equal +1 if $f(net)$ > 0, and 0 otherwise.

Perceptron can solve the following two-class classification problem: compute a two-class membership grouping of input stimuli patterns.

For example, the components of the input vector *x*, given by the $(x_1, x_2, ..., x_n)$, may describe visual properties of object. The output of the perceptron might correspond to the presence of some visual object.

In a two-class classification problem, the perceptron assigns an input pattern to class 1 if $\sum_{j=1}^{n} w_j x_j > u$,

and to the class 2 , if

$$\sum_{j=1}^{n} w_j x_j < u.$$

The linear equation

$$\sum_{j=1}^{n} w_j x_j - u = 0$$

(11-3) defines the decision boundary (a hyperplane in the *n*-dimensional input space) that halves the space.

Consider, for example, 2-dimensional case. In this case (11-3) will be as:

$$w_1 x_1 + w_2 x_2 - u = 0; \quad x_2 = \frac{w_1}{w_2} x_1 + \frac{u}{w_2}.$$

(Copia modificata per una migliore consultazione on-line)

Separating plane classification line for this case is shown in Fig. 11-6.

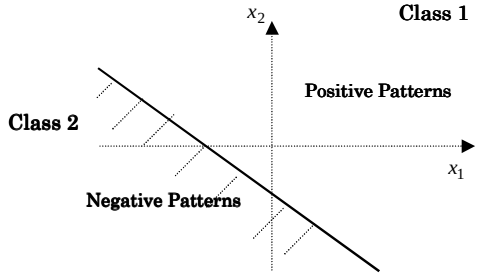


Figure 11-6. Two-class classification problem.

Layer representation of ANN

Feed-forward ANN may be represented as a set of interconnected layers, where there is no connection between neurons in the same layer.
This representation is shown in Fig.11-7.

Multiple Layer Perceptron

The Multiple Layer Perceptron (MLP) is used in a lot of practical problems including classification, recognition and control. A typical structure of MPL is shown in Fig.11-7.

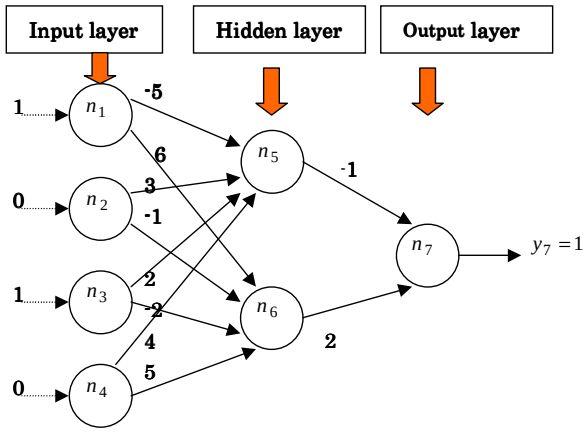


Figure 11-7. A typical structure of MLP.

According to Eq. (11-2) we can calculate net-inputs and outputs of nodes n_5, n_6, n_7 as follows:

$$f(n_5) = 1(-5) + 0 \cdot 3 + 1 \cdot 2 + 0 \cdot 4 = -3 < 0 \rightarrow y_5 = output(n_5) = 0$$

$$f(n_6) = 1 \cdot 6 + 0 \cdot (-1) + 1 \cdot (-2) + 0 \cdot 5 = 4 > 0 \rightarrow y_6 = output(n_6) = 1$$

$$f(n_7) = y_5 \cdot (-1) + y_6 \cdot 2 = 0 \cdot (-1) + 1 \cdot 2 = 2 > 0 \rightarrow y_7 = output(n_7) = 1$$

Output y_7 of the node n_7 is the final output of the multi-layered perceptron and equal 1.

Multi Layer Perceptron as Universal Approximator

If we consider a Multi Layer Perceptron (MLP) with nonlinear continuous activation functions then the following theorem is proven [25].

Existence Theorem (Hornik et al. 1989):

An MLP with one hidden layer can approximate any continuos function to any desired accuracy, subject to a sufficient number of hidden nodes.

The proof of this fundamental theorem is based on the *Kolmogorov theorem* (1957), which states that:

Any real-valued continuos function f defined on an *n*-dimensional cube can be represented as a sum of functions which have their arguments as sums of single-variable continuous functions.

Formally, it can be written as:

$$f(x_1, \ldots, x_n) = \sum_{k=1}^{2n+1} g_k \left(\sum_{j=1}^{n} \eta_{k,j}(x_j) \right), \qquad (11\text{-}4)$$

where g_k and $\eta_{k,j}$ are continuous functions.

So, for any continuous function, there is an MLP which can approximate it to a desired degree of accuracy if, for example, g and η-functions are chosen as sigmoid functions. Unfortunately, this theorem does not suggest how to construct the MLP.

(Copia modificata per una migliore consultazione on-line)

Remark. Remind a sigmoid function, which is defined as $g(x) = \dfrac{1}{1+\exp[-a(x-c)]^s}$.

Neural Networks as Associative Memories

Associative memories are used in tasks of memorizing, association and recalling of patterns.

Pattern association is the process of memorizing input patterns in an *autoassociative network* architecture in order to recall the patterns when a new input pattern is presented.

It is not required that the new input pattern be exactly the same as one that is memorized. It can be different but similar to it.

An example of autoassociative neural network is shown in Fig. 11- 8. Every neuron $j, j = 1,2,...,n$ in the network is connected back to every other one, except itself. Input patterns x_j are supplied to the *external inputs* I_j and cause activation of the *external outputs* O_k.

The response of such a network, when an input vector is supplied during the recall procedure, is dynamic, that is, after supplying the new input pattern, the network calculates the outputs and then feeds them back to the neuron. New values are then calculated, and so on, until an equilibrium is reached. The *equilibrium* is considered to be the state of the system when output signals do not change for two consecutive cycles, or change within a small constant. The weights in a Hopfield network are symmetrical for reasons of stability in reaching equilibrium, that is $w_{ij} = w_{ji}$. The network is of an additive type, that is,

$$u_j = \sum_i (w_{ij} \cdot o_j) + I_j, i \neq j,$$

where o_j (output of *j* neuron) =1 if $u_j > \Theta_j$ (threshold for the *j*th neuron), and o_j = 0 if $u_j = \Theta_j$.

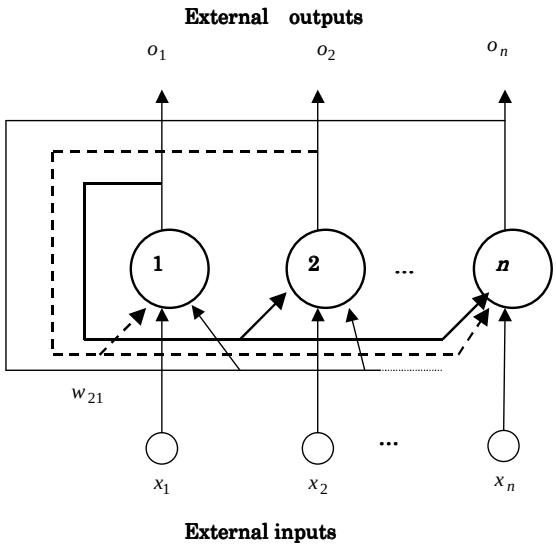


Figure 11-8. Hopfield autoassociative neural network.

Adaptive networks

The structure of adaptive network is shown in Fig. 11-9.

This network has *L* layers and layer *l* (*l* =0,1,…,*L*; *l* =0 represents the input layer) has *N*(*l*) nodes.

The output of a node depends on the input signals and the parameters set of the node.

Denote the output of node i in layer *l* as $y_{l,i}$:

$$y_{l,i} = f_{l,i}(y_{l-1,1},...,y_{l-1,N(l-1)},\alpha,\beta,\gamma),$$

where α,β,γ are the parameters of $f_{l,i}$ function of the node.

(Copia modificata per una migliore consultazione on-line)

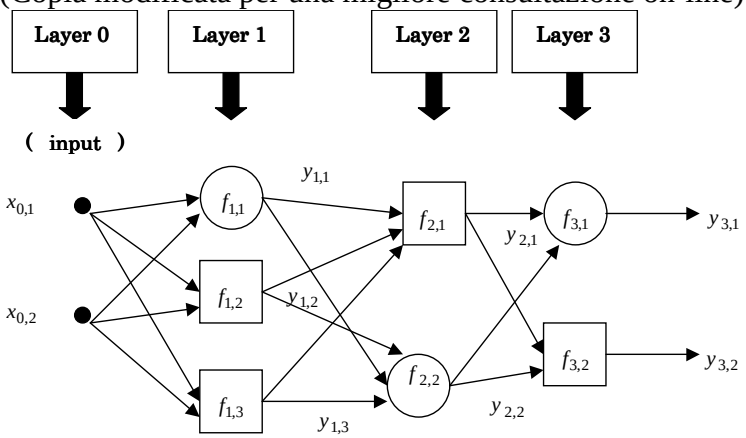


Figure 11-9. Architecture of an adaptive network.

Fuzzy neural networks

The neural network models presented so far use variants of McCulloch and Pitt's neuron to build a network.
New types of neuron – *fuzzy neurons* - have been introduced for tasks of control.
A *fuzzy neuron* has the following features, which distinguish it from the ordinary types of neurons:

- the input to the neuron represent the fuzzy input variables;
- the weights are replaced by membership functions of input variables and firing strengths of fuzzy rules ;
- a threshold level is not assigned.

A fuzzy inference process can be implemented as a generalized adaptive neural network including fuzzy neurons in their structure. This kind of neural network is called *Fuzzy neural networks* (FNN).
Remark. In some models of fuzzy neural networks in addition to membership function weights are also introduced.
Consider an example of FNN design for a fuzzy inference system based on the first-order Sugeno fuzzy model [26].

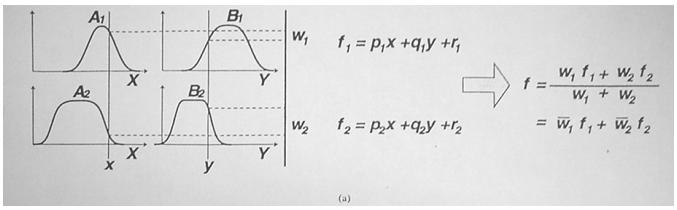
Example of an FNN design for a fuzzy inference system

For simplicity we assume that the system has two inputs *x* and *y* , and one output *z*.
Typical rules in Sugeno fuzzy model can be expressed as follows:

Rule 1: "if x is A_1 and y is B_1 then $z = f_1(x,y) = p_1 x + q_1 y + r_1$,

Rule 2: "if x is A_2 and y is B_2 then $z = f_2(x,y) = p_2 x + q_2 y + r_2$.

In Fig.11-10,(a) a reasoning mechanism for this Sugeno model is shown. In Fig.11-10, (b) the corresponding FNN architecture is shown.



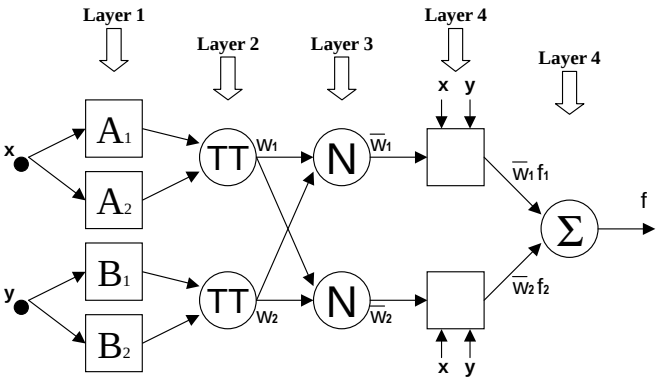


Figure 11-10. FNN architecture of Sugeno fuzzy model.

Denote the output of node i in layer *l* as $y_{l,i}$ and discuss this layered representation.

Layer 1

Every node i in this layer has output defined by $y_{1,i} = \mu_{A_i}(x)$, for i = 1,2, and

$y_{1,i} = \mu_{B_{i-2}}(y)$ for i = 3,4, where *x* and *y* is the input to the node, $\mu_{A_i}(x)$ and $\mu_{B_{i-2}}(y)$ are membership functions .

For example, they can be the generalized bell functions as follows:

(Copia modificata per una migliore consultazione on-line)

$$\mu_{A_i}(x) = \frac{1}{\left[1 + (\frac{x - c_i}{a_i})^2\right]^{b_i}},$$

where $\{a_i, b_i, c_i\}$ are the parameters set. Parameters in this layer are called *premise parameters*.

Layer 2

Every node i in this layer has output (a product of two fuzzy values) defined by:

$y_{2,i} = w_i = \mu_{A_i}(x) \times \mu_{B_i}(y)$, i =1,2.

Thus, each node in layer 2 represents the firing strength of the rule.

Remark: In this case the product operation represents the fuzzy AND, but another interpretation of fuzzy AND can be used.

Layer 3

Every node i in this layer has the following output:

$$y_{3,i} = \overline{w}_i = \frac{w_i}{w_1 + w_2}, \; i = 1,2.$$

Layer 4

Every node i in this layer has the following output :

$$y_{4,i} = \overline{w}_i f_i = \overline{w}_i (p_i x + q_i y + r_i),$$

where $\overline{w}_i$ is the output of layer 3 and { p_i , q_i , r_i } is the parameters set called *consequent parameters.*

Layer 5

Every node i in this layer computes the overall output and has the following output:

$$y_{5,1} = overal - output = \sum_i \overline{w}_i f_i = \frac{\sum_i w_i f_i}{\sum_i w_i}. \qquad (11\text{-}5)$$

We can written formula (11-5) as follows:

$$f = \frac{w_1}{w_1 + w_2} f_1 + \frac{w_1}{w_1 + w_2} f_2 = \overline{w}_1 f_1 + \overline{w}_2 f_2 =$$

$$= (\overline{w}_1 x) p_1 + (\overline{w}_1 y) q_1 + (\overline{w}_1 x) r_1 + (\overline{w}_2 x) p_2 + (\overline{w}_2 y) q_2 (\overline{w}_2) r_2.$$

By using learning algorithm the consequent parameters { p_i , q_i , r_i } and premise parameters $\{a_i, b_i, c_i\}$ can be updated.

Thus we constructed a fuzzy neural network that has the same function as a Sugeno fuzzy model but can be learned.

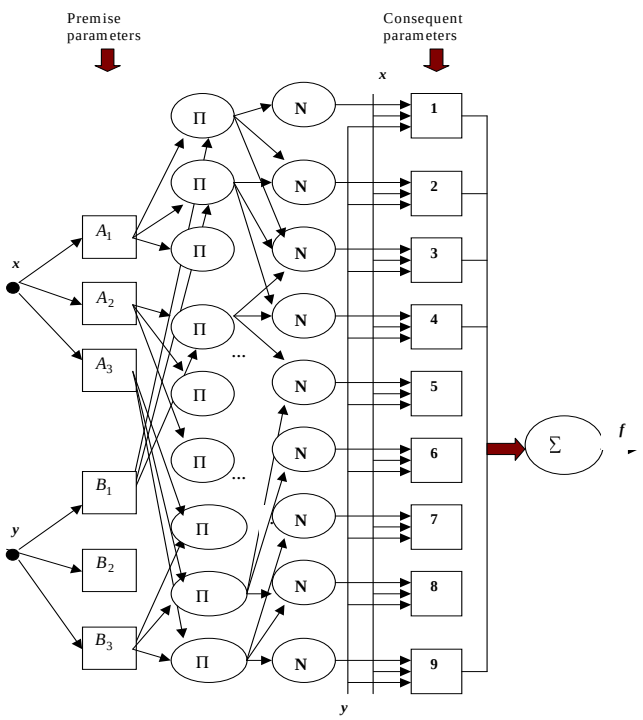


Figure 11-11.

(Copia modificata per una migliore consultazione on-line)

FNN architecture that is equivalent to a two-input first-order Sugeno fuzzy model with nine rules, where each input is assumed to have three associated membership functions is shown in Fig.11- 11.

Final discussions

ANN can be viewed as *directed graphs in which artificial neurons are nodes and directed edges (with weights or without weights) are connections between neuron outputs and neuron inputs.*

The functioning of ANN, when an the vector *x* is the input, can be considered as a mapping function *F* : $X \rightarrow Y$, where *X* is the input state space and *Y* is the output state space.

Thus, the ANN simply maps input vector *x* $\in$ X into output vector *y* $\in$ Y through the "filter"of the weights. That is output vector $y = F(W, x)$, where *W* is called the *connection weight matrix*.

The weight matrix represents the "knowledge", the long-term memory, of the ANN system, while activation values of the neuron represents the current state, the short-term memory of the ANN system.

Lets consider the following main characteristics of ANNs :

- learning and adaptation capabilities;
- generalization ability;
- massive parallelism property;
- robustness ;
- associative storage of information;
- spatio-temporal information processing.

Learning and adaptation capabilities are the fundamental trait of human intelligence. A precise definition of learning is difficult to formulate.

A *learning process* in ANN context can be viewed as *the problem of updating network architecture and connection weights so that a network can efficiently perform a specific task*.

A *generalization ability* (of ANN) means the recall process, when similar stimuli recall similar patterns of activity. For example, it can recognize full patterns from partial or noisy input patterns.

A *robustness property* means that ANN can continue to perform successfully when part of the network is disabled or when presented with noisy data.

Associative memory is characterized by its ability to store the patterns, to recall these patterns when only parts of them are given as input (Fig.11-12).

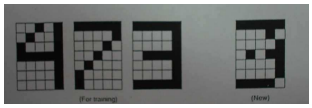


Figure 11.12. Pattern association task.

Lecture N 12 Learning in Artificial Neural Networks Fuzzy Neural Networks and GA-based FNN Tuning

One of the most important developments of recent neural network research is the discovery of learning algorithms to adjust the weights in networks. This development opened the way for more general ANN computing to solve different nonlinear problems.

The learning (or training) process in ANN is a process of mapping between the output data and the input data.

Different network architectures require appropriate learning algorithms.

The network usually must learn the connection weights from available training pattern.

To understand or design a learning process you must have a model of the environment in which an ANN operates, that is you must know what information is available to the network. This is so called a *learning paradigm*.

There are three learning paradigms: *supervised, unsupervised and hybrid*.

In *supervised learning*, the network is provided with a correct output for every input pattern, and then weights are determined as close as possible to the known correct output (Fig.12-1).

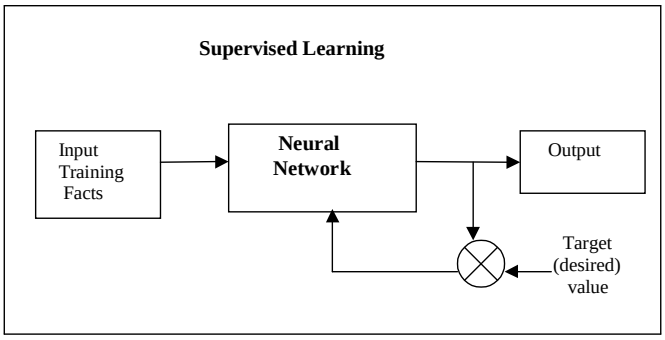


Figure 12-1. Supervised learning scheme in a neural network.

During the training process the set of training data facts is repeatedly applied to the network *until the difference between the output results and the target values are within the desired tolerance*.

In *unsupervised learning*, an known output is not needed (Fig.12-2). ANN explores the underlying structure in the input data, and organizes patterns into categories.

Unlike supervised learning in unsupervised one there is no target value. Instead, the set of data which contains the facts is repeatedly applied to the network *until a stable network output is obtained*. It has been suggested that this form of training is more similar to the biological neuron as in biological situation there is no normally a target value.

Hybrid learning combines the first two paradigms.

Designing of efficient algorithms for ANN learning is a very active research topic.

Consider typical learning rules in ANN.

(Copia modificata per una migliore consultazione on-line)

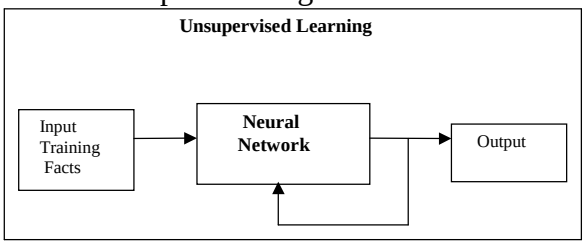


Figure 12-2. Unsupervised learning scheme in a neural network.

Error-correction rules of learning

Consider the *error-correction rules*.
This is a supervised learning paradigm, in which the network is given a desired output d for each input pattern (called also as training example). During the learning process the actual output y is generated which is not equal desired output d . The basic principle is to use the error signal $(d-y)$ to modify the connection weights to gradually reduce this error.
Evaluating an error can be done in many ways.
(1) *Err* = $|(d-y)|$ (instantaneous error);

(2) *Err* = $(d-y)^2/2$ (mean-square error (MSE));

(3) *Err* = $\sum_{(p)}\sum_{(j)} Err_j^p$ (a total MSE),

where *p* is the number of patterns (training examples), *j* is the number of outputs of neural network , and Err_j^p is the error for the *p*-training example.
There are two widely used kinds of supervisory learning based on error-correction rule: *perceptron learning* and *error back propagation-based learning*.

Perceptron Learning

The perceptron learning rule is based on the simplest case of the mentioned above error-correction principle.
Rosenblatt developed a learning procedure to determine the weights and threshold in a perceptron, given a set of training patterns [27]. The learning algorithm for a perceptron neural network consists of the following steps:

1) Initialize the weights w_j (*j* = 0,1. 2,...*n*) to small random numbers.

2) Present a pattern input $(x_1, x_2, ..., x_n)^t$, where *t* is the iteration number.

3) Calculate the actual output of perceptron neuron as follows:

$$y = \begin{cases} 1, & if \sum_{j=0}^{n} w_j x_j > 0, \\ 0, & otherwise \end{cases}$$

4) Compute the error *Err* = $(d-y)$, where d is the desired output for the input Pattern $(x_1, x_2, ..., x_n)^t$.

5) Update (modify) the weights according to :
$w_j(t+1) = w_j(t) + \eta \cdot Err \cdot x_j$,where η is a learning coefficient - a number between 0 and 1 .

6) Repeat steps 2 - 5 until error becomes sufficiently low ($Err < Err_{\max}$), that is the perceptron goes into a convergence.

Back-Propagation Learning

Consider the back-propagation learning algorithm for the feed-forward type neural networks with hidden layers and *m*-output node (Fig.12-3).

(Copia modificata per una migliore consultazione on-line)

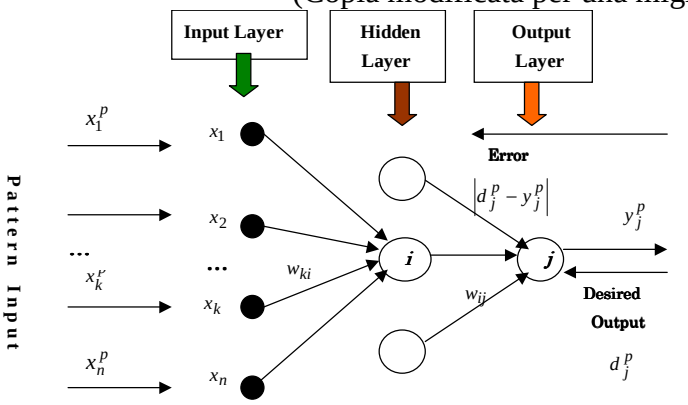


Figure 12-3. A schematic representation of error back-propagation learning

Neurons in these networks have continuous value inputs and outputs and nonlinear activation function.

In order to find optimal connection weights w_{ij} which minimize a global error *E*, we will use a *gradient descent rule*. This rule says: a change of a weight Δw_{ij} at a cycle $(t+1)$ is *in the direction of the negative gradient of the error E* .

The general formula will be as follows:

$$\Delta w_{ij}(t+1) = -\eta \frac{\partial E}{\partial w_{ij}(t)}, \qquad (12\text{-}1)$$

where η is a learning rate coefficient.

This rule ensures that after a number of cycles, the error *E* will reach a minimum value.

A global error for all the training examples can be calculated as follows:

$$E = \sum_{(p)} \sum_{(j)} Err_j^p ,$$

where *p* is the number of patterns (training examples), *j* is the number of outputs of neural network , and Err_j^p is the error for the *p*-training example which can be calculated, for example as MSE :

$$Err_j^p = \frac{(d_j^p - y_j^p)^2}{2},$$

where d_j^p is the desired output of the *j*-th output node for the *p*-training example, y_j^p is the actual output of the *j*-th output node for the *p*-training example.

Usually two types of (12-1) are used for training feed-forward NN.

Delta rule:

$$\Delta w_{ij}(t+1) = \eta \cdot Err_j \cdot y_i \qquad (12\text{-}2)$$

and

Generalized delta rule:

$$\Delta w_{ij}(t+1) = \eta \cdot Err_j \cdot g'(net_j) \cdot y_i \qquad (12\text{-}3)$$

where Err_j is the error between desired output value and actual value of the *j* output node, $g'(net_j)$ is the derivative of the activation function of the *j* output node, i.e.

$$g'(net_j) = \left.\frac{\partial g(z)}{\partial z}\right|_{z=net} .$$

If the activation function is, for example, the logistic function $g(z) = \dfrac{1}{1+e^{-z}}$, then

$$g'(z) = \left[-\frac{1}{(1+e^{-z})^2}\right] \cdot e^{-z} = \left(\frac{1}{1+e^{-z}}\right)\left(1 - \frac{1}{1+e^{-z}}\right) = g(z)(1-g(z)) \qquad (12\text{-}4)$$

By using (12-4) we can write

$$g'(net_j) = g(net_j)(1-g(net_j)) = y_j(1-y_j) \qquad (12\text{-}5)$$

Eq. (12-3) can be written now as:

$$\Delta w_{ij}(t+1) = \eta \cdot Err_j \cdot y_j \cdot (1-y_j) \cdot y_i \qquad (12\text{-}6)$$

(Copia modificata per una migliore consultazione on-line)

Discuss now how to calculate Err_j? If *j*-th node is an *output node*, then we may calculate it as $\left| d_j^p - y_j^p \right|$, but how to find error for hidden nodes.

Consider two versions of back-propagation algorithms.
The training algorithm consists of two passes: (1) a *forward pass*, when inputs are supplied and propagated trough the hidden layers to the output layer; and (2) *a backward pass*, when an error is calculated at the outputs and propagated backward for calculating the weight's changes.
Changing weights can be done by two ways: (1) by minimizing of average error for all training examples; (2) the error is calculated and weights are changed after every training example.

Back-propagation Algorithm 1

Forward pass:

1. Initialize the weights to small random values.
2. Randomly choose an input pattern $\vec{x} = (x_1, x_2, ... x_n)^p$.
3. Propagate the input vector $\vec{x}$ forward through the network and calculate the output.
4. For every output neuron *j* (layer *L*) compute Err_j as: $Err_j = \left| d_j^p - y_j^p \right|$.

Backward pass:

5. Adjust the weights between the intermediate neurons (layer *L*-1) i and output neurons j according to the calculated error:

   $\Delta w_{ij}(t+1) = \eta Err_j y_j (1 - y_j) y_i + \alpha \Delta w_{ij}(t)$, where α is a parameter, called *momentum*.
6. Calculate the error Err_i for neurons i in the intermediate layer as:

   $Err_i = \sum_j Err_j w_{ij}$.
7. Propagate the error back to the neurons k of lower layer $(l = L-2,...,0)$:

   $\Delta w_{ki}(t+1) = \eta Err_i y_i (1 - y_i) y_k + \alpha \Delta w_{ki}(t)$
8. Go to step 2 and repeat for the next pattern until the error in the output layer is below the given value or a maximum number of iterations is reached.

Back-propagation Algorithm 2

1) Initialize the weights to small random values.
2) Randomly choose an input pattern $\vec{x} = (x_1, x_2, ... x_n)^p$.
3) Propagate the input signal forward through the network.
4) Compute the error signal for the node i in the output layer (l = *L*) as

   $\varepsilon_{L,i} = g'(net_{L,i})(d_i^p - y_i^L)$, here $g'(net_{L,i})$ is the derivative of the activation function of the node i, $net_{L,i}$ is the *net input* to the node i.
5) Compute the error $\varepsilon_{l,i}$ for node i in layer l by propagating the errors backward by the following way:

   $\varepsilon_{l,i} = g'(net_{l,i}) \sum_j w_{ij}^{l+1} \varepsilon_{l+1}$ for $l = (L-1),...,1$
6) Update weights as follows:

   $\Delta w_{ij}^l (t+1) = \eta Err_j y_i = \eta \varepsilon_{l,i} y_i^{l-1}$ for $l = 1,2,...,L$
7) Go to step 2 and repeat for the next pattern until the error in the output layer is below the given threshold or a maximum number of iterations is reached.

Back-Propagation Learning in Feed-Forward Adaptive Networks

Consider the back-propagation learning rule for generalized type of networks called feed-forward adaptive networks. The structure of a feed-forward adaptive network was introduced in Lecture 11. Let us repeats it here (Fig.12-4).

(Copia modificata per una migliore consultazione on-line)

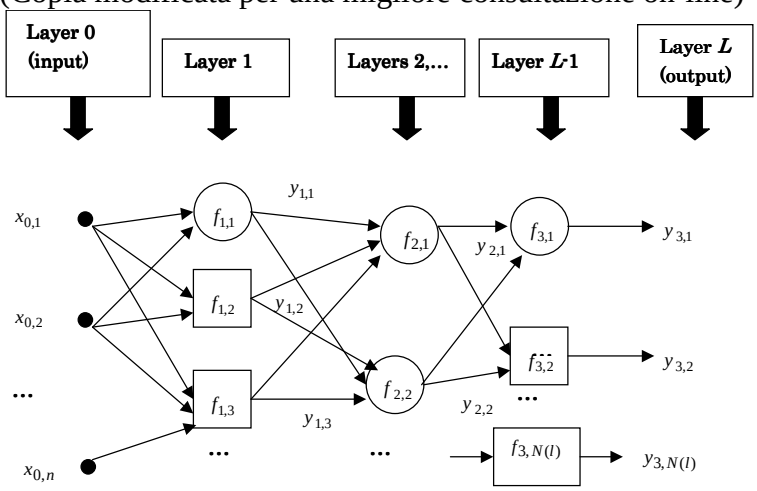


Figure 12-4. The architecture of feed-forward adaptive networks.

This network has L layers and layer l (l = 0, 1,...,L; l = 0 represent the input layer) has N (l) nodes. The output of a node i in the layer l depending on the input signals and the parameter set of the node is described as follows:

$$y_{l,i} = f_{l,i}(y_{l-1,1}, \ldots, y_{l-1,N(l-1)}, \alpha, \beta, \gamma, \ldots),$$

where α, β, γ are the parameters set of the node i .

The central idea of a learning rule for adaptive networks concerns how to obtain learning algorithm in which the error measure is minimal with respect to parameters. That is we want to construct the algorithm of updating (learning) of parameters of network. In this case we also use a *gradient descent rule*.

Before the calculation of the gradient vector, we can note that change in parameter α in the node i of the level l results in as shown in Fig.12-5.

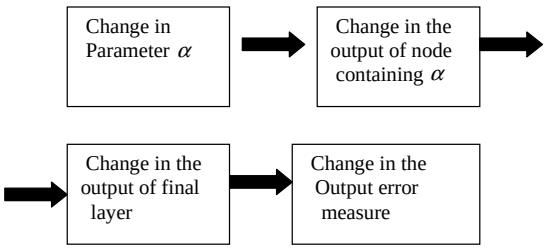


Figure 12-5. Results of change of parameter α in the node i of the level l .

Assuming the given training data set has P training examples. Define the error measure for the pattern p (p = 1,2,...,P) as

$$E^p = \sum_{k=1}^{N(L)} (d_k^p - y_{L,k}^p)^2 \cdot \qquad (12\text{-}7)$$

where d_k^p is the k-component of the desired output vector for input vector p , and $y_{L,k}^p$ is the actual output vector for input vector p .

We define *error signal* $\varepsilon_{l,i}$ as the derivative of the error measure with respect to the output of node i in layer l :

$$\varepsilon_{l,i} = \frac{\partial E^p}{\partial y_{l,i}}$$

The error signal for node i at the L output layer can be calculated as:

$$\varepsilon_{L,i} = \frac{\partial E^p}{\partial y_{L,i}}$$

By using (12-7) we can write:

$$\varepsilon_{L,i} = -2(d_i - y_{L,i}) \cdot$$

The error signal for node i at the hidden layer l is a function of error signals of nodes at layer (l +1) and derivative of the activation function of layer (l +1).

Thus the error signal for node i at the hidden layer l can be derived by the following way:

$$\varepsilon_{L,i} = \frac{\partial E^p}{\partial y_{L,i}} = \sum_{m=1}^{N(l+1)} \frac{\partial E^p}{\partial y_{l+1,m}} \cdot \frac{\partial f_{l+1,m}}{\partial y_{l,i}} = \sum_{m=1}^{N(l+1)} \varepsilon_{l+1,m} \cdot \frac{\partial f_{l+1,m}}{\partial y_{l,i}} . \quad (12\text{-}8)$$

The gradient vector is defined as the derivative of the error with respect to each parameter. If α is a parameter of the node i in layer l , we can write by using Eq. (12-8)

(Copia modificata per una migliore consultazione on-line)

$$\frac{\partial E^p}{\partial \alpha} = \frac{\partial E^p}{\partial y_{l,i}} \cdot \frac{\partial f_{l,i}}{\partial \alpha} = \varepsilon_{l,i} \cdot \frac{\partial f_{l,i}}{\partial \alpha}.$$

The total error measure with respect to α is:

$$\frac{\partial E}{\partial \alpha} = \sum_{p=1}^{P} \frac{\partial E^p}{\partial \alpha}.$$

The update (learning) rule for the parameter α is expressed as follows:

$$\Delta\alpha = -\eta \frac{\partial E}{\partial \alpha}$$

where η is the learning rate coefficient.

Unsupervised Learning in Neural Networks

There are two main approaches in the contemporary unsupervised algorithms: (1) *noncompetitive learning*, and (2) *competitive learning*. *In the first case* many neurons may be activated at a time, *in the second case* neurons compete, *and after that*, only one neuron is activated at one time.

Noncompetitive Learning

Most of today noncompetitive training algorithms [3,20,29] are influenced by the Hebb training model which is expressed as follows:

$$w_{ij}(t+1) = w_{ij}(t) + c \cdot o_i \cdot o_j,$$

where $w_{ij}(t)$ is the weight of the connection between the i th and j th neuron at the moment t, and o_i and o_j are the output signals of neurons i and j at the same time moment t. The weight $w_{ij}(t+1)$ is the adjusted weight at the next moment (t + 1). This principle was used in supervised learning, but then we knew the desired outputs. Here, the outputs are as they are produced by the network only. Different modifications of this rule have been suggested, for example, the differential Hebbian learning law (Kosko,1988):

$$w_{ij}(t+1) = w_{ij}(t) + c \cdot o_i \cdot o_j + \Delta o_i \Delta o_j.$$

The differential Hebbian learning law introduces the first derivatives of the activation signal to the Hebbian law.

- Grossberg's competitive law (Grossberg,1982) expressed as:

$$\Delta w_{ij} = c \cdot o_j \cdot (o_i \cdot w_{ij}),$$

- The differential competitive learning law (Kosko,1990):

$$\Delta w_{ij} = c \cdot \Delta o_j \cdot (o_i - w_{ij})$$

The differential competitive learning law introduces the first derivative of neuronal output values to the competitive learning law.

- Adaptive vector quantization (Kohonen,1982,1990):

$$w_{ij}(t+1) = w_{ij}(t) + c \cdot (\vec{x}(t) - \vec{w}_j(t)),$$

where c is a learning rate, $\vec{x}(t)$ is the input vector at moment t, $\vec{w}_j$ is the vector of the weights from the input neurons to the neuron j.

Unsupervised learning is also applicable when noise is present. For example, the random signal Hebbiian law looks like:

$$w_{ij}(t+1) = w_{ij}(t) + c \cdot o_i \cdot o_j + n_{ij},$$

where n_{ij} is a noise introduced to the connection $i \rightarrow j$.

Competitive Learning

With no available information regarding the desired outputs, unsupervised learning networks update weights only on the basis of the input patterns. The *competitive learning network* is a popular scheme to achieve this type of unsupervised data clustering or classification. In Fig. 12-6 an example of a competitive learning network architecture is shown.

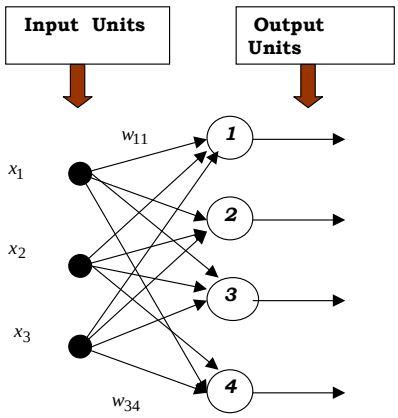


Figure 12-6. Competive learning network.

(Copia modificata per una migliore consultazione on-line)

All input units i are connected to all output units j with weights w_{ij}. The number of inputs is the input dimension, while the number of outputs is equal to the number of clusters that the data are to be divided into.
A cluster center's position is specified by the weight vector connected to the corresponding output unit. For the simple network in Fig.12-7, the three-dimensional input data are divided into four clusters, and the cluster centers, denoted as the weights, are updated via the competitive learning rule.

The input vector $\vec{x} = [x_1, x_2, x_3]^T$ and the weight vector $\vec{w}_j = [w_{1j}, w_{2j}, w_{3j}]^T$ for an output unit j are generally assumed to be normalized to unit length. The activation value a_j of output unit j is then calculated by the inner product of the input and weight vectors:

$$a_j = \sum_{i=1}^{3} x_i w_{ij} = \vec{x}^T \cdot \vec{w}_j = \vec{w}_j^T \cdot \vec{x}$$

Next, the output unit with the highest activation must be selected for further processing, which is what is implied by *competitive*. Assuming that output unit k has the maximal activation, the weights leading to this unit are updated according to the competitive or so-called winner-take-all learning rule:

$$\vec{w}_k(t+1) = \frac{\vec{w}_k(t) + \eta(\vec{x}(t) - \vec{w}_k(t))}{\left\| \vec{w}_k(t) + \eta(\vec{x}(t) - \vec{w}_k(t)) \right\|}.$$

The preceding weight update formula includes a normalization operation to ensure that the updated weight is always of unit length. Note that only the weights at the winner output unit k are update; all other weights remain unchanged.

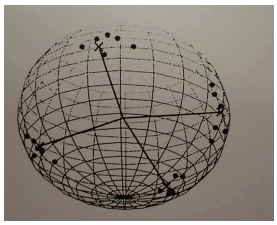


Figure 12-7. Clastering of input data based on competitive learning paradigm.

Training Algorithm for Hopfield Networks

The training procedure for a Hopfield network is reduced to a simple calculation of the weights w_{ij} on the basis of the training examples with the use of the following formula:

$$w_{ij} = \left(\sum_{p=1}^{m} (2 \cdot x_i^{(p)} - 1) \cdot (2 \cdot x_j^{(p)} - 1) \right)$$

where the summation is held for all the training patterns $x^{(p)}$, $x_i^{(p)}$ is the i th binary value of the input pattern *p*; and the expression in the parenthesis can be only 1 or 0 according to the value of the input pattern.
An interesting characteristic of the weights w_{ij} is that they measure the correlation between the frequencies of firing of neurons i and j over the full set of examples. It is a variant of the Hebbian learning law, that is, the connection weights increase if two adjacent nodes fire simultaneously.

Recall Procedure for Hopfield Networks

The recall procedure is described by the following steps:

1. Apply new input pattern x^{new}.
2. Assign initial values $o_j(0)$ to the outputs of the neurons *j*, *j* = 1,2,…,*n*, to be the corresponding input values: $o_j(0) = x_j^{new}$.
3. Calculate the next output value $o_j(t+1)$ as follows: $o_j(t+1) = s(u_j(t+1))$, where s is the threshold activation function and $u_j(t+1)$ is calculated as a sum.
4. Repeat step for *t* = 1,2,3, and so on until all the outputs no longer change their values for at least two consecutive cycles (moments *t* and *t*+1). Such a state of the network is called a stable state. In this case the network reaches an equilibrium.

The recall procedure executed as shown above can be organized in following modes:
- asynchronous updating: each neuron may change its state at any time;
- synchronous updating: all neurons change their states simultaneously at a given moment;
- sequentional updating: only one neuron changes its state at any moment, thus all neurons change their states, but sequentially.

GA-based Training of Neural Networks: an example

Consider the task of design of fuzzy neural network (FNN) for cart-centering control problem and the FNN training by using GA.
Let refresh some aspects of this task (see Lecture N9). We considered the following problem: design a fuzzy controller which for given initial velocity and position on the track will bring the cart to zero velocity and zero location in minimum time.
The input variables for this problem is a location, *x*, of the cart, and a velocity of the cart *v*. The output variable is force *F* applied to the cart. Equations of motion for the cart are:

(Copia modificata per una migliore consultazione on-line)

$$x(t+\tau) = x(t) + \tau\, v(t)$$
$$v(t+\tau) = v(t) + \tau\, F(t)/m,$$

where τ is the time step, *m* is a mass of the cart.
We introduce the following fuzzy rules:
IF X is {NM,…, PM} and *v* is {NM,…, PM} THEN *F* is { NM, NS, ZE, PS, PM}.

GA have found a set of fuzzy rules for optimal control (in minimal time), and for each step τ the following pairs are determined:

$$(x_1, v_1) \to F_1, (x_2, v_2) \to F_2, \ldots, (x_\tau, v_\tau) \to F_\tau$$

Now we may consider these pairs as the training pairs for our fuzzy neural network.

Let input-output fuzzy variables are described by Gaussian membership functions: $\mu_{F_i}(x_i) = 1 \cdot e^{-\frac{(x-\bar{x}_i)^2}{\sigma_i^2}}$.

FNN-tuning task: by using the error back-propagation algorithm perform training of FNN with teaching patterns generated by GA (Fig 12-7).

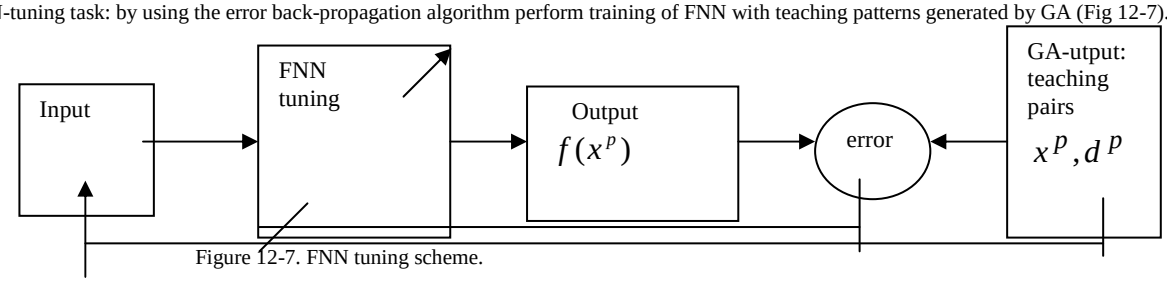


Figure 12-7. FNN tuning scheme.

Error is calculated as ($f(x^p) - d^p$).
FNN imitates the work of a fuzzy system (in our case, a fuzzy controller). Let us define the fuzzy inference model for our task. It contains the *max-product* inference method, singleton-fuzzyfier, and COG (the center of gravity) defuzzyfier.
Let us write a general form of a fuzzy rule by the following way:

Rule *l:*

IF *AND* (x_i is $\mu^l{}_{F_i}$) THEN (y^l is $\mu^l{}_y$)

According to the theorem of Wang (see Lecture 7), we can calculate the total output of given fuzzy system as follows:

$$y = f(x_1, \ldots, x_n) = \frac{\sum_{l=1}^{M} \bar{y}^l \prod_{i=1}^{n} \mu^l{}_{F_i}(x_i)}{\sum_{l=1}^{M} \prod_{i=1}^{n} \mu^l{}_{F_i}(x_i)} = \frac{a}{b} = \frac{\sum_{l=1}^{M} \bar{y}^l z^l}{\sum_{l=1}^{M} z^l}, \text{where } z^l = \prod_{i=1}^{n} \mu^l{}_{F_i}(x_i).$$

Here $\bar{y}^l$ is the point of maximum value of $\mu_y{}^l(y)$.
The architecture of the fuzzy neural network for a fuzzy cart control is shown in Fig.12-8 (a,b).
Consider now the following task:

By using training input-output patterns update FNN-parameters ($\bar{y}^l, \bar{x}_i{}^l, \sigma_i{}^l$) so that the total error measure E^p is minimal.

Define E^p as follows: $E^p = \sum_p \frac{1}{2}(f(x^p) - d^p))^2$.

We will update our parameters according *gradient descent method* where for some parameter α its change is given by $\Delta\alpha = -\eta \frac{\partial E^p}{\partial \alpha}$.

Calculate derivatives of error measure function for our parameters:

$$\frac{\partial E^p}{\partial \bar{y}^l} = \frac{\partial}{\partial \bar{y}^l}\left(\frac{1}{2}(f(x^p) - d^p)^2\right) = (f(x^p) - d^p) \cdot \frac{\partial f(x^p)}{\partial \bar{y}^l} =$$
$$= (f(x^p) - d^p)\frac{1}{b} z^l \;;$$
$$\frac{\partial E^p}{\partial \bar{x}_i{}^l} = \frac{\partial}{\partial \bar{x}_i{}^l}\left(\frac{1}{2}(f(x^p) - d^p)^2\right) = (f(x^p) - d^p) \cdot \frac{\partial f(x^p)}{\partial \bar{x}_i{}^l} =$$
$$= (f(x^p) - d^p)\frac{1}{b}(\bar{y}^l - f(x^p)) \cdot z^l \cdot \frac{2(x_i{}^p - \bar{x}^l{}_i)}{(\sigma^l{}_i)^2} \;;$$

(Copia modificata per una migliore consultazione on-line)

$$\frac{\partial E^p}{\partial \sigma_i^l} = \frac{\partial}{\partial \sigma_i^l}\left(\frac{1}{2}(f(x^p)-d^p)^2\right) = (f(x^p)-d^p)\cdot\frac{\partial f(x^p)}{\partial \sigma_i^l} =$$

$$= (f(x^p)-d^p)\frac{1}{b}\cdot z^l \cdot \frac{2(x_i^p - \bar{x}^l{}_i)^2}{(\sigma^l{}_i)^3}.$$

Error back-propagation algorithm for FNN training developed for the task of neuro-fuzzy cart-centering control based on GA:

1) Choose randomly (or apriori) FNN parameters as small numbers (between 0 and 1); set up $k = 0$; $E^p = 0$;
2) Set up $p = 0$;
3) If $p = P$ (the given number of patterns), then (if $E^p < \varepsilon_{\max}$, then stop , else go to step 2);
4) $p = p+1$;
5) propagate the input p-pattern through the FNN, calculate output;
6) Calculate the total error as

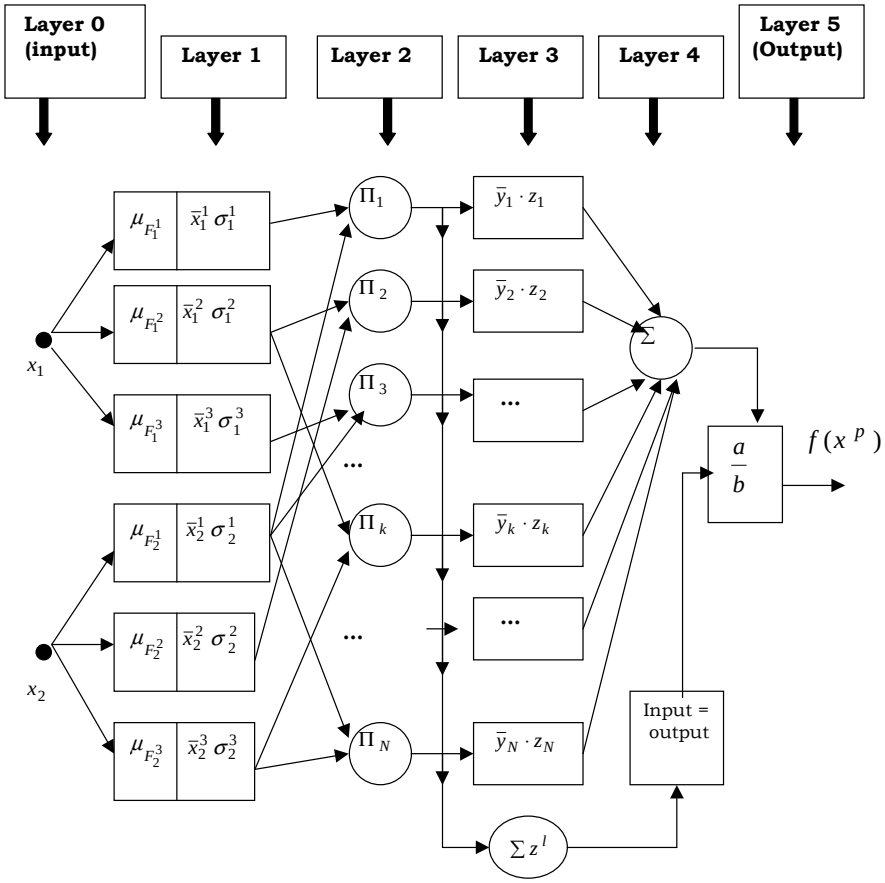
$$E^p = E^p + 1/2(f(x^p)-d^p)^2;$$

7) Update parameters as follows:

$$\bar{y}^l(k+1) = \bar{y}^l(k) - \eta\frac{\partial E^p}{\partial \bar{y}^l};$$

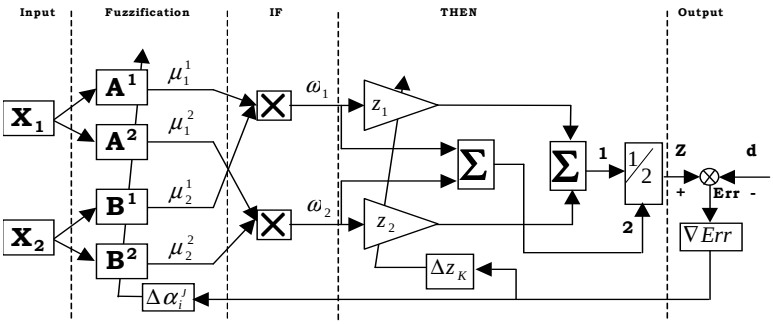
$$\bar{x}_i^l(k+1) = \bar{x}_i^l(k) - \eta\frac{\partial E^p}{\partial \bar{x}_i^l}, \quad \sigma_i^l(k+1) = \sigma_i^l(k) - \eta\frac{\partial E^p}{\partial \sigma_i^l},$$

8) Go to Step 3.



(a)

(Copia modificata per una migliore consultazione on-line)



$$Err = \frac{\sum_K \left(\prod_i e^{-\frac{(x_i - x_c^j)^2}{(\sigma^j)^2}}\right) z_K}{\sum_K \left(\prod_i e^{-\frac{(x_i - x_c^j)^2}{(\sigma^j)^2}}\right)} - d$$

$$\Delta\{x_c^j, \sigma^j, z_K\} = -\eta \frac{\partial Err}{\partial\{x_c^j, \sigma^j, z_K\}}$$

$$\{x_c^j, \sigma^j, z_K\}^{NEW} = \{x_c^j, \sigma^j, z_K\}^{OLD} + \Delta\{x_c^j, \sigma^j, z_K\};$$

MF: $e^{-\frac{(x_i - x_c^j)^2}{(\sigma_i^j)^2}}$

(b)

Figure 12-8. The FNN architectures (a,b) of the cart fuzzy controller.

Lecture N 13 Soft Computing Application Benchmark: Intelligent Robust Control of Extension-Cableless Robotic Unicycle

Discuss now the applications of soft computing algorithms for design of *Benchmarks* of advanced robotics and intelligent mechatronics. Consider the example of soft computing application for the intelligent robust control system design of the *Extension-Cableless Robotic Unicycle* [30] (Fig. 13-1). (Details of the robotic unicycle hardware design are given in Appendix.)

Intelligent mechatronics is based on researching results of new kinds of nonlinear mechanical systems motion, modern control methods and intelligent computation for the development of smart control algorithms. The extraction of knowledge from new kinds of a movement is based on the study of different Benchmarks. The unicycle motion is a new kind of movement and described as a *nonlinear nonholonomic, global unstable dynamic system*. The research of such dynamic systems is quite interest as for nonlinear mechanics (for development of new research methods of nonlinear effects) and so for the modern control theory (for development of new intelligent control algorithms).

Unicycle systems developed earlier were considered only from the mechanical point of view, i.e. from the point of a mechanical model design. Earlier researchers didn't take into account that a unicycle control is realized by a skilful human-operator, i.e. they didn't consider the unicycle as a biomechanical system. Now we do it. We consider the unicycle as the *biomechanical system* including the new phenomena in a control system such as intuition, instinct and emotion. It is the algorithmic unsolvable problem for advanced control system theory based on traditional calculation methods. But it solvable problem for the new intelligent control method based on soft computing [18]. Background of this method is a qualitative physical analysis of unicycle dynamic motion and introduction of intelligent level in control system with realization of instinct and intuition mechanisms on basis of FNN and GA.

Control of a unicycle movement is based on the coordination of complex movement's components (pedaling and movement of an operator trunk). A change of coordination types gives new types of movements (straightforward movement, obstacles avoidance, dance, jumping and etc.).

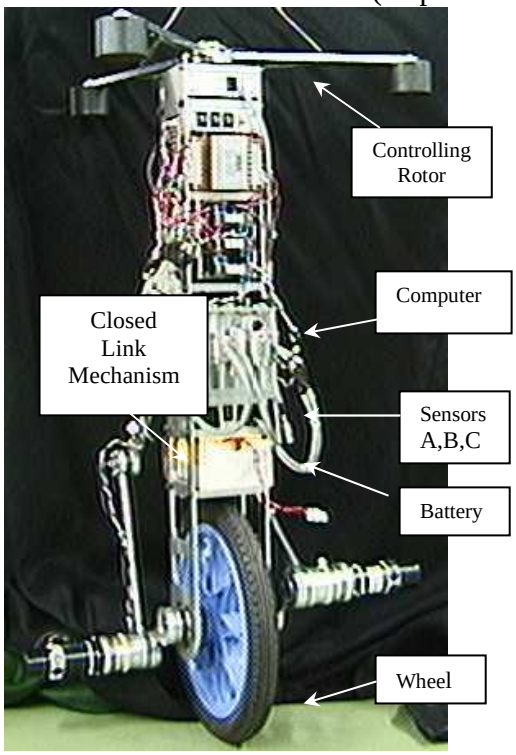
Remark 1. A unicycle is a good example of a simulator in particular for rehabilitation and training invalids, for example, to use an artificial limb.

Figures 13-2 and 13-3 show the new structure of fuzzy simulation of intelligent control of *Extension-Cableless Robotic Unicycle.*

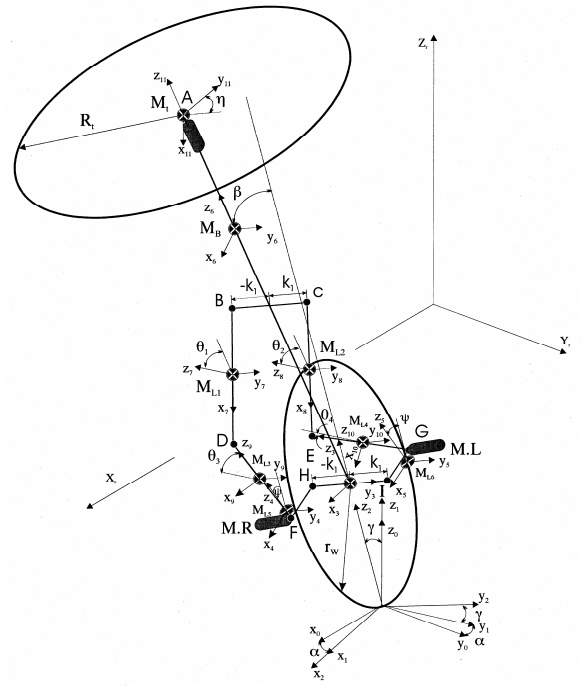
Remark 2. In Fig.13-3 the following designations are used:

1. GA – Genetic Algorithm;
2. f - Fittness Function of GA;
3. S- Entropy of System;
4. S_c - Entropy of Controller;
5. S_i - Entropy of Controlled Plant;
6. □ - Error;
7. u*- Optimal Control Signal;
8. m(t) – Disturbance;
9. FC - Fuzzy Controller;
10. FNN - Fuzzy Neural Network;
11. FLCS - Fuzzy Logic Classifier System;
12. SSCQ - Simulation System of Control Quality;
13. K - Global Optimum Solution of Coefficient Gain Schedule (Teaching Signal);
14. LPTR - Look-up Table of Fuzzy Rules;
15. CGS - Coefficient Gain Schedule $K=(k_1,k_2,k_3,k_4)$.

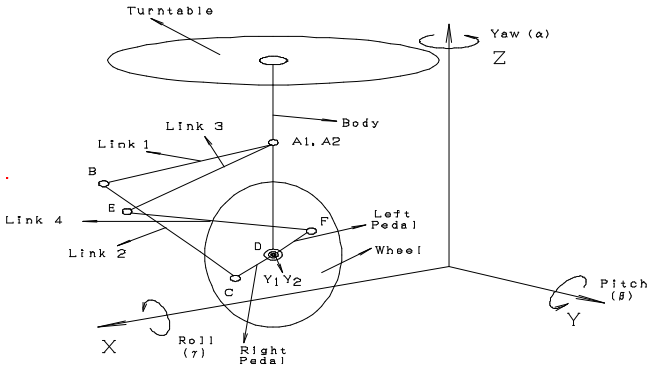
(Copia modificata per una migliore consultazione on-line)



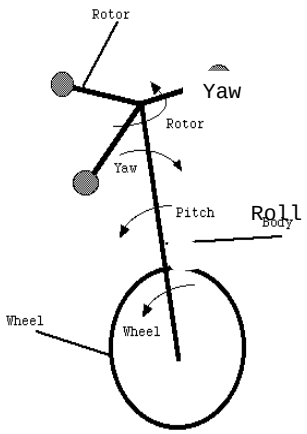
a)



b)



c)



d)

Figure 13-1. a) Photo of extension-cableless robotic unicycle; b) Coordinate description of the unicycle model; c) "Complicated" model for emulating human riding of a unicycle model. d) "Simple" model for emulating human riding of a unicycle model.

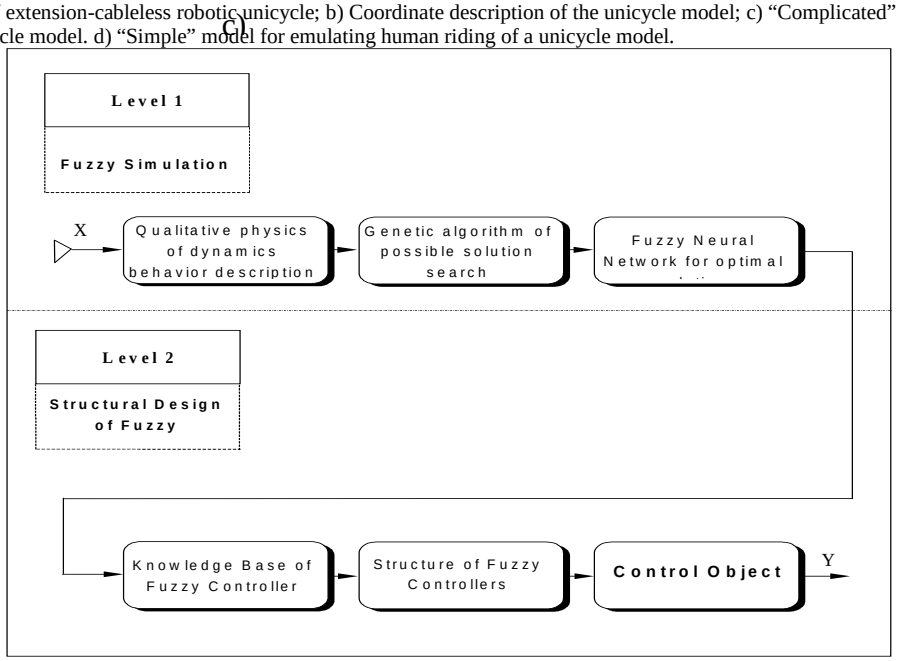


Figure 13-2. The structure of fuzzy simulation based on soft computing for Intelligent Controller Design.

(Copia modificata per una migliore consultazione on-line)

The control of nonlinear global unstable objects as unicycle demands introducing new control principles. We introduce the following new physical control principle – *the principle of* <u>*minimum entropy production rate*</u> *in control systems and in control object motion in general* [31].

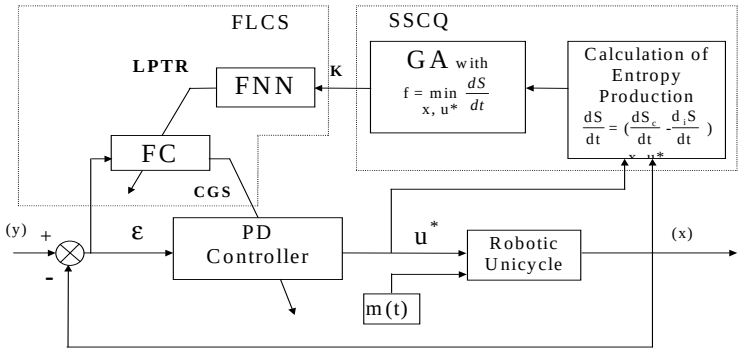


Figure 13-3. Self-Organization Structure of Intelligent Robust Control System with the new Physical Measure of Control Quality.

The physical measure of entropy production rate is the fitness function in GA. Such approach guarantees the global dynamic stability of control object (our unicycle) and provides *robust control*.

In off-line mode the calculation of entropy production rate in a movement and control system is carried out. Entropy production rate and entropy measures for the robotic unicycle motion and the control system are calculated directly from the proposed thermodynamic equations of motion (see below). On the basis of entered fitness function the GA selects (from a space of all possible decisions) an optimal decision, which is the law of change PD - Controller parameters (the control law). In the next step, the control law is the teaching signal for FNN, which performs training and adaptation to the given control law. The FNN output signal forms the look-up-table for the Fuzzy Controllers. On basis of the developed look-up-tables changing the parameters of PD Controllers is carried out.

The Biomechanical Control Model of the Robotic Unicycle

Human riders (called also as human-operators) controlling actions on a unicycle are using their torsos, shoulders and arms in a quite complicated mode.

The study of human rider's stability control of a unicycle was began with the observation and analysis of the human riding behavior due to vestibular and control systems (Fig.13-4).

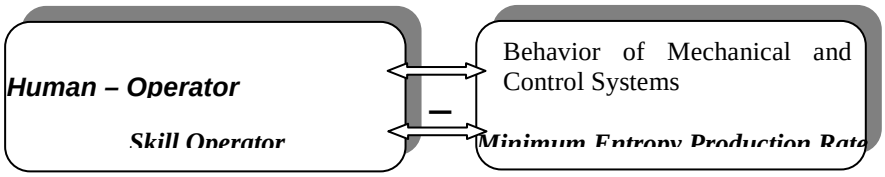


Figure 13-4. Conceptual Interrelation.

From the observation and analysis we found that the rider body's thighs and shanks construct a two closed links loop. This special mechanism plays an important role for the rider's postural stability control in the unicycle system (Fig.13-1). To emulate human riding a unicycle by a robot including an intuition and instinct control mechanisms (like in a human rider's behavior), the new biomechanical model of the robotic unicycle was developed in [31]. In the unicycle model, two unique and characteristic structures are contrived. One is an overhead rotor (hereafter, rotor) mounted on the torso (body) and another is the double 4 bar-closed link mechanism on both sides of the wheel. These two structures are considered to play important roles in the biomechanical control system.

For the computer simulation of a skillful human-operator's intuition mechanism, the GA with a fitness function as a physical measure of entropy production rate was used. So, the intuition mechanism is considered to be like GA performing a *global* randomized search of optimal control for a global stability of the robotic unicycle throughout the full space of possible solutions. The instinct mechanism is considered as a *local* active adaptation process with the minimum entropy production rate in the learning process of a vestibular system. For the computer simulation of the skillful human-operator's instinct mechanism FNN was used.

Qualitative Physics and Thermodynamic Equations of Motion for the Extension-Cableless Robotic Unicycle Model

First of all for the entropy measures of the robotic unicycle motion and the control system, *thermodynamic equations of motion* are proposed [31].

The thermodynamic equations of motion for the robotic unicycle with a symmetric rotor are given as follows,

$$\begin{bmatrix} \ddot{q} \\ \lambda \end{bmatrix} = \begin{bmatrix} M(q) & -\dfrac{\partial c}{\partial q} \\ E(q) & 0 \end{bmatrix}^{-1} \begin{bmatrix} \tau - B(q)[\dot{q},\dot{q}] - C(q)[\dot{q}^2] - D(q)[\dot{q}] - G(q) \\ -F(q,\dot{q}) \end{bmatrix}, \qquad (13\text{-}1)$$

$$\begin{bmatrix} \dfrac{dS_u}{dt} \\ \dfrac{dS_c}{dt} \end{bmatrix} = \begin{bmatrix} M(q) & 0 \\ 1 & 0 \end{bmatrix}^{-1} \begin{bmatrix} \tau_d - B(q)[\dot{q},\dot{q}] - C(q)[\dot{q}^2] - D(q)[\dot{q}] \\ -F(q,\dot{q}) \end{bmatrix} \begin{bmatrix} \dot{q} \\ 0 \end{bmatrix}. \qquad (13\text{-}2)$$

Parameter's description of equations (13-1) *and* (13-2):

In equations (13-1) and (13-2) the following designations are used:

$q_j(t) = [\alpha, \gamma, \beta, \theta w, \theta 1, \theta 2, \theta 3, \theta 4, \eta]$, j = 1,2,…9 are *generalized coordinates* (see Fig.13-1);

λ - Lagrangian multipliers;

$M(q)$ - an inertial acceleration's matrix;

$E(q)$ - closed-links mechanism constraints ;

(Copia modificata per una migliore consultazione on-line)

$B(q,\dot{q})$, $C(q,\dot{q})$ - matrices including Coriolis and centrifugal terms;

$G(q)$ - a 9-dimensional vector of gravity terms;

$D(q,\dot{q})$ - a 9-dimensional vector of viscous friction coefficients;

$F(q,\dot{q})$ - a 9-dimensional vector of friction forces terms;

τ_i - a 9-dimensional vector of torque $\tau_i = [0,0,0,0,0,0,0,0,\tau_{(\eta)3}]$;

In Eq. (13-2) S_u is the entropy of the robot unicycle's motion and S_c is the entropy of both controllers, τ_d are dissipative parts of the control torque (for the PD-controller the dissipative part is described by $k_i(\dot{\gamma},\dot{\beta})$).

For the stability analysis and the computer simulation of the robotic unicycle's dynamic behavior Eqs (13-1) are rewritten in the traditional form of ordinary differential equations as $\dot{q}_i = \varphi_i(q_i,\tau,t)$.

Lyapunov Function and Thermodynamic Conditions
for Stability of the Robotic Unicycle

The analysis of the robot model stability as essential nonlinear system is based on the interrelation between Lyapunov and entropy production rate functions. We will use the new approach for the definition of Lyapunov function [31]. The Lyapunov function for the system (13-1) is defined as $V = \frac{1}{2}(\sum_{i=1}^{6} q_i^2 + S^2)$, where $S = S_u - S_c$ and $q_i = (\alpha,\gamma,\beta,\dot{\alpha},\dot{\gamma},\dot{\beta})$. Here we introduce the following relation between Lyapunov function and entropy production for an *open* system like a unicycle

$$\frac{dV}{dt} = \sum_{i=1}^{6} q_i\,\varphi_i(q_i,\tau,t) + (S_u - S_c)(\frac{dS_u}{dt} - \frac{dS_c}{dt}) < 0. \qquad (13\text{-}3)$$

From Eq. (13-3) the necessary and sufficient conditions for the Lyapunov stability of a robotic unicycle is expressed as follows:

$$\sum_{i=1}^{6} q_i\,\varphi_i(q_i,\tau,t) < (S_u - S_c)(\frac{dS_c}{dt} - \frac{dS_u}{dt}), \qquad \frac{dS_c}{dt} > \frac{dS_u}{dt}. \qquad (13\text{-}4)$$

From Eq. (13-4) the stability measure for the robotic unicycle can be obtained by computing the minimum entropy production rate of the system and the controllers.

Remark 3. From the qualitative physics the robotic unicycle has two unstable states: 1) a local unstable kinematic equilibrium in lateral plane (angle of rolling γ); and 2) a global unstable dynamic state in longitudinal plane (angle of pitching β).

These two correlated states have to be controlled with two fuzzy controllers [31]. This is necessary and sufficient condition for the improvement of the control stability of our robotic unicycle.

Two fuzzy controllers realize the control of an energy transfer with the minimum entropy production from lateral to the longitudinal planes using the dynamic of non-linear cross braces in our robotic unicycle model (a compensation of the energy transfer from unstable dynamic motion "in large" to longitudinal plane according to Eq. (13-3)).

The fuzzy controller in the lateral plane executes a role of human riding by organizing a special *parametric* excitation in the non-linear cross braces. These parametric excitations generate some energy amount that compensates the transfer energy from the longitudinal plane with the unstable state (thus, the unstable state "in small" compensates the unstable state "in large"). A stable motion of the robotic unicycle model is the result of nonlinear control (at the intelligent level) of the correlated energy transfer between two unstable virtual states. This is the physical point of view on the human riding behavior based on intuition and instinct mechanisms.

So, two adaptive fuzzy controllers mentioned above realize a self-organized process of the control stability on robotic unicycle using intuition and instinct schemes.

Fuzzy Intelligent Control of the Extension-Cableless Robotic Unicycle with Soft Computing Based on GA and FNN

In the intelligent control system of the unicycle two PD-controllers are adopted: one is for the symmetric rotor, and the other is for the closed link mechanisms. The control torque to the symmetric rotor is given as,

$$\tau_\eta = kp_2 \times k_3 \times \gamma + kd_2 \times k_4 \times \dot{\gamma}, \qquad (13\text{-}5)$$

where, τ_η is the torque to the rotor; kp_2 and kd_2 are constant feedback gains; k_3 and k_4 are fuzzy schedulers changed in [0,1] with FNN. The control torque applied to the link 2 and link 4 are given as,

$$\tau_{\theta 2} = -\tau_{\theta 4} = -kp_1 \times k_1 \times \beta - kd_1 \times k_2 \times \dot{\beta}, \qquad (13\text{-}6)$$

where, $\tau_{\theta 2}$ and $\tau_{\theta 4}$ are the torques to the link 2 and 4 respectively, kp_1 and kd_1 are constant feedback gains; k_1 and k_2 are fuzzy values changed in [0,1] with FNN.

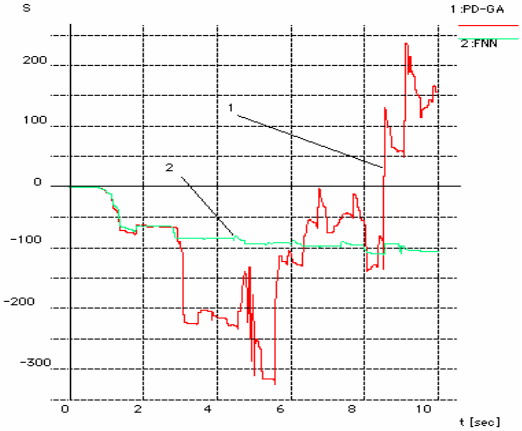
Remark 4. The biomechanical analysis of a posture stability of the unicycle shows that a PD-control represents the minimum complexity necessary to the stable posture control. The component *P* (proportional) contains anti-gravitational forces and compensates the position errors. The component *D* (derivative) contains an anti-Coriolis compensation and provides also some kind of damping actions. The parameters (kp_1, kp_2) may be interpreted as a stiffness (spring constant) arising from passive and active muscular forces, whereas (kd_1, kd_2) might be compared with a viscous damping, as obtained with a wheel dashpot.

The fuzzy tuning rules for k_1, k_2, k_3 and k_4 are formed by the learning system of the FNN by using teaching patterns from GA.

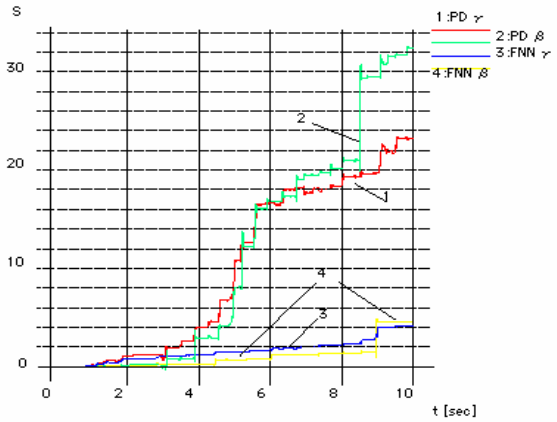
Simulation Results

By using the proposed control methods we are able to conduct computer simulations. In the first case GA simulates an intuition mechanism of choosing the optimal structure of the PD-controller, using the capacity of the fitness function, which is the measure of the entropy production rate.

(Copia modificata per una migliore consultazione on-line)



Temporal thermodynamic behavior of the Yaw.



Temporal thermodynamic behavior of the Pitch & Roll.

Figure 13-5. Simulation result of thermodynamic behavior of the control system in the Yaw, Pitch &Roll.

Figures 13-5 shows simulation results of the temporal thermodynamic behavior of the robotic unicycle. We have calculated the entropy production rate $\frac{dS_\alpha}{dt}$ (yawing angle), $\frac{dS_\beta}{dt}$ (pitching angle), and $\frac{dS_\gamma}{dt}$ (rolling angle).

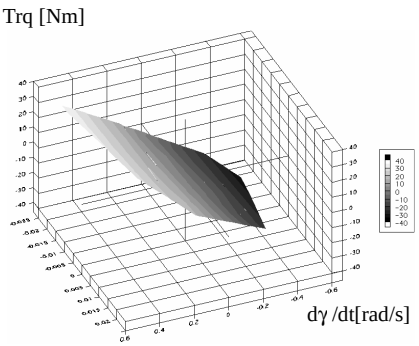
From the simulation results we obtain that the relation $\frac{dS_c}{dt} > \frac{dS_u}{dt}$ (Eq. (13-4)) is true and the GA realizes the search of optimal parameters for the PD controllers with a simple structure using the principle of minimum entropy production rate. The FNN controller offers a more flexible structure of controllers with smaller torques, and the learning process produces less entropy (Fig.13-5). However, a time necessary for achieving an optimal control with the learning process on FNN (instinct) is larger the time necessary for the global search on GA (intuition).

These results have confirmed the possibility of finding of the optimum decision in case of soft computing application.

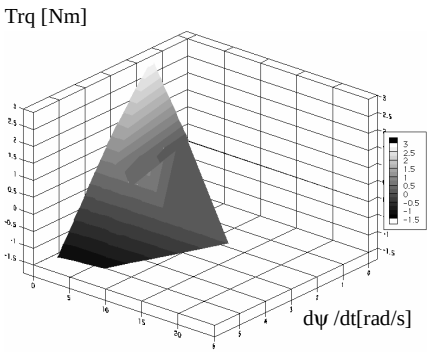
The results show also that the entropy production rate in the rotor control system is bigger then the entropy production rate in the links control system. This is experimental proving of results described below.

Simulation Results for the Fuzzy Gain PD-Controllers

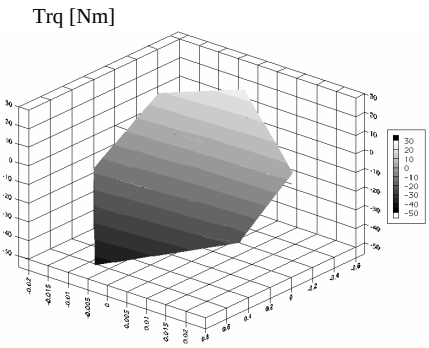
In this subsection, two fuzzy gain schedule PD-controllers are considered. One is for the rotor and the other is for the closed-link mechanisms. The torque to the rotor is the same as that in Eq. (13-5) and the torque to links 2 and 4 is given in Eq. (13-6).



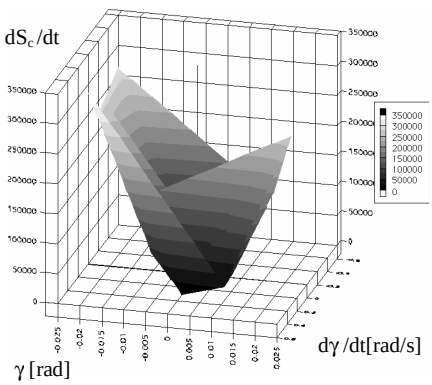
a) Simulation Result of rotor torque with PD-GA controller



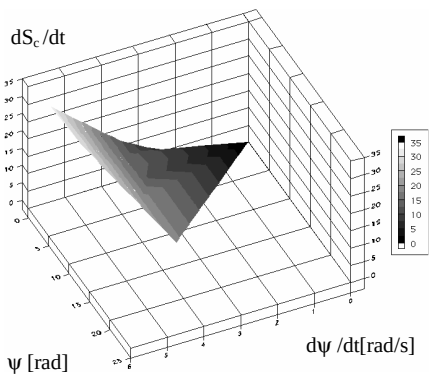
b) Simulation Result of wheel torque with PD-GA controller



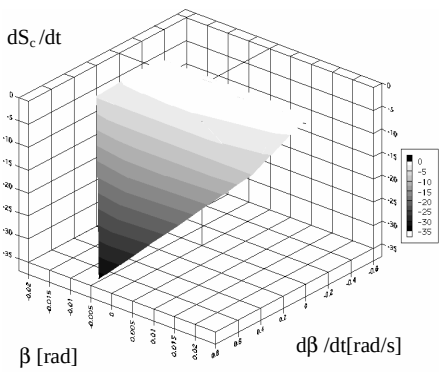
c) Simulation Result of pitch torque with PD-GA controller



d) Simulation Result of entropy production rate of rotor PD-GA controller



e) Simulation Result of entropy production rate of wheel PD-GA controller



c) Simulation Result of entropy production rate of pitch PD-GA controller

Figure13- 6. Simulation results.

Figures 13-6 show the 3D simulation results of the mechanical and thermodynamic behavior of the robotic unicycle with PD-GA-controller for the "simple"

unicycle model. The initial posture is set by the operator, who removes his hand immediately after the wheel starts to go forward.

Remark 5. Comparing the experimental results reported in [32] and in this chapter we find that it is much easier to achieve the robot's posture stability with the fuzzy gain schedule PD-controllers. If the initial posture of the robotic unicycle is near the ideal stable posture (□ = 0.0 rad and □ = 0.0 rad), the postural stability can be obtained in almost all trials. However, the postural stability with two fuzzy gain schedule PD-controllers can be achieved even if the posture is randomly disturbed to some extent by the ground condition.

In Fig.13-7 the temporal behavior of β, γ, α - angles and fuzzy gains k_1, k_2, k_3 and k_4 are shown.

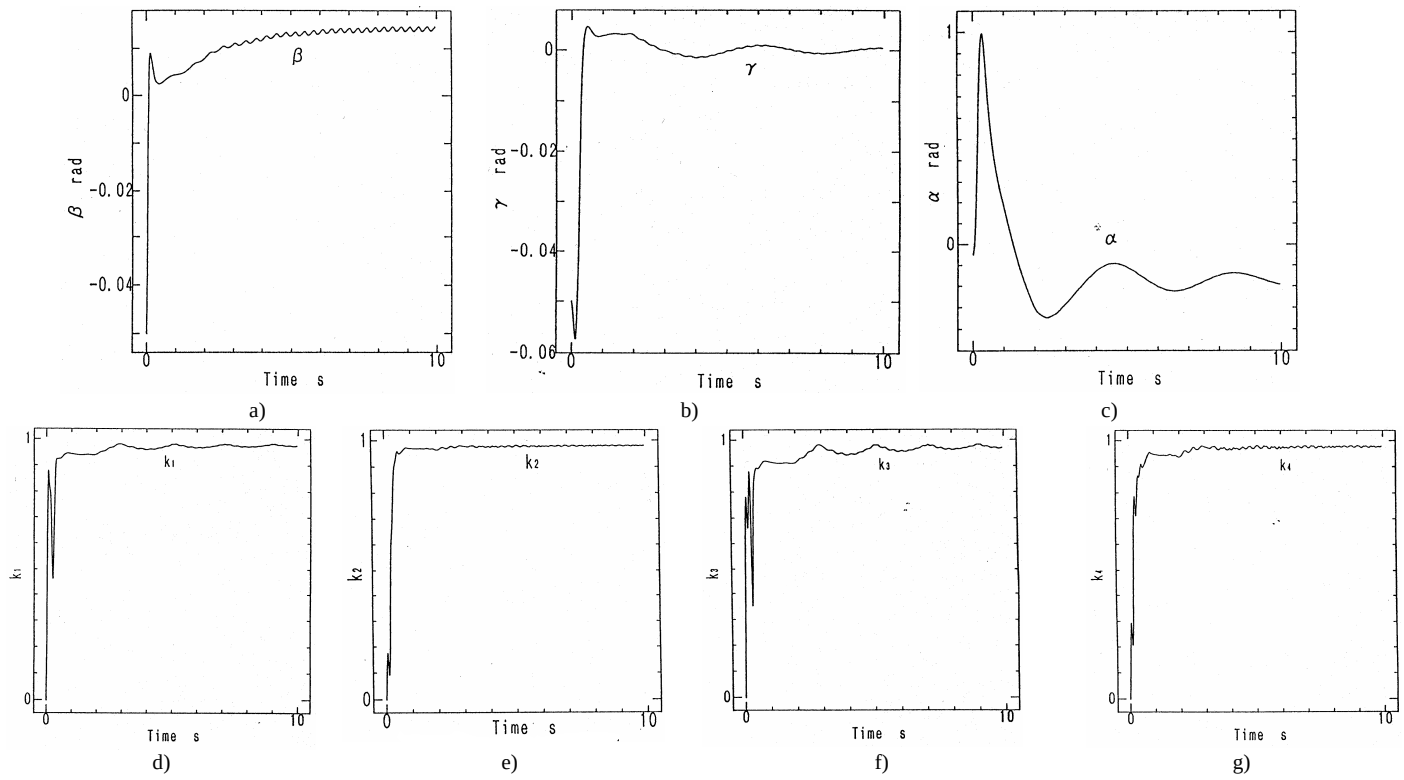


Figure 13-7. Simulation results for the two fuzzy gain schedules PD controllers: a) pitch angle; b) roll angle; c) yaw angle; d-g) k_1 , k_2, k_3 and k_4.

The simulation results show that the proposed control methods are quite effective for achieving postural stability, which can be maintained over a fairly long time period. The simulation results also show that the roll angle can be stabilized around zero efficiently and the pitch angle can also be stabilized around a small positive value. This indicates that it is necessary to stabilize the pitch angle not at zero but at a small positive value in order to maintain postural stability. The comparison with the results above indicates that the postural stability is not influenced to a large extent by the robot's initial posture. Furthermore the simulation results show that the postural stability can be achieved if both the initial pitch angle |□| and the initial roll angle |□|are less than ~ 0.1 rad.

Appendix

The manufactured new extension-cableless robotic unicycle shown in Fig.13-1 is composed of a wheel with two cranks, a main body, an overhead rotor and two closed links on both sides of the wheel. The closed link mechanism is used for a longitudinal stability and the symmetric rotor is used for a lateral stability. The wheel is driven through only the close links mechanism's motors.

The rotor is driven by a harmonic drive motor (for old - 34W, for new - 60W) installed on the body. The left and right closed links are driven directly by harmonic drive motors (for old - 20.3W, for new - 60W) on links 2 and 4 (Fig.13-1). The links motors are the same for the symmetric geometrical structure and balance of the robot.

A 32-bit personal computer is used for the system controller. The wheel, symmetric rotor and closed link mechanisms are driven by torque controlled motors with a software-servo control. All the control programs are written in C-language.

As shown in Fig.13-1, there are three rate-gyro sensors (sensor A, B and C) mounted on the three principal axes of the robot's body for measuring the angular velocities of the body inclination in the pitch, roll and yaw directions. The resolution of the angular velocity of the sensors is 0,1 degree/second. An optical rotary encoder (500 pulses/revolution) is installed on each servomotor to detect the rotation angle caused by the rotation of the servomotors. The usage of the coordinates defined in Fig.13-1,b, enables us to calculate the robot's posture or Euler's angle (□□□) related to the global reference coordinates measured from the angular velocity $\square_x$, $\square_y$, and $\square_z$ by the three rate-gyro sensors and $\square_x$ is the angular velocity related to Px_6 , $\square_y$ is the angular velocity related to Py_6, $\square_z$ is the angular velocity related to Pz_6 .

Using this kind of small rate-gyro sensors there is a drift on the output due to the time and the change of temperature. The drift may yield an unfavorable influence on the calculation of the postural angles in the experiment. Thus we selected the sensors with the smallest drift. The experiments are conducted within 8 seconds, because the drift of the sensor's output within is not so big this time.

(Copia modificata per una migliore consultazione on-line)

AI Bibliography

References to PART 1 and *PART* 2:

1. Turing A.M. (1950) Computing machinery and intelligence. *Mind*, 59, 433-460.
2. Oxford dictionary of computing (1991)
3. Kasabov N.K. (1996) *Foundations of Neural Networks, Fuzzy Systems, and Knowledge Engineering*, MIT Press, Cambridge, Massachusetts; Smolensky P. (1988) On the proper treatment of connectionism, *Behavioral and Brain Sciences*, Vol.2, N 1.
4. Rai R. (1996) The Challenge of Artificial Intelligence, *Computer*, Vol.29, N 10, 86-98.
5. Newell A. (1980) Physical symbol systems. *Cognitive Science*, 4, 135-183.
6. Minsky M. (1968) *Semantic Information processing*, MIT Press, Cambridge.
7. *Computer Models of Thought and Language*, (1973) (Ed. R.Schank, K.Colby),

   W.F. Freeman and Co., San Francisco.
8. Baidyn V. and Semenova E. (1989) *FRL – Frame Representation Language*, MEI Press, Moscow.
9. Lupina N.V, Slepchenko A.N., Ulyanov S.V. and Shakhnasarov M.M, (1993) A Hybrid Expert System with Depth Representation of Knowledge for the Design and Diagnostics of Bioengineering Products, *Journal of Computer and System Sciences International*, 31 (3). 74-95.
10. Zadeh L.A. (1975) The concept of a linguistic variable and its application to approximate reasoning, *Information Sciences*, 8, 199-249; 9, 43-80.
11. Tanaka K. (1991) *An Introduction to Fuzzy Logic for practical Applications*, Springer Verlag.
12. Dubois D. and Prade H. (1980) Fuzzy Sets and Systems and Applications, Academic Press, New York.; (1987) Fuzzy numbers: an overview, In: Bezdek JC (Ed) *Analysis of Fuzzy Information*, Vol 1. CRC Press, 3-39.
13. Jang J-S.R. and Sun C-T. (1995) Neuro-Fuzzy Modeling and Control, *Proc. of IEEE*, Vol. 83, N 3, 378-405.
14. Batyrshin I. and Kaynak O. (1999) Parametric Classes of Generalized Conjunction and Disjunction Operations for Fuzzy Modeling, IEEE Transactions on *Fuzzy Systems*, Vol. 7, N 5, 586-597
15. Kosko B. (1992) *Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence*, Prentice Hall, Englewood Cliffs, NJ.; Wang L-X. (1992) Fuzzy systems are universal approximators, In: Proc. *IEEE Int.Conf. Fuzzy Systems*, San Diego, CA.
16. Wang Li-Xin (1994) *Adaptive fuzzy systems and control*. PTR Prentice Hall, Englewood Cliffs, NJ.
17. Saffiotti A. (1997) The uses of fuzzy logic in autonomous robot navigation, *Soft Computing*, Springer,Vol.1, No 4,180-197
18. Ulyanov S.V., Watanabe S., Ulyanov V.S., Yamafuji K., Litvintseva L.V. and Rizzotto G. (1998) Soft computing for the intelligent robust control of a robotic unicycle with a new physical measure for mechanical controllability, *Soft Computing*,Vol.2, N 2, 73-89.
19. Holland J.H. (1975) *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor.
20. Wasserman P. (1993) *Advanced Methods in Neural Computing*. Van Nostrand Reinhold, New York.
21. Goldberg D.E. (1989) *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA.
22. Nawa N.E. and Furuhashi T. (1999) Fuzzy System parameters Discovery by bacterial Evolutionary Algorithm, IEEE Transactions on *Fuzzy Systems*, vol.7, No.5, 608-617.
23. Wong C-C. and Fan C-S. (1999) Rule mapping fuzzy controller design, *Fussy sets and Systems*, 108, 253-261.
24. Hebb D.O. (1949) *The Organization of Behavior*, New York: Wiley.
25. Patterson D.W. (1996), Artificial neural networks, theory and applications, Prentice Hall, Singapore.
26. Jang J-S.R. , Sun C-T., Mizutani E. (1996) *Neuro-Fuzzy Modeling and Soft Computing (A computational approach to learning and machine intelligence*), Matlab Curriculum series.
27. Rosenblatt R. (1962) *Principles of Neurodynamics*, Spartan Books, New York; Minsky M. and Papert S. (1969) *Perceptrons: An Introduction to Computational Geometry*, MIT Press, Cambridge, Mass.
28. Rumelhart D.E., Hinton G.E.and Williams R.J (1986) Learning representations by backpropagating errors, *Nature*, Vol.323, 533-536; Freeman J.A. (1994) *Back propagation and its variants*, Addison-Wesley, Reading, MA.
29. Anil K.J, Jianchang M., Mohiuddin K.M. (1996) Artificial Neural Networks: A Tutorial, *Computer*, March-96.
30. Ulyanov S.V., Sheng Z.Q. and Yamafuji K. (1995), "Fuzzy Intelligent control of robotic unicycle: A New benchmark in nonlinear mechanics", *Intern. Conf. on Recent Advanced Mechatronics*, Istanbul, Turkey, Vol. 2, 704 – 70.
31. Ulyanov S.V., Yamafuji K., Ulyanov V.S., et.al.(1999), Computational intelligence for robust control algorithms of complex dynamic systems with minimum entropy production. Part1: simulation of entropy-like dynamic behaviour and Lyapunov stability, *Journal of Advanced Computational Intelligence*, Vol.3, No.2,82-98; Ulyanov V.S., Yamafuji K., Ulyanov S.V. and Tanaka K. (1999), Computational intelligence with new physical controllability measure for robust control algorithms of extension-cableless robotic unicycle, *Journal of Advanced Computational Intelligence*, Vol.3, No.2,136-147.
32. Sheng Z.Q., Yamafuji K. and Ulyanov S.V.(1996) Study on the stability and motion control of a unicycle. Parts 3,4,5, *JSME International Journal*, Vol. 39, No. 3,560-568; 569-576; and (1996) *Journal of Robotics & Mechatronics*, Vol. 8, No. 6, 571-579.

Additional Reading

For Part1:

33. Newell A. (1982) The knowledge level. *Artificial Intelligence*, 18, 87-127.
34. Newell A. and Simon H.A.(1972) *Human Problem Solving*. Prentice-Hall, Englewood Cliffs, New Jersey.
35. Nilsson N.J. (1971) *Problem Solving Methods in Artificial Intelligence*, McGraw-Hill, N.Y.
36. Sowa J.F. (1984) *Conceptual Structures: Information Processing in Mind and Machine*, Addison-Wesley, Reading, Mass.
37. Hoffmann A. (1998) *Paradigms of Artificial intelligence*, Springer-Verlag.
38. Pedrych W. (1998), *Computational Intelligence: An Introduction*, CRC Press, Boca Raton, New York.
38. Medsker L.R. (1995) *Hybrid Intelligent Systems*, Boston: Kluwer Academic Publ.
39. Eriksson H. (1996) Expert Systems as Knowledge Servers, *IEEE Expert*, Vol.11 No 3, 14-19.

For Part 2:

40. Klir G.J. and Folger T.A. (1996) *Fuzzy sets, uncertainty and information*, Prentice-Hall, Englewood Cliffs, New Jersey.
41. Klir G,J, Pan Y. (1998) Constrained fuzzy arithmetic: Basic questions and some answers, *Soft Computing*, vol.2, N 2, 100-108
42. Van Rooij J.F, Jain L.C. and Johnson R.P. (1996) *Neural Network Training Using Genetic Algorithms*, series "Machine Perception, Artificial intelligence", Vol.26, World Sceintific.
43. *Fuzzy systems. Modeling and Control*. (Eds. H.T. Nguen and M.Sugeno) (1998) Kluwer Academic Publ.
44. Cai Zi-Xing (1997) *Intelligent control: principles, techniques and applications*, Series in intelligent control and intelligent automation (Ed. F.-Y. Wang), Vol.7, World Scientific.