SREENIVASA INSTITUTE OF TECHNOLOGY AND MANAGEMENT STUDIES
(Autonomous)
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LECTURE NOTES

23CSE354D — SOFT COMPUTING

(Professional Elective - I)
III B.Tech – V Semester

Covering:
UNIT I — Introduction to Soft Computing
UNIT II — Fuzzy Systems
UNIT III — Fuzzy Decision Making and Particle Swarm Optimization
UNIT IV — Genetic Algorithms
UNIT V — Rough Sets and Integration of Soft Computing Techniques

Prepared by: A. Elavarkuzhali

UNIT – I
INTRODUCTION TO SOFT COMPUTING

1.1 Introduction

For more than five decades, conventional or 'hard' computing has been the dominant approach used by engineers and scientists to model and solve real-world problems. Hard computing requires a precisely stated analytical model, produces precise, guaranteed and repeatable results, and generally demands a large amount of computational time to arrive at a solution. However, a large class of real-world problems — such as recognising a handwritten signature, controlling the temperature of a room, predicting the behaviour of the stock market, or driving a car through traffic — are inherently imprecise, uncertain, and difficult to model mathematically. Human beings routinely solve such problems using approximate reasoning, learning from experience, and common sense, rather than exact mathematical formulae.

Soft Computing is a consortium of computational techniques that attempts to imitate this human capability of dealing with imprecision, uncertainty, partial truth and approximation in order to achieve tractability, robustness, low cost, and better rapport with reality. The term was coined by Professor Lotfi A. Zadeh, the founder of Fuzzy Set Theory, in the early 1990s at the University of California, Berkeley. Zadeh described soft computing as a collection of methodologies that work synergistically, rather than competitively, to provide flexible information-processing capabilities for handling real-life ambiguous situations.

It is instructive to note that Zadeh's original motivation stemmed from what he termed the 'principle of incompatibility' — as the complexity of a system increases, our ability to make precise and yet significant statements about its behaviour diminishes, until a threshold is reached beyond which precision and significance become mutually exclusive. Soft computing was proposed precisely to operate beyond this threshold, where classical hard-computing methods become intractable or simply inapplicable.

This unit introduces the philosophy of soft computing, contrasts it with traditional hard computing, describes its principal constituent technologies, discusses evolutionary computing as one of its most important paradigms in considerable depth, and surveys the characteristics, recent trends, and application areas of soft computing.

1.2 What is Soft Computing?

Soft Computing (SC) is defined as an integration of several computing paradigms including Fuzzy Logic (FL), Neural Networks (NN), Genetic Algorithms/Evolutionary Computing (GA/EC), Probabilistic Reasoning (PR), and Rough Set Theory (RS), that are combined synergistically rather than used independently. Unlike conventional (hard) computing, soft computing is tolerant of imprecision, uncertainty, partial truth, and approximation.

- Soft computing is not a single technique but a partnership of complementary methodologies.
- The guiding principle is: exploit the tolerance for imprecision, uncertainty, partial truth and approximation to achieve tractability, robustness, low solution cost, and better rapport with reality.
- Its model of intelligence is human-centred; it draws inspiration from how the human mind reasons and learns under uncertainty.
- It emphasises that in dealing with real-world problems, an approximate but computationally cheap and quick solution is often more useful than an exact but expensive one.

> *"Soft computing differs from conventional (hard) computing in that, unlike hard computing, it is tolerant of imprecision, uncertainty, partial truth, and approximation." — Lotfi A. Zadeh*

It is important to appreciate that soft computing is best regarded as a foundation for the emerging field of 'computational intelligence', and its role continues to expand as the amount of imprecise, uncertain, and unstructured data generated by modern systems (sensors, social media, IoT devices) keeps growing. Whereas conventional artificial intelligence (symbolic AI) largely relies on formal logic and explicit symbolic knowledge representation, soft computing relies on numerical, sub-symbolic representations and learning from data, making it particularly effective for perceptual, pattern-recognition, and control tasks.

1.3 Hard Computing versus Soft Computing

The conventional approach to computing, referred to as hard computing, is based on precision, certainty, and rigor. It requires a precisely stated analytical model and often a great deal of computation time. Hard computing is appropriate when the underlying problem is well defined mathematically and does not involve ambiguity. Soft computing, by contrast, is designed for problems where the model is not fully known, data may be noisy or incomplete, and an

approximate solution is acceptable, even preferable, because it can be obtained much faster and more robustly.

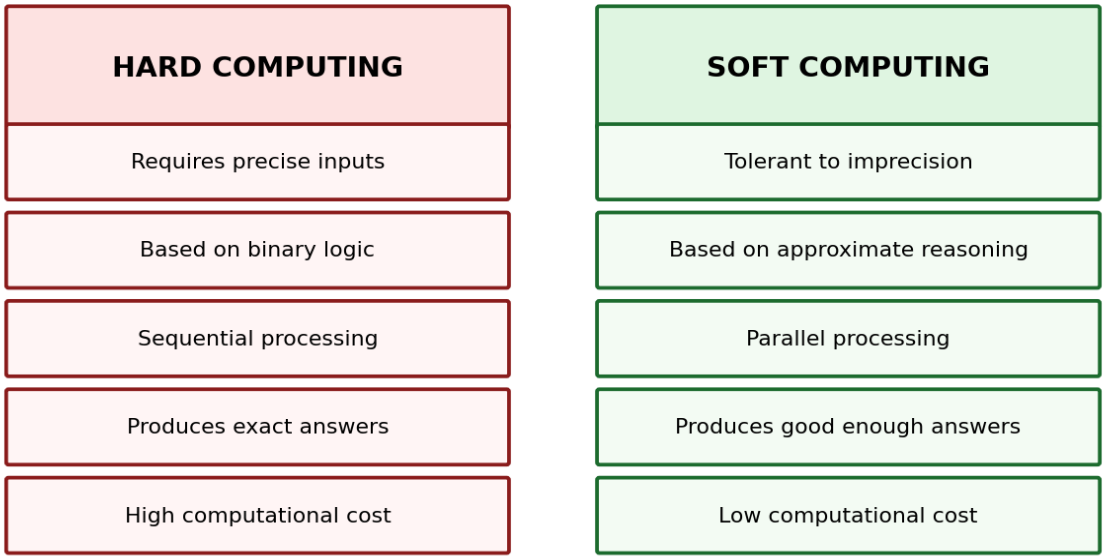

Fig. 1.1 – Hard Computing versus Soft Computing

1.3.1 Detailed Comparison

Basis	Hard Computing	Soft Computing
Precision	Requires exact, precisely stated analytical model	Tolerant of imprecision and approximate models
Logic base	Two-valued (Boolean) logic — true/false	Multi-valued / fuzzy logic — degrees of truth
Nature	Sequential, deterministic computation	Parallel, distributed and stochastic computation
Output	Exact, guaranteed, repeatable result	Approximate, 'good enough' result
Basis of solution	Mathematical/analytical rigor	Learning, adaptation and biological inspiration
Cost	High computational and implementation cost	Low cost, faster, more economical
Adaptability	Rigid — cannot easily learn or adapt	Adaptive — learns from data/experience
Failure tolerance	Fails when input is noisy or incomplete	Robust — degrades gracefully with noisy/incomplete data
Examples of techniques	Numerical analysis, classical control theory, symbolic AI	Fuzzy logic, neural networks, genetic algorithms, rough sets

It is important to note that soft computing does not replace hard computing; rather it is complementary to it. In most practical, hybrid intelligent systems, hard-computing modules (for numerical computation) and soft-computing modules (for handling uncertainty and

learning) work together. For example, in a fuzzy logic controller, the fuzzification and defuzzification stages internally use precise numerical (hard) computations, while the reasoning stage in between uses soft, approximate fuzzy inference.

1.3.2 Soft Computing versus Conventional (Symbolic) Artificial Intelligence

It is also useful to distinguish soft computing from conventional, symbolic artificial intelligence (sometimes called 'hard AI' or Good Old-Fashioned AI, GOFAI). Symbolic AI represents knowledge using explicit symbols and rules (e.g. expert systems using crisp production rules, predicate logic, semantic networks) and performs reasoning through logical inference. Soft computing, on the other hand, favours numeric, sub-symbolic representations (membership functions, connection weights, chromosomes) and favours learning and search over explicit symbolic manipulation. Symbolic AI tends to be brittle in the presence of noisy or incomplete knowledge, whereas soft computing techniques are specifically designed to be robust under such conditions.

1.4 Constituents (Methods) of Soft Computing

Soft computing is best viewed as a 'consortium' of methodologies that work in a complementary rather than competitive manner. The principal constituents are Fuzzy Logic, Artificial Neural Networks, Genetic Algorithms/Evolutionary Computing, Probabilistic Reasoning, and Rough Set Theory. Each provides a distinctive and flexible information-processing capability for handling real-life situations.

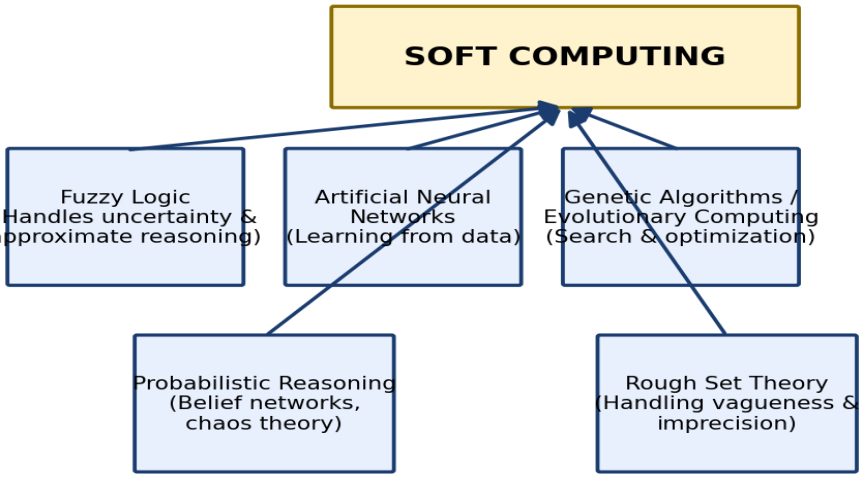

Fig. 1.2 – Principal Constituents of Soft Computing

1.4.1 Fuzzy Logic (FL)

Fuzzy Logic, introduced by Lotfi Zadeh in 1965, provides a mathematical framework to capture the uncertainties associated with human cognitive processes such as thinking and reasoning. Unlike classical (crisp) set theory, where an element either fully belongs or does not belong to a set, fuzzy set theory allows partial membership, expressed by a membership function taking values in the range [0, 1]. Fuzzy logic is primarily concerned with imprecision and approximate reasoning; it provides the decision-making component in most soft computing systems and is discussed at length in Unit II of this course.

1.4.2 Artificial Neural Networks (ANN)

Artificial Neural Networks are computing systems inspired by the biological neural networks found in the human brain. An ANN consists of a large number of simple, interconnected processing elements called neurons, organised in layers (input, hidden and output). Each connection has an associated weight that is adjusted during a learning process (such as the backpropagation algorithm for multilayer feed-forward networks). ANNs excel at learning from examples (data), recognising patterns, and generalising to unseen inputs, even in the presence of noise. Their primary contribution to soft computing is the capability of learning and adaptation.

1.4.2.1 The Perceptron — A Simple Neuron Model

The simplest artificial neuron model, the perceptron, computes a weighted sum of its inputs, adds a bias term, and passes the result through an activation function to produce an output: $y = f(\Sigma w_i x_i + b)$. During training, the weights w_i are iteratively adjusted, typically using the perceptron learning rule $w_i \leftarrow w_i + \eta(t - y)x_i$ (where t is the target/desired output and η is the learning rate), so that the network's output progressively approaches the desired output for each training example. Multiple such neurons, arranged in interconnected layers and trained using the backpropagation algorithm, form a Multi-Layer Perceptron (MLP) — the most widely used type of feed-forward neural network.

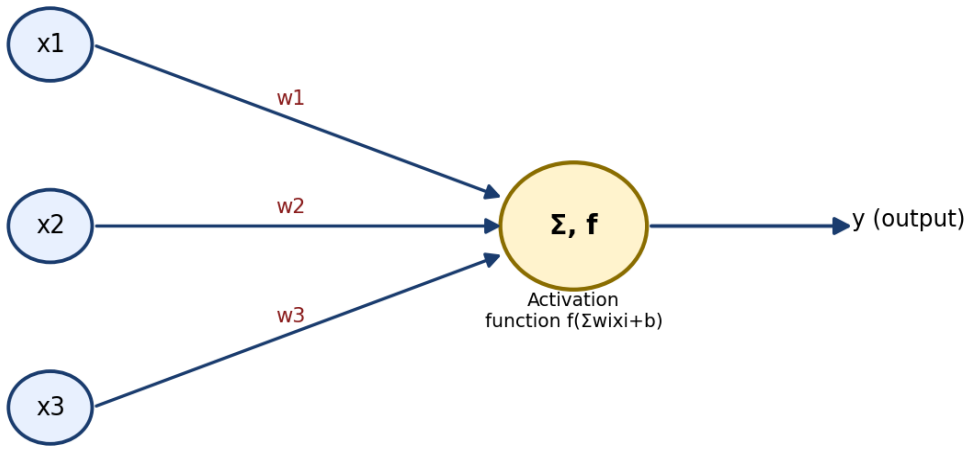

Fig. 1.2(a) – A Simple Perceptron (Artificial Neuron) Model

1.4.3 Genetic Algorithms / Evolutionary Computing (GA/EC)

Genetic Algorithms, developed by John Holland in the 1970s, are stochastic, population-based search and optimisation techniques based on the mechanics of natural selection and natural genetics — 'survival of the fittest'. They are part of the broader field known as Evolutionary Computing, discussed in detail in Section 1.5 and again in Unit IV of this course. Their contribution to soft computing is the capability of systematic, randomised, yet directed, search and optimisation, particularly useful for problems with large, complex, multi-modal search spaces where gradient-based methods fail.

1.4.4 Probabilistic Reasoning (PR)

Probabilistic Reasoning encompasses techniques such as belief networks (Bayesian networks), Dempster-Shafer theory of evidence, chaos theory, and parts of learning theory, which are used

to deal with uncertainty arising from randomness rather than vagueness. While fuzzy logic deals with vagueness (ambiguity in meaning — e.g. 'is 35°C hot?'), probability deals with likelihood (chance of occurrence — e.g. 'what is the chance it will rain tomorrow?'). Probabilistic reasoning is used in soft computing systems to reason under uncertainty when statistical information about a domain is available, and is frequently combined with fuzzy and neural approaches in hybrid systems.

A representative tool of probabilistic reasoning is Bayes' theorem, $P(H|E) = [P(E|H) \times P(H)] / P(E)$, which allows the probability of a hypothesis H to be updated in light of new evidence E. For example, if a certain disease occurs in 1% of a population ($P(H) = 0.01$), a diagnostic test correctly detects the disease 95% of the time when present ($P(E|H) = 0.95$), and it also gives a false positive 5% of the time in healthy people, then, using the theorem, the probability that a patient actually has the disease given a positive test result works out to be far lower than 95% — illustrating how naive interpretation of test accuracy can be misleading, and why formal probabilistic reasoning is essential in decision support systems.

1.4.5 Rough Set Theory (RS)

Rough Set Theory, proposed by Zdzisław Pawlak in 1982, is a mathematical approach to handle vagueness and uncertainty arising from insufficient or incomplete information (granularity of data) rather than fuzziness. A rough set is characterised by a pair of precise (crisp) sets, called the lower and upper approximations, that bracket the imprecise concept. Rough sets are especially useful in data mining, feature selection, and rule induction, and are studied in depth in Unit V of this course.

1.4.6 Comparative Summary of the Constituents

Technique	Deals primarily with	Key strength	Learning ability
Fuzzy Logic	Vagueness / imprecision	Human-interpretable reasoning	Limited (unless hybridised)
Neural Networks	Pattern recognition, non-linearity	Learning from data	High
Genetic Algorithms	Search and optimisation	Global, population-based search	Adaptive via evolution
Probabilistic Reasoning	Randomness / statistical uncertainty	Formal treatment of likelihood	Depends on model
Rough Sets	Data granularity / vagueness from incompleteness	Rule induction, attribute reduction	Limited (data-driven)

1.4.7 Principles of Hybrid Soft-Computing System Design

Because the true power of soft computing lies in combining its constituent technologies rather than using any one in isolation, it is useful to note the general principles that guide the design of hybrid soft-computing systems. First, each constituent technology should be assigned to the sub-task for which it is best suited — for instance, using a neural network for the perceptual/pattern-recognition sub-task, and a fuzzy system for the reasoning/decision sub-task that follows it. Second, the interfaces between constituent technologies should be designed so that the output representation of one component is directly usable as the input representation of the next (e.g. a neural network's numeric output can be directly fuzzified as the input to a subsequent fuzzy inference stage). Third, wherever one technology has a well-known weakness (such as a genetic algorithm's relatively slow fine-tuning of continuous parameters, or a neural network's need for large labelled training datasets), a complementary technology should be introduced specifically to compensate for that weakness (such as using local gradient-based fine-tuning after a GA has found a good starting region, or using fuzzy expert rules to pre-structure a neural network's initial weights). These principles recur throughout the neuro-fuzzy, genetic-fuzzy, and neuro-genetic hybrid architectures introduced in Section 1.7 and studied in greater depth in later units of this course.

1.5 Evolutionary Computing

Evolutionary Computing (EC) is the umbrella term for a family of algorithms that draw inspiration from biological evolution — reproduction, mutation, recombination, and selection. The unifying idea is to maintain a population of candidate solutions, and to iteratively evolve this population, using operators inspired by genetics, so that fitter (better) solutions become more prevalent over successive generations.

1.5.1 Historical Background

- 1859 — Charles Darwin publishes 'On the Origin of Species', introducing the theory of natural selection, the conceptual seed of evolutionary computing.
- 1960s — Rechenberg and Schwefel (Germany) develop Evolution Strategies (ES) for parameter optimisation of engineering devices.
- 1960s — Fogel, Owens and Walsh develop Evolutionary Programming (EP) to evolve finite-state machines.
- 1970s — John Holland (University of Michigan) develops Genetic Algorithms (GA) and publishes 'Adaptation in Natural and Artificial Systems' (1975).

- 1989 — David Goldberg publishes 'Genetic Algorithms in Search, Optimization and Machine Learning', popularising GAs for engineering applications.
- 1990s — John Koza extends GAs to evolve computer programs, giving rise to Genetic Programming (GP).

1.5.2 The Evolutionary Computing Cycle

All evolutionary algorithms share a common cycle consisting of population initialisation, fitness evaluation, selection of parents, application of genetic operators (crossover and mutation) to generate offspring, and replacement of the old population by the new one. This cycle is repeated for many generations until a satisfactory solution is found or a stopping criterion (e.g. a maximum number of generations, or convergence of fitness values) is met.

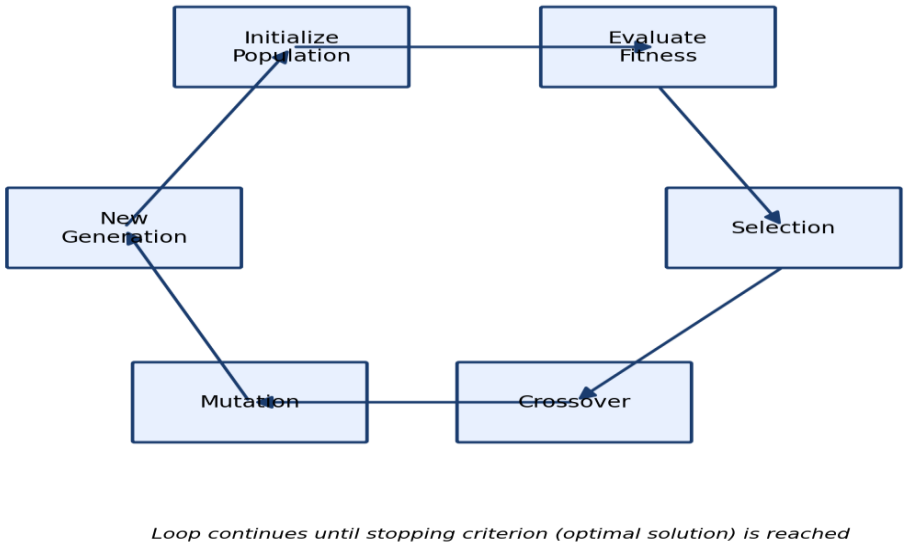

Fig. 1.3 – The Evolutionary Computing (Genetic Algorithm) Cycle

1.5.3 Basic Terminology

Term	Meaning
Chromosome / Individual	An encoded candidate solution to the problem, typically a string of bits, real numbers, or symbols
Gene	A single element/position in a chromosome, representing one decision variable
Allele	The value taken by a gene at a particular locus (e.g. 0 or 1 in a binary chromosome)
Population	A collection of chromosomes present in a given generation
Fitness function	A function that measures how good (fit) a chromosome is as a solution to the problem

Term	Meaning
Selection	The process of choosing fitter chromosomes to become parents of the next generation
Crossover (recombination)	An operator that combines genetic material of two parents to produce offspring
Mutation	An operator that randomly alters a gene value to maintain genetic diversity
Generation	One complete iteration of the evolutionary cycle
Elitism	A strategy of carrying over the best chromosome(s) unchanged into the next generation

1.5.4 Encoding Schemes

The choice of chromosome representation (encoding) is one of the most important design decisions in a genetic algorithm, since it determines the search space that the crossover and mutation operators explore.

- Binary encoding: Each gene is a bit (0 or 1); the classical representation used in Holland's original GA, well suited to combinatorial problems such as the knapsack problem.
- Real-valued (floating-point) encoding: Each gene is a real number; well suited to continuous parameter-optimisation problems, avoiding the precision-loss inherent in binary discretisation.
- Integer encoding: Each gene is drawn from a finite set of integers; useful for problems with discrete but non-binary alternatives (e.g. scheduling).
- Permutation encoding: The chromosome is a permutation of a set of items; used for ordering problems such as the Travelling Salesman Problem (TSP), where standard crossover must be replaced by specialised operators (e.g. order crossover, OX) to preserve permutation validity.

1.5.5 Steps of a Simple Genetic Algorithm

1. Represent each candidate solution as a chromosome (commonly a binary string).
2. Generate an initial population of chromosomes at random.
3. Evaluate the fitness of every chromosome in the population using the fitness function.
4. Select parent chromosomes using a selection scheme (e.g. roulette-wheel selection, tournament selection, rank selection), giving fitter individuals a higher probability of being selected.

5. Apply the crossover operator, with a certain probability (crossover rate, e.g. 0.6–0.9), to pairs of parents to produce offspring.
6. Apply the mutation operator, with a small probability (mutation rate, e.g. 0.001–0.01), to the offspring to introduce diversity.
7. Form the new generation from the offspring (and possibly some retained parents, i.e. elitism).
8. Repeat steps 3–7 until a stopping criterion is satisfied (fixed number of generations, or no further improvement in the best fitness).

1.5.6 Selection Methods

Selection is the operator that biases the search towards fitter chromosomes by giving them a higher chance of contributing offspring to the next generation. Common selection methods include:

- Roulette-wheel selection: Each chromosome is allocated a slice of a conceptual roulette wheel proportional to its fitness (fitness / total fitness); the wheel is spun to choose parents, so fitter chromosomes have a proportionally higher chance of selection.
- Rank selection: Chromosomes are ranked by fitness and selection probability is assigned based on rank rather than raw fitness value, preventing a single 'super-fit' individual from dominating the population too early.
- Tournament selection: A small subset of chromosomes is chosen at random and the fittest among them is selected as a parent; this process is repeated as many times as parents are needed.
- Elitism: Ensures the best chromosome(s) found so far always survive unchanged into the next generation, preventing loss of the best solution due to the randomness of selection/crossover/mutation.

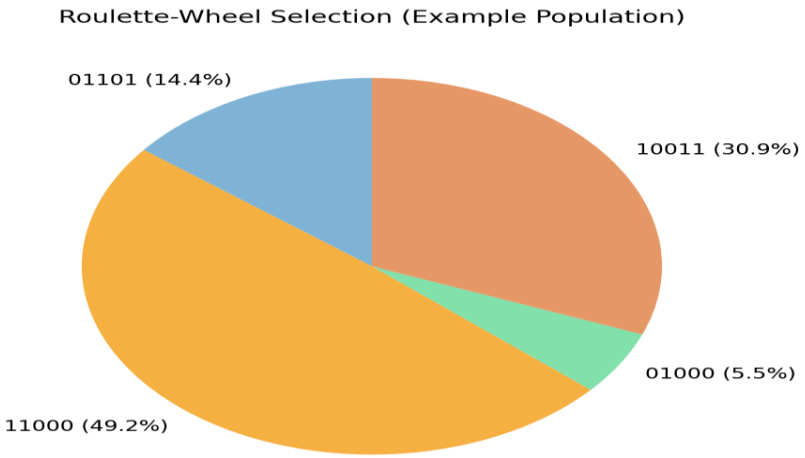

Fig. 1.4 – Roulette-Wheel Selection for the Example Population of Section 1.5.8

1.5.7 Crossover and Mutation Operators

Crossover (recombination) combines genetic material from two parent chromosomes to produce one or more offspring, with the hope that useful 'building blocks' (schemata) from each parent combine to form a fitter offspring. Common crossover types, illustrated in Fig. 1.5, are:

- Single-point crossover: A single crossover point is chosen at random along the length of the chromosome; the segments after that point are swapped between the two parents.
- Two-point crossover: Two crossover points are chosen; the segment between the two points is swapped between the parents.
- Uniform crossover: A random binary mask decides, gene by gene, whether the value is inherited from parent 1 or parent 2, allowing much greater mixing of genetic material than single- or two-point crossover.

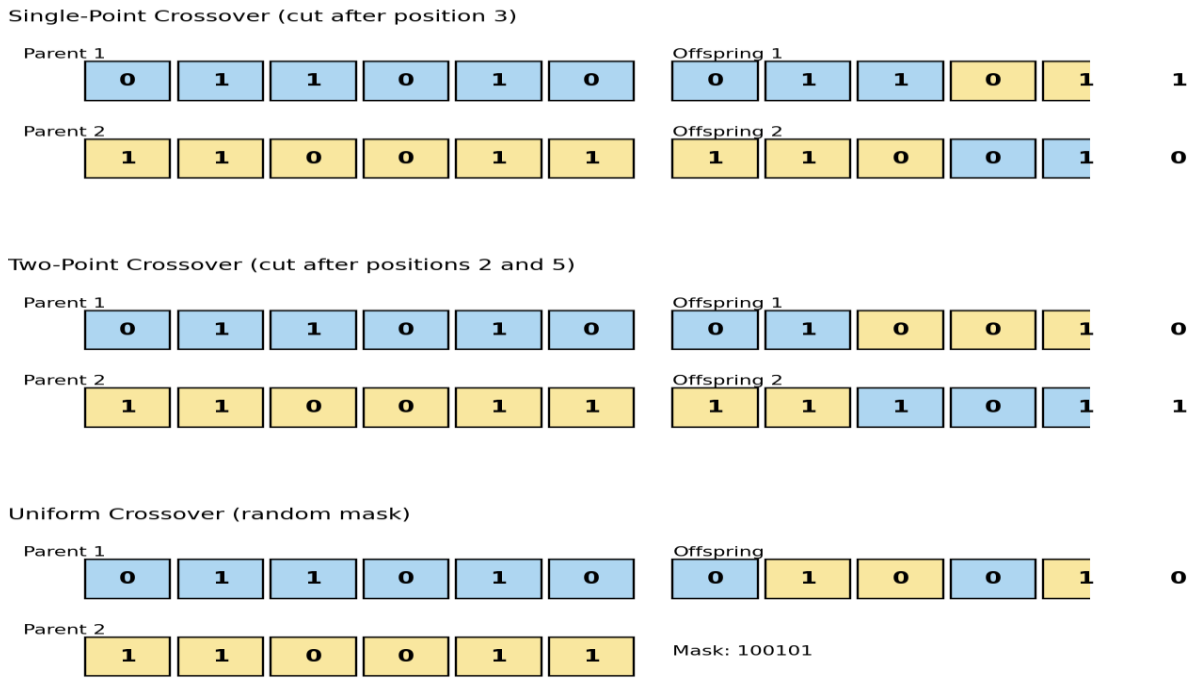

Fig. 1.5 – Single-Point, Two-Point, and Uniform Crossover

Mutation introduces small, random changes into a chromosome, ensuring that the algorithm continues to explore new regions of the search space and does not become trapped prematurely at a local optimum. In a binary-encoded GA, bit-flip mutation is used, in which each bit of an offspring is flipped (0 to 1, or 1 to 0) with a small, fixed mutation probability p_m, independently for every bit. For real-valued encodings, mutation is typically implemented by adding a small random perturbation, often drawn from a Gaussian distribution, to the gene value.

1.5.8 Worked Example

Consider the problem of maximising $f(x) = x^2$ over the integers 0 to 31, using a genetic algorithm with 5-bit chromosomes (since $2^5 = 32$). Suppose the initial population, generated at random, consists of four chromosomes:

Chromosome	x (decoded)	Fitness $f(x)=x^2$	% of total fitness
01101	13	169	14.4%
11000	24	576	49.2%
01000	8	64	5.5%
10011	19	361	30.9%

The total fitness of the population is 169 + 576 + 64 + 361 = 1170. Using roulette-wheel selection, chromosome '11000' (x = 24), having the highest fitness, has the greatest probability of being selected as a parent — approximately 49.2%, as depicted in Fig. 1.4. Suppose chromosomes '01101' and '11000' are selected as parents and a single-point crossover is applied after the third bit: parent1 = 011|01, parent2 = 110|00. Exchanging the tails produces offspring '011 00' = 01100 (x = 12) and '110 01' = 11001 (x = 25). Notice that offspring '11001' (x = 25, fitness 625) is fitter than either parent — this illustrates how crossover can combine good 'building blocks' from parents to create even better solutions. Mutation would then flip a randomly chosen bit of an offspring with a very small probability, e.g. changing '11001' to '11011' (x = 27), to maintain diversity and to allow exploration of parts of the search space not currently represented in the population.

This process is repeated over many generations; because fitter chromosomes are selected more often as parents, the average fitness of the population, as well as the fitness of the best individual, tends to increase from generation to generation until the algorithm converges close to the optimum, x = 31, f(x) = 961. If the new generation (after crossover and mutation on all four parents) turns out to be {01100, 11001, 11011, 10011}, decoded as x = 12, 25, 27, 19 with fitnesses 144, 625, 729, 361, the average fitness of the new population is (144+625+729+361)/4 = 464.75, a marked increase over the original average fitness of 1170/4 = 292.5 — demonstrating the improving trend that a genetic algorithm exhibits across generations.

1.5.9 The Schema Theorem (Building Block Hypothesis)

Holland's Schema Theorem provides a theoretical explanation for why genetic algorithms work. A schema is a template (e.g. '1*0*1', where '*' denotes 'don't care') that describes a subset of chromosomes sharing certain fixed bit values. The Schema Theorem states that short, low-order schemata with above-average fitness (called 'building blocks') receive an exponentially increasing number of representatives in successive generations. The Building Block Hypothesis conjectures that a genetic algorithm achieves good overall performance by combining these building blocks together via crossover, progressively assembling better and better partial solutions into a global solution.

1.5.10 Types of Evolutionary Algorithms

Algorithm	Representation	Primary Operator	Typical Use
Genetic Algorithm (GA)	Binary / real-valued strings	Crossover + Mutation	Optimisation, search problems

Algorithm	Representation	Primary Operator	Typical Use
Genetic Programming (GP)	Tree-structured programs	Sub-tree crossover	Automatic program/formula induction
Evolution Strategy (ES)	Real-valued vectors	Mutation (self-adaptive)	Continuous parameter optimisation
Evolutionary Programming (EP)	Finite-state machines / real vectors	Mutation only	Behavioural/strategy evolution

1.5.11 Effect of GA Control Parameters

The performance of a genetic algorithm is strongly influenced by the choice of its control parameters — population size, crossover rate, and mutation rate. Selecting appropriate values for these parameters is often problem-dependent and is itself an optimisation task (sometimes performed using a 'meta' genetic algorithm).

Parameter	Typical Range	Effect of Increasing the Value
Population size	20 – 200 (problem-dependent)	Improves diversity and solution quality but increases computation time per generation
Crossover rate (p_c)	0.6 – 0.9	Higher rate promotes faster mixing of genetic material but may disrupt good building blocks if too high
Mutation rate (p_m)	0.001 – 0.05	Higher rate increases exploration/diversity but too high a value turns the search into random search, destroying good solutions
Number of generations	Problem-dependent, often 100 – 1000+	More generations allow further refinement but with diminishing returns and higher computational cost

1.5.12 Convergence and Premature Convergence

A genetic algorithm is said to have converged when the population has largely lost its diversity, and successive generations no longer produce significant improvement in the best (or average) fitness. Ideally, convergence should occur once the algorithm has located the global optimum (or a solution acceptably close to it). However, GAs are susceptible to a phenomenon known as premature convergence, in which the population converges to a suboptimal (locally optimal) solution well before the search space has been adequately explored. Premature convergence typically occurs when a small number of highly fit ('super') individuals dominate the selection process early on, rapidly reducing genetic diversity. Common remedies include: using rank-based or tournament selection instead of pure fitness-proportional (roulette-wheel) selection; increasing the mutation rate; employing niching or fitness-sharing techniques to preserve diverse sub-populations; and using larger population sizes.

1.5.13 Genetic Algorithms versus Traditional Optimisation Techniques

Basis	Traditional (Calculus-based/Enumerative) Methods	Genetic Algorithms
Search points	Work with a single point at a time	Work with a population of points simultaneously
Derivative information	Often require gradient/derivative information	Require only the objective (fitness) function value — derivative-free
Search rule	Deterministic transition rules	Probabilistic transition rules (stochastic search)
Risk of local optima	High — can easily get trapped in a local optimum	Lower — population diversity helps escape local optima
Applicability	Best suited to smooth, differentiable, unimodal problems	Well suited to complex, discontinuous, multi-modal, combinatorial problems

1.5.14 Worked Example 2 — GA Applied to the 0/1 Knapsack Problem

The 0/1 Knapsack Problem is a classic combinatorial optimisation problem: given a set of items, each with a weight and a value, and a knapsack with a maximum weight capacity, choose a subset of items that maximises total value without exceeding the capacity. Consider 4 items with (weight, value) pairs (2,3), (3,4), (4,5), (5,8), and a knapsack capacity of 8. A natural chromosome representation is a 4-bit binary string, where bit i = 1 means item i is included and 0 means it is excluded (i.e. binary encoding, as discussed in Section 1.5.4).

The fitness function must penalise infeasible solutions (those exceeding the capacity). A common approach is: fitness = total value of selected items, if total weight ≤ capacity; fitness = 0 (or a heavily penalised value), otherwise. For chromosome '1011' (items 1, 3, 4 selected), total weight = 2+4+5 = 11 > 8, so this chromosome is infeasible and receives zero (or penalised) fitness. For chromosome '1010' (items 1 and 3 selected), total weight = 2+4 = 6 ≤ 8, and total value = 3+5 = 8, a feasible, reasonably fit solution. For chromosome '0011' (items 3 and 4 selected), total weight = 4+5 = 9 > 8, infeasible. For chromosome '0110' (items 2 and 3), weight = 3+4=7 ≤ 8, value = 4+5 = 9, an even fitter feasible solution. Running the GA cycle (selection, crossover, mutation) over successive generations on a population of such chromosomes progressively favours feasible, high-value combinations — in this small example, the optimal solution is items 2 and 4 (weight 3+5=8, value 4+8=12), which the GA is expected to discover after a sufficient number of generations, even though it never exhaustively enumerates all 2^4 = 16 possible combinations.

1.5.15 Extended Case Study — GA Applied to the Travelling Salesman Problem (TSP)

The Travelling Salesman Problem asks for the shortest possible route that visits each of a given set of cities exactly once and returns to the starting city. It is a classic NP-hard combinatorial optimisation problem frequently used to benchmark evolutionary algorithms, because the number of possible tours grows factorially with the number of cities, quickly making exhaustive enumeration infeasible.

- Encoding: Since every city must appear exactly once in a tour, binary encoding is unsuitable; instead, permutation encoding is used, in which a chromosome is simply an ordered list of city indices, e.g. [3,1,4,2,5] for a 5-city problem, representing the tour 3→1→4→2→5→3.
- Fitness function: The fitness of a chromosome is usually taken as the reciprocal of the total tour distance (or the negative of the distance, if minimisation is handled directly), so that shorter tours receive higher fitness.
- Crossover: Standard single-point or uniform crossover cannot be used directly, because naively swapping segments between two permutation chromosomes typically produces an invalid tour (with repeated or missing cities). Specialised operators such as Order Crossover (OX), Partially Mapped Crossover (PMX), or Cycle Crossover (CX) are used instead, which are specifically designed to preserve the permutation property of offspring.
- Mutation: Common permutation-preserving mutation operators include swap mutation (exchanging the positions of two randomly chosen cities in the tour) and inversion mutation (reversing the order of a randomly chosen sub-sequence of the tour).

As the GA evolves the population of candidate tours across generations, selection favours shorter tours, and the specialised crossover/mutation operators progressively recombine and perturb good partial routes (sub-tours), typically converging to a tour that is very close to, though not always exactly, the true shortest possible route — illustrating both the power and the approximate nature of genetic-algorithm-based optimisation.

1.5.16 Comparative Overview of Artificial Neural Network Architectures

Although a detailed treatment of neural networks lies outside the scope of this unit, it is useful, for context, to be aware of the major categories of ANN architecture that are commonly hybridised with fuzzy and evolutionary techniques in soft computing systems.

Architecture	Connectivity	Typical Use
Feed-forward (e.g. Multi-Layer Perceptron)	Signals flow strictly from input to output, no loops	Classification, regression, function approximation

Architecture	Connectivity	Typical Use
Recurrent Neural Network (RNN)	Contains feedback loops/connections	Sequential/time-series data, language modelling
Radial Basis Function (RBF) Network	Single hidden layer using radial basis activation functions	Function approximation, used within ANFIS-like hybrid systems
Self-Organising Map (SOM)	Unsupervised, competitive learning, topology-preserving	Clustering, dimensionality reduction, visualisation
Convolutional Neural Network (CNN)	Local receptive fields with shared weights (convolutions)	Image and spatial pattern recognition

1.5.17 Soft Computing and Industry 4.0

The advent of Industry 4.0 — characterised by cyber-physical systems, the Industrial Internet of Things (IIoT), and large-scale automation — has considerably widened the relevance of soft computing. Modern smart factories generate vast amounts of imprecise, noisy sensor data that must be processed in real time to support predictive maintenance, adaptive process control, and autonomous decision making. Soft computing techniques, particularly hybrid neuro-fuzzy and genetic-fuzzy systems, are increasingly embedded directly into industrial controllers and edge-computing devices, since they combine the interpretability required for safety-critical decisions with the adaptability required to cope with continuously changing operating conditions.

1.5.18 Fitness Scaling and Ranking

In a genetic algorithm using fitness-proportional (roulette-wheel) selection, raw fitness values can sometimes cause problems: early in a run, one or two exceptionally fit individuals may dominate selection, causing premature convergence (Section 1.5.12); later in a run, when most individuals have similar, near-optimal fitness, the selection pressure may become too weak to drive further improvement. Fitness scaling techniques address this by transforming raw fitness values before they are used for selection.

- Linear scaling: The raw fitness f is transformed to f' = a·f + b, with constants a and b chosen so that the average scaled fitness equals the average raw fitness, while controlling the ratio of maximum to average fitness (typically kept close to a value such as 2.0) to maintain steady, moderate selection pressure throughout the run.
- Sigma truncation: Fitness is scaled relative to the population's standard deviation, reducing the effect of a few outlier individuals with extremely high fitness.
- Rank-based scaling: Rather than using raw fitness values at all, individuals are simply ranked from best to worst, and selection probability is assigned based on this rank (e.g.

using a linear or exponential ranking function), which decouples selection pressure entirely from the numerical spread of the fitness values.

1.5.19 Case Study — A Neuro-Genetic Hybrid System (Illustrative)

To appreciate how the constituent technologies of soft computing are combined in practice, consider a neuro-genetic hybrid system used to design an optimal neural-network-based controller for a home appliance, such as a smart washing machine. Here, a multi-layer perceptron (Section 1.4.2.1) is trained to map sensor readings (load weight, water turbidity, fabric type) to an appropriate wash-cycle setting. Rather than fixing the network's architecture (number of hidden neurons, choice of activation functions) and learning-rate parameters by trial and error, a genetic algorithm (Section 1.5) is used to search over these design choices: each chromosome encodes a candidate network architecture and parameter set, the fitness function is the resulting network's accuracy (or energy efficiency) on a validation dataset of past wash cycles, and the GA evolves, across generations, increasingly well-performing network designs. This example illustrates the central theme of soft computing: rather than using any single technique in isolation, the learning capability of neural networks and the global search capability of genetic algorithms are combined synergistically, each compensating for a limitation of the other (the difficulty of manually designing a good network architecture, and the relatively slow convergence of GA-only approaches to fine parameter tuning).

1.5.20 Multi-Objective Genetic Algorithms (Brief Overview)

Many real-world optimisation problems involve multiple, often conflicting, objectives — for example, designing a product that simultaneously minimises cost and maximises quality. Multi-Objective Genetic Algorithms (MOGAs), such as the widely used Non-dominated Sorting Genetic Algorithm (NSGA-II), extend the simple genetic algorithm framework studied in this section to handle such problems. Rather than converging to a single best solution, a MOGA seeks to find a set of 'Pareto-optimal' solutions — solutions for which no objective can be improved without worsening at least one other objective — collectively known as the Pareto front. The decision maker can then examine this entire front of trade-off solutions and select the one that best matches their particular priorities, rather than being forced to commit to a single, pre-combined weighted objective before the search even begins. Multi-objective evolutionary algorithms are increasingly used in engineering design, scheduling, and resource-allocation problems where trade-offs between competing goals are unavoidable.

1.5.21 Parallel and Distributed Genetic Algorithms

Since a genetic algorithm evaluates the fitness of every chromosome in a population independently, the fitness-evaluation step is 'embarrassingly parallel' and lends itself naturally to parallel and distributed implementation on multi-core processors, GPUs, or computing clusters. Two common parallelisation strategies are the island model, in which the overall population is divided into several independent sub-populations ('islands'), each evolved somewhat independently (typically on a separate processor), with occasional migration of individuals between islands to share good genetic material while still preserving diversity across islands; and the cellular (fine-grained) model, in which each individual chromosome is assigned to a cell in a spatial grid and can only mate with chromosomes in neighbouring cells, naturally limiting the spread of any single highly fit individual and thereby reducing the risk of premature convergence (Section 1.5.12). Parallel genetic algorithms are especially valuable for computationally expensive fitness functions, such as those requiring a full engineering simulation (e.g. a finite-element or computational-fluid-dynamics run) to evaluate a single candidate design.

1.6 Characteristics of Soft Computing

Soft computing systems share a common set of defining characteristics that distinguish them from conventional computing paradigms. Understanding these characteristics helps in deciding when a soft-computing approach is appropriate for a given problem.

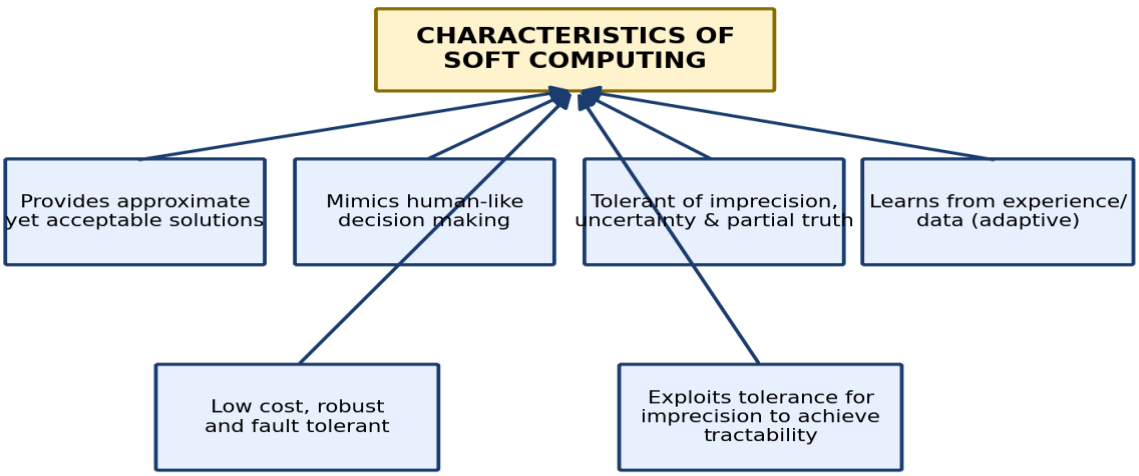

Fig. 1.6 – Characteristics of Soft Computing

- Approximate reasoning: Soft computing systems provide approximate, though acceptable, solutions to imprecisely or incompletely formulated problems, in contrast to hard computing's requirement of exactness.
- Human-like decision making: They incorporate human-like reasoning, learning, and adaptation, making them suitable for tasks that humans perform intuitively, such as recognition and control.
- Tolerance of imprecision, uncertainty and partial truth: This is the defining principle of soft computing; it is exploited deliberately to gain tractability and lower cost.
- Learning and adaptability: Constituent techniques such as neural networks and genetic algorithms can learn from data/experience and adapt to changing environments without being explicitly reprogrammed.
- Robustness and fault tolerance: Soft computing systems degrade gracefully in the presence of noisy, incomplete, or conflicting information rather than failing outright.
- Low solution cost: Approximate solutions can typically be obtained with significantly less computational effort and cost than exhaustive, exact methods.
- Parallelism: Many soft computing techniques, especially neural networks and evolutionary algorithms, are inherently parallel, distributing computation across many simple units.

1.6.1 Illustrative Example — Approximate Reasoning versus Exact Computation

To appreciate the practical significance of these characteristics, consider the everyday task of deciding how much to brake while approaching a red traffic signal while driving. An exact, hard-computing solution would require precise knowledge of the vehicle's current speed, the exact distance to the stop line, the road's friction coefficient, the vehicle's mass, and the braking system's exact response characteristics, and would then solve the equations of motion to compute the precise braking force needed to stop exactly at the line. In practice, a human driver does nothing of the sort: relying on approximate perception ('I am going somewhat fast', 'the signal is fairly close') and approximate reasoning ('so I should brake moderately hard now'), the driver arrives at a perfectly adequate, safe braking decision almost instantaneously, without any exact numerical computation. A fuzzy-logic-based automotive braking-assistance system is designed to mimic exactly this human approximate-reasoning process, using linguistic rules such as 'IF speed is HIGH AND distance is SMALL THEN braking-force is STRONG', rather than solving the underlying differential equations of motion explicitly. This example

crystallises why soft computing's tolerance for imprecision is not a weakness but a deliberate design choice that mirrors, and often outperforms, natural human decision-making in real-time, resource-constrained situations.

1.7 Recent Trends in Soft Computing

Research in soft computing has moved rapidly from studying its constituent technologies in isolation towards hybridising them to exploit the complementary strengths of each. Some of the important recent trends are summarised below.

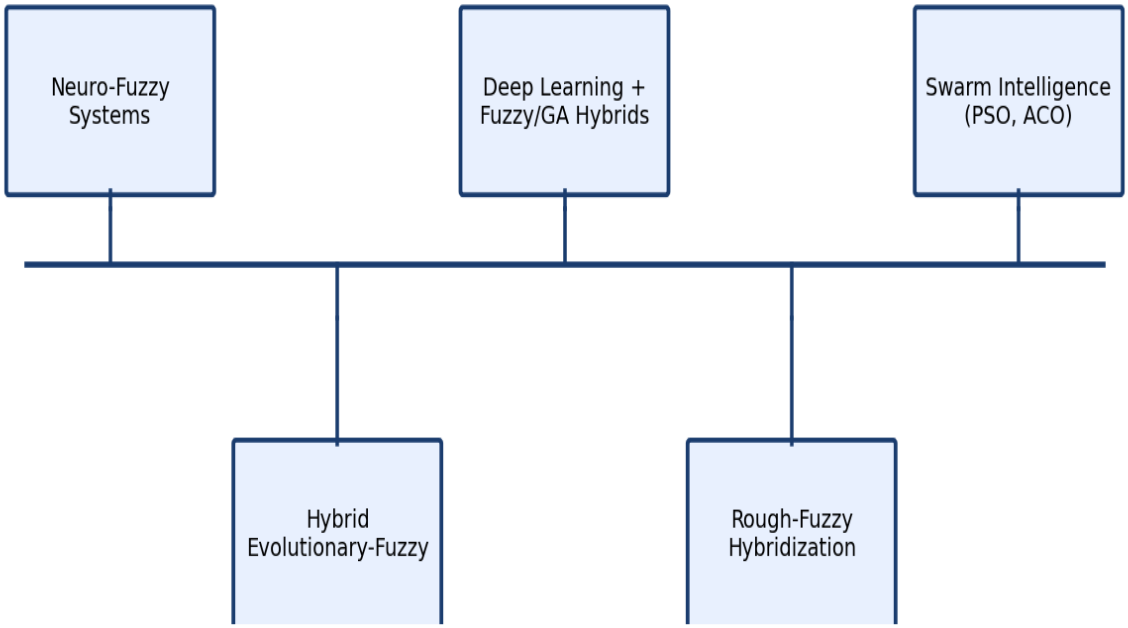

Fig. 1.7 – Recent Trends in Soft Computing

- Neuro-Fuzzy Systems: Combine the learning capability of neural networks with the human-interpretable, linguistic rule-based reasoning of fuzzy logic (e.g. Adaptive Neuro-Fuzzy Inference System — ANFIS).
- Genetic-Fuzzy Systems: Use genetic algorithms to automatically tune the membership functions or rule base of a fuzzy system, reducing the burden of manual design.
- Neuro-Genetic Systems: Use genetic algorithms to optimise the topology, weights, or learning parameters of a neural network.

- Rough-Fuzzy and Fuzzy-Rough Hybridisation: Combine rough set theory's ability to handle data granularity with fuzzy logic's ability to handle vagueness, useful in advanced data mining.
- Swarm Intelligence: Techniques such as Particle Swarm Optimisation (PSO) and Ant Colony Optimisation (ACO), inspired by the collective behaviour of social organisms, are now regarded as important soft computing tools alongside classical GAs.
- Deep Learning integration: Modern deep neural architectures are increasingly hybridised with fuzzy and evolutionary components to improve interpretability and to automate hyper-parameter tuning.
- Applications in Big Data and IoT: Soft computing techniques are being adapted to handle the volume, velocity and uncertainty of data generated by IoT devices and large-scale sensor networks.
- Explainable Soft Computing: Increasing emphasis on making fuzzy, neural, and evolutionary models more interpretable and explainable, especially in safety-critical and regulated domains such as healthcare and finance.

1.8 Applications of Soft Computing Techniques

Because of their ability to handle imprecision and to learn from data, soft computing techniques have found application in almost every engineering and scientific discipline.

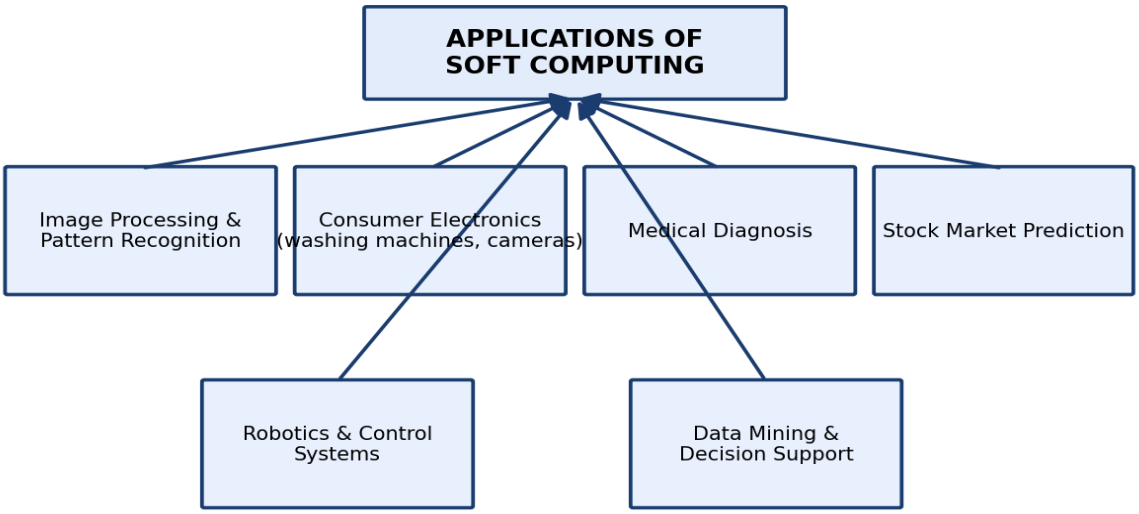

Fig. 1.8 – Application Domains of Soft Computing

Domain	Typical Application
Consumer electronics	Fuzzy-logic-controlled washing machines, rice cookers, air conditioners, and camera auto-focus/auto-exposure systems
Automotive engineering	Anti-lock braking systems (ABS), automatic transmission control, engine fuel-injection control
Medicine	Medical diagnosis support systems, ECG/EEG signal classification, drug-dosage optimisation, medical image segmentation
Robotics & control	Path planning, obstacle avoidance, adaptive control of robotic manipulators
Finance	Stock-market prediction, credit-risk scoring, algorithmic trading, fraud detection
Pattern recognition	Handwriting and speech recognition, fingerprint and face recognition, image processing
Data mining	Clustering, classification, feature selection, association rule mining under uncertainty
Power systems	Load forecasting, fault diagnosis, optimal power-flow control
Manufacturing	Scheduling, quality control, process optimisation using genetic algorithms

A few of these application domains merit a closer look. In consumer electronics, a fuzzy-logic-controlled washing machine determines the wash cycle duration, water level, and detergent amount based on fuzzy sensor readings of load size, dirtiness, and fabric type, providing a level of adaptability that would be very difficult to achieve using a fixed, rule-based hard-computing controller. In medicine, soft-computing-based diagnostic support systems combine fuzzy reasoning to handle imprecise symptom descriptions (e.g. 'moderate fever', 'severe pain') with neural-network-based pattern recognition on medical images or signals, providing decision support to clinicians while explicitly quantifying the uncertainty inherent in a diagnosis. In finance, genetic algorithms are used to evolve trading strategies and optimise investment portfolios, exploiting their ability to search vast combinatorial spaces of possible strategies far more efficiently than exhaustive enumeration.

1.8.1 Detailed Case Study — Fuzzy Logic in a Consumer Appliance

As a concrete illustration of how soft computing techniques are embedded in everyday consumer products, consider a commercially available fuzzy-logic rice cooker. A conventional (hard-computing) rice cooker typically uses a simple bimetallic thermostat that switches off heating once the internal temperature crosses a single, fixed threshold, regardless of the type or quantity of rice being cooked, often resulting in inconsistent results (undercooked or overcooked rice) across different conditions. A fuzzy-logic rice cooker, by contrast, continuously monitors the temperature-rise profile inside the cooking pot using multiple sensors, and fuzzifies this information into linguistic terms such as 'temperature rising

Slowly/Moderately/Rapidly'. A fuzzy rule base, containing rules such as 'IF temperature rise is Rapid AND moisture level is Low THEN reduce heating power Moderately', continuously adjusts the heating element's power level throughout the cooking cycle, rather than performing a single on/off switch. This allows the appliance to adaptively compensate for differences in rice quantity, rice variety, water quantity, and even altitude (which affects boiling point), consistently producing well-cooked rice under a wide range of conditions — a level of adaptive, human-like performance that would be extremely difficult, and considerably more expensive, to replicate using a purely hard-computing (fixed threshold or classical PID) control strategy.

1.9 Advantages and Limitations of Soft Computing

1.9.1 Advantages

- Can model and solve problems that are difficult or impossible to model analytically.
- Robust to noise, incomplete data and uncertainty.
- Capable of learning and adapting to changing environments.
- Often computationally cheaper than exhaustive/exact methods.
- Provides human-interpretable outputs when combined with fuzzy rules.
- Facilitates hybridisation, combining the complementary strengths of different techniques for improved overall performance.

1.9.2 Limitations

- Solutions are approximate and not guaranteed to be globally optimal.
- Design and tuning of membership functions, network architectures, or genetic operators often require expert knowledge and experimentation.
- Convergence of algorithms such as GA and neural network training can be slow, and there is no absolute guarantee of convergence.
- Lack of a unifying mathematical theory (unlike hard computing) — soft computing is a collection of heuristic methodologies rather than a single formal framework.
- Performance can be sensitive to the choice of parameters (e.g. mutation rate, learning rate, membership-function shape), often requiring trial-and-error tuning.

1.9.3 Future Scope of Soft Computing

Looking ahead, soft computing is expected to play an increasingly central role wherever large volumes of imprecise, incomplete, or rapidly changing data must be processed under real-time constraints. Emerging directions include the integration of soft computing with quantum computing (quantum-inspired evolutionary algorithms), the design of explainable and trustworthy hybrid AI systems for regulated sectors such as healthcare and finance, and the deployment of lightweight, energy-efficient fuzzy and evolutionary algorithms on edge devices and embedded microcontrollers for real-time IoT applications. As the boundary between symbolic, hard-computing AI and soft, sub-symbolic computational intelligence continues to blur — for instance, in modern large-scale machine-learning systems that combine statistical learning with rule-based reasoning — the foundational principles of soft computing studied in this unit remain as relevant as ever.

1.10 Summary

Soft computing represents a paradigm shift from the precision-driven approach of hard computing to a more flexible, human-inspired approach that tolerates imprecision, uncertainty, and partial truth in order to achieve tractability, robustness, and low-cost solutions for complex real-world problems. Its principal constituents — fuzzy logic, artificial neural networks, genetic algorithms/evolutionary computing, probabilistic reasoning, and rough set theory — are designed to complement rather than compete with one another, and are increasingly hybridised (neuro-fuzzy, genetic-fuzzy, neuro-genetic, rough-fuzzy) to exploit their combined strengths. Evolutionary computing, inspired by Darwinian natural selection, provides a powerful, population-based search and optimisation methodology built around the operators of selection, crossover, and mutation, with theoretical underpinnings such as the Schema Theorem explaining its effectiveness. Soft computing techniques today underpin applications ranging from consumer electronics and automotive control to medical diagnosis, finance, robotics, and big-data analytics.

1.11 Review Questions

9. Define soft computing. How does it differ from conventional (hard) computing? Illustrate with a suitable table.
10. Who coined the term 'soft computing' and what is the underlying guiding principle of the discipline?
11. List and briefly explain the five principal constituent technologies of soft computing.

12. Differentiate between soft computing and conventional symbolic artificial intelligence.
13. What is evolutionary computing? Describe, with the help of a diagram, the cycle of operations in a genetic algorithm.
14. Explain the following terms with reference to genetic algorithms: chromosome, gene, allele, fitness function, crossover, mutation, elitism.
15. Describe the different chromosome encoding schemes used in genetic algorithms with examples.
16. Explain roulette-wheel, rank, and tournament selection methods with suitable examples.
17. Differentiate between single-point, two-point, and uniform crossover, with a suitable diagram.
18. State the Schema Theorem and explain the Building Block Hypothesis.
19. Trace the historical development of evolutionary computing, mentioning at least five important milestones.
20. Discuss the important characteristics of soft computing systems.
21. What are the recent trends in soft computing? Explain any three hybrid soft-computing paradigms in detail.
22. Describe five real-world application areas of soft computing with suitable examples.
23. What are the advantages and limitations of soft computing as compared to hard computing?
24. Consider the fitness function $f(x) = x^2$, $x \in [0,31]$, encoded using 5-bit binary chromosomes. Given a population of chromosomes 10010, 00101, 11010 and 01110, compute the fitness of each and determine the probability of selection of each chromosome under roulette-wheel selection.
25. Differentiate between Genetic Algorithms, Genetic Programming, Evolution Strategies, and Evolutionary Programming.

1.12 Additional Solved Numerical Problems

Problem 1

A genetic algorithm uses 4-bit binary chromosomes to represent integers 0–15, with fitness function f(x) = x. Given the population {0110, 1001, 0011, 1111}, compute the fitness and roulette-wheel selection probability of each chromosome.

Solution: Decoding gives x-values 6, 9, 3, and 15 respectively, so the fitness values are 6, 9, 3, and 15 (since f(x)=x). The total fitness is 6+9+3+15 = 33. The selection probabilities (fitness ÷ total fitness) are therefore 6/33 ≈ 18.2%, 9/33 ≈ 27.3%, 3/33 ≈ 9.1%, and 15/33 ≈ 45.4% respectively. Chromosome '1111' (x=15), being the fittest, has the highest chance of being selected as a parent, as expected.

Problem 2

Two parent chromosomes, 101100 and 011011, undergo two-point crossover with crossover points after bit positions 2 and 4. Determine the resulting offspring.

Solution: Splitting each parent at positions 2 and 4 gives Parent 1 = 10|11|00 and Parent 2 = 01|10|11. Swapping the middle segments produces Offspring 1 = 10|10|00 = 101000 and Offspring 2 = 01|11|11 = 011111.

Problem 3

A chromosome 110101 is subjected to bit-flip mutation with mutation probability p_m = 0.1 applied independently to every bit. If bits at positions 2 and 5 (counting from the left, starting at 1) are selected for mutation, what is the resulting chromosome?

Solution: The original chromosome is 1 1 0 1 0 1 (positions 1–6). Flipping bit 2 (currently '1') gives '0', and flipping bit 5 (currently '0') gives '1'. The mutated chromosome becomes 1 0 0 1 1 1 = 100111.

1.13 Glossary of Key Terms

Term	Meaning
Soft computing	A consortium of computing methodologies tolerant of imprecision, uncertainty, and partial truth
Hard computing	Conventional computing requiring precise analytical models and producing exact results
Fuzzy logic	Multi-valued logic that generalises Boolean logic to handle degrees of truth
Neural network	A computing system of interconnected neurons inspired by the biological brain
Genetic algorithm	A population-based search/optimisation algorithm inspired by natural selection
Chromosome	An encoded candidate solution used in a genetic algorithm
Fitness function	A function evaluating how good a candidate solution is

Term	Meaning
Crossover	A genetic operator combining genetic material from two parent chromosomes
Mutation	A genetic operator introducing small random changes to a chromosome
Elitism	Preserving the best chromosome(s) unchanged across generations
Premature convergence	Loss of population diversity before the global optimum is found
Rough set	A set characterised by lower and upper crisp approximations, used to handle data granularity
Probabilistic reasoning	Reasoning under uncertainty using probability theory, e.g. Bayesian inference
Neuro-fuzzy system	A hybrid system combining neural-network learning with fuzzy-logic reasoning
Genetic-fuzzy system	A hybrid system that uses genetic algorithms to tune a fuzzy system
Swarm intelligence	Optimisation techniques inspired by the collective behaviour of social organisms (e.g. PSO, ACO)
Perceptron	The simplest artificial neuron model, computing a weighted sum passed through an activation function
Schema	A template describing a subset of chromosomes that share certain fixed gene values
Population	The set of candidate solutions (chromosomes) maintained by a genetic algorithm at a given generation
Generation	One complete iteration of the evaluate–select–recombine–mutate cycle in an evolutionary algorithm

1.14 Multiple Choice Questions

26. Soft computing was introduced by: (a) John Holland (b) Lotfi A. Zadeh (c) Zdzisław Pawlak (d) Charles Darwin — Ans: (b)

27. Which of the following is NOT a constituent of soft computing? (a) Fuzzy Logic (b) Neural Networks (c) Relational Databases (d) Genetic Algorithms — Ans: (c)

28. Hard computing is based on: (a) Two-valued logic (b) Multi-valued logic (c) Approximate reasoning (d) Learning from data — Ans: (a)

29. Genetic Algorithms are based on the principle of: (a) Backpropagation (b) Survival of the fittest (c) Membership functions (d) Lower/upper approximation — Ans: (b)

30. Genetic Programming evolves: (a) Binary strings (b) Real-valued vectors (c) Tree-structured computer programs (d) Fuzzy rules — Ans: (c)

31. The operator that combines genetic material from two parent chromosomes is called: (a) Selection (b) Mutation (c) Crossover (d) Elitism — Ans: (c)

32. Which soft computing technique is primarily used to handle vagueness caused by insufficient information/granularity? (a) Fuzzy logic (b) Rough set theory (c) Neural networks (d) Genetic algorithms — Ans: (b)

33. ANFIS is an example of which hybrid soft-computing paradigm? (a) Rough-Fuzzy (b) Neuro-Fuzzy (c) Neuro-Genetic (d) Genetic-Fuzzy — Ans: (b)

34. Which of these is a characteristic of soft computing? (a) Requires exact analytical model (b) Intolerant of uncertainty (c) Tolerant of imprecision and partial truth (d) Always produces exact results — Ans: (c)

35. Evolution Strategies were developed by: (a) John Holland (b) Rechenberg and Schwefel (c) John Koza (d) Lotfi Zadeh — Ans: (b)

36. In a genetic algorithm, elitism refers to: (a) Selecting only the worst individuals (b) Always mutating the best individual (c) Retaining the best individual(s) unchanged in the next generation (d) Ignoring the fitness function — Ans: (c)

37. Permutation encoding is best suited for problems such as: (a) Knapsack problem (b) Travelling Salesman Problem (c) Sudoku (d) Bit-string counting — Ans: (b)

UNIT – II

FUZZY SYSTEMS

2.1 Introduction

In our everyday life we frequently use terms that do not have a sharp, precise boundary — 'the water is hot', 'he is tall', 'the traffic is heavy', 'the room is dim'. Classical (crisp) set theory, and consequently classical two-valued logic, cannot adequately represent such statements because an element either belongs to a set or it does not; there is no notion of partial belonging. Fuzzy Set Theory, introduced by Professor Lotfi A. Zadeh in his landmark 1965 paper 'Fuzzy Sets', extends classical set theory by allowing partial membership, thereby providing a rigorous mathematical framework to model and reason with such vague, imprecise, linguistic concepts.

This unit covers fuzzy sets, fuzzy relations, fuzzy logic, and fuzzy rule-based systems — the foundation on which fuzzy control, fuzzy decision making, and neuro-fuzzy systems (studied in later units) are built.

2.2 Fuzzy Sets

2.2.1 Crisp Sets versus Fuzzy Sets

In classical (crisp) set theory, a set A defined over a universe of discourse X is described by a characteristic function $\chi_A: X \rightarrow \{0, 1\}$, where $\chi_A(x) = 1$ if $x \in A$ and $\chi_A(x) = 0$ if $x \notin A$. There is no intermediate possibility. In contrast, a fuzzy set A defined over X is characterised by a membership function $\mu_A: X \rightarrow [0, 1]$, where $\mu_A(x)$ denotes the degree (or grade) to which x belongs to A. A value of $\mu_A(x) = 1$ indicates full membership, $\mu_A(x) = 0$ indicates no membership, and any value strictly between 0 and 1 indicates partial membership.

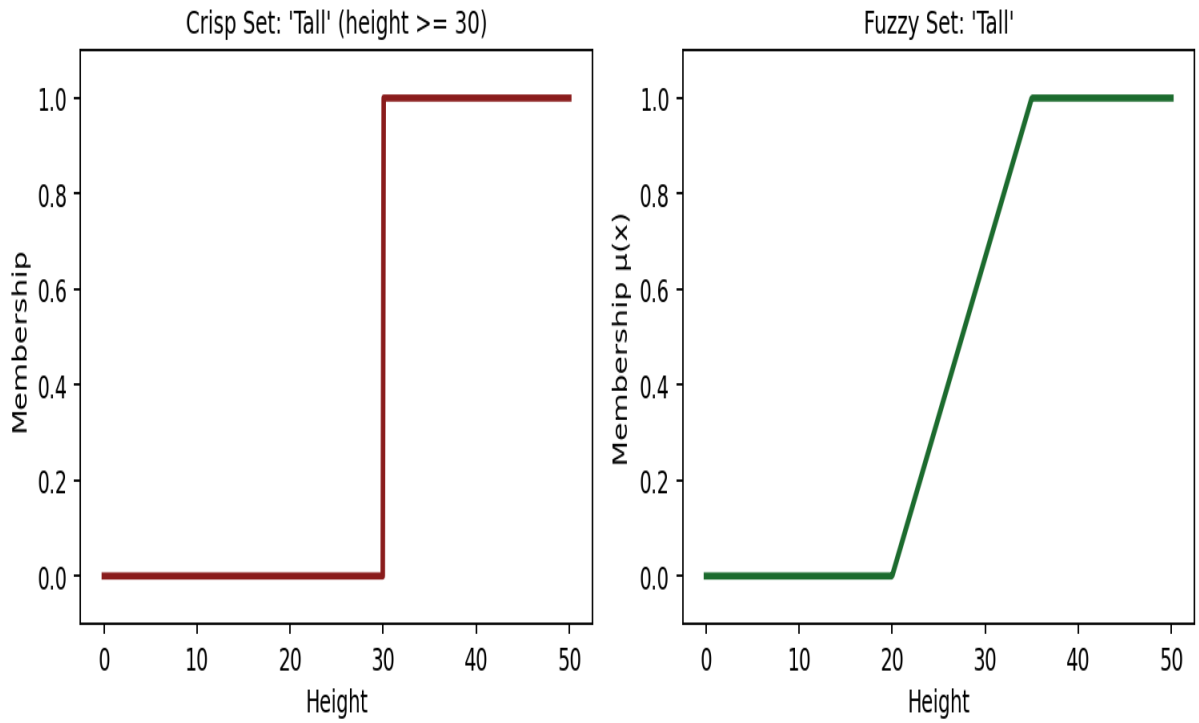

Fig. 2.1 – Crisp Set versus Fuzzy Set for the Concept 'Tall'

A fuzzy set A is formally written as the set of ordered pairs A = {(x, μ_A(x)) | x ∈ X}. For example, if X represents a set of people and A represents the fuzzy set 'tall people', a person of height 150 cm might have μ_A = 0.1, a person of 170 cm might have μ_A = 0.6, and a person of 190 cm might have μ_A = 1.0. When X is a discrete, finite set, A may also be written using Zadeh's notation A = μ_A(x1)/x1 + μ_A(x2)/x2 + ... + μ_A(xn)/xn, where '+' denotes union (not arithmetic addition) and '/' merely associates a membership value with an element.

2.2.2 Basic Definitions and Terminology

Term	Definition
Universe of discourse (X)	The set of all possible values relevant to a fuzzy variable
Support of A	The crisp set of all x ∈ X such that μ_A(x) > 0
Core of A	The crisp set of all x ∈ X such that μ_A(x) = 1
Height of A	The largest membership value attained by any x in X; height(A) = sup μ_A(x)
Normal fuzzy set	A fuzzy set whose height is exactly 1 (at least one element has full membership)
Subnormal fuzzy set	A fuzzy set whose height is less than 1
α-cut (alpha-cut)	The crisp set A_α = {x ∈ X | μ_A(x) ≥ α}, for 0 < α ≤ 1
Strong α-cut	The crisp set A_α⁺ = {x ∈ X | μ_A(x) > α}
Convex fuzzy set	A fuzzy set whose membership function is such that μ_A(λx1+(1-λ)x2) ≥ min(μ_A(x1), μ_A(x2)) for all x1,x2 ∈ X, λ ∈ [0,1]
Fuzzy singleton	A fuzzy set whose support is a single point in X
Cardinality of a fuzzy set	For a discrete fuzzy set, the sum of the membership values of all its elements, |A| = Σ μ_A(x)

As a worked illustration, consider a discrete universe of discourse X = {1,2,3,4,5,6,7,8,9,10} representing 'number of years of work experience', and let the fuzzy set A = 'experienced employee' be defined as A = 0/1 + 0.1/2 + 0.2/3 + 0.4/4 + 0.6/5 + 0.8/6 + 1/7 + 1/8 + 1/9 + 1/10. Here the support of A is {2,3,...,10} (all elements with non-zero membership), the core of A is {7,8,9,10} (elements with membership exactly 1), the height of A is 1 (so A is normal), and the 0.5-cut is A_0.5 = {6,7,8,9,10} (those elements with membership ≥ 0.5).

2.2.2.1 Representation of Fuzzy Sets

A fuzzy set may be represented in several equivalent notations depending on whether the universe of discourse is discrete or continuous:

- List (enumerative) notation: A = {(x1,μ1), (x2,μ2), ..., (xn,μn)}, used for discrete, finite universes.
- Zadeh's summation notation: A = μ1/x1 + μ2/x2 + ... + μn/xn (discrete case), or A = ∫ μ_A(x)/x (continuous case), where '+'/'∫' denote union rather than arithmetic addition/integration.
- Functional (analytical) notation: A is specified directly by giving the formula for μ_A(x), as is done for triangular, trapezoidal, and Gaussian membership functions.

2.2.2.2 The Decomposition (Resolution Identity) Theorem

An important theoretical result connecting fuzzy sets and their α-cuts is the Decomposition Theorem (also called the Resolution Identity), which states that any fuzzy set A can be exactly reconstructed from its family of α-cuts: A = ∪_{α∈(0,1]} α·A_α, where α·A_α denotes the fuzzy set obtained by scaling the crisp set A_α (i.e. assigning membership α to every element of A_α), and the union is taken over all α in (0,1]. In practical terms, this theorem justifies representing and manipulating a fuzzy set purely through a nested family of crisp α-cut sets — a technique frequently used in fuzzy arithmetic and in the implementation of fuzzy systems on digital computers, since crisp-set operations are computationally simpler than direct operations on continuous membership functions.

2.2.3 Membership Functions (MFs)

The membership function is the backbone of a fuzzy set; it quantitatively defines how each point in the input (universe of discourse) is mapped to a membership value between 0 and 1. Membership functions may be chosen based on the shape that best represents the vague concept

being modelled and the ease of computation required in a given application. The most commonly used shapes are triangular, trapezoidal, and Gaussian.

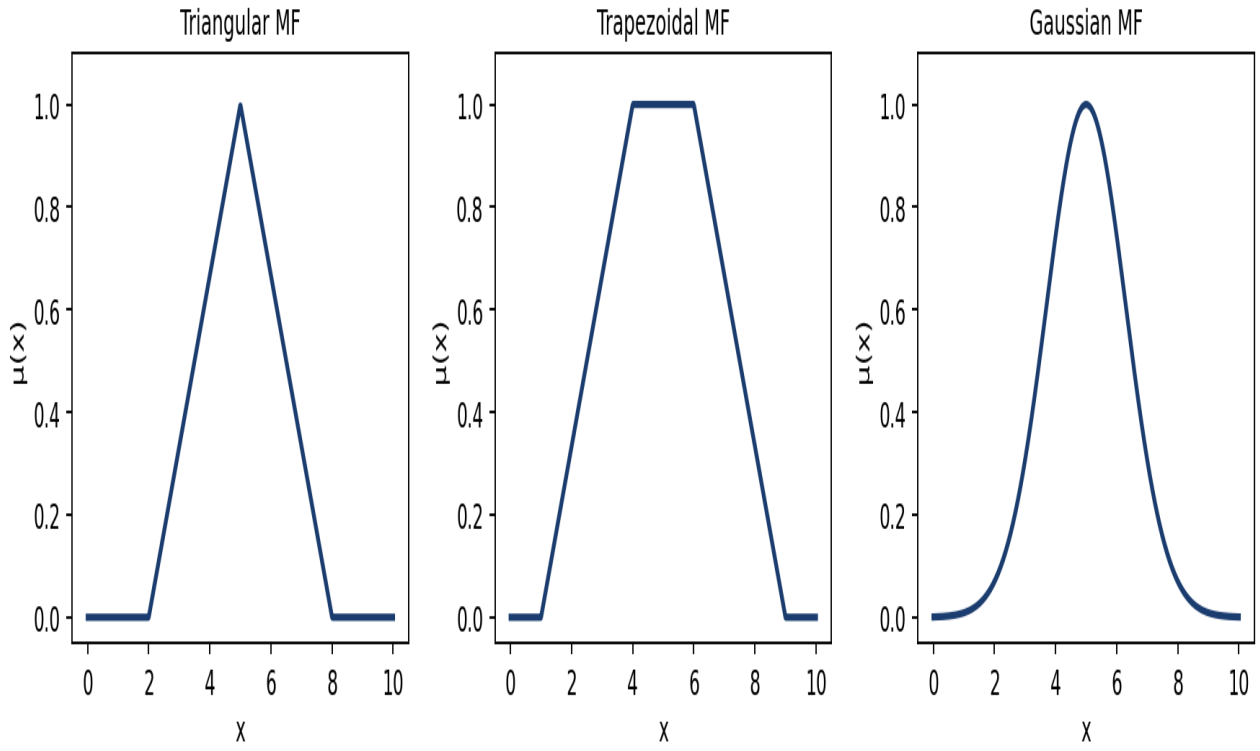

Fig. 2.2 – Commonly Used Membership Function Shapes

MF Type	Parameters	Formula (piecewise)
Triangular	a, b, c (a<b<c)	0 for x≤a; (x-a)/(b-a) for a<x≤b; (c-x)/(c-b) for b<x≤c; 0 for x>c
Trapezoidal	a, b, c, d (a<b≤c<d)	0 for x≤a; (x-a)/(b-a) for a<x<b; 1 for b≤x≤c; (d-x)/(d-c) for c<x<d; 0 for x≥d
Gaussian	mean m, spread σ	μ(x) = exp(-(x-m)² / 2σ²)

Other MF shapes used in practice include the sigmoidal (S-shaped) function and the generalised bell function, illustrated in Fig. 2.3. The generalised bell function, μ(x) = 1 / (1 + |(x-c)/a|^(2b)), is controlled by three parameters (a — half-width, b — slope steepness, c — centre), giving it more shaping flexibility than a simple Gaussian. The sigmoidal function, μ(x) = 1 / (1 + e^(-k(x-c))), is asymmetric and is commonly used to represent one-sided linguistic concepts such as 'large' or 'small' at the extreme ends of a universe of discourse, or as an activation function bridging fuzzy systems and neural networks in neuro-fuzzy models.

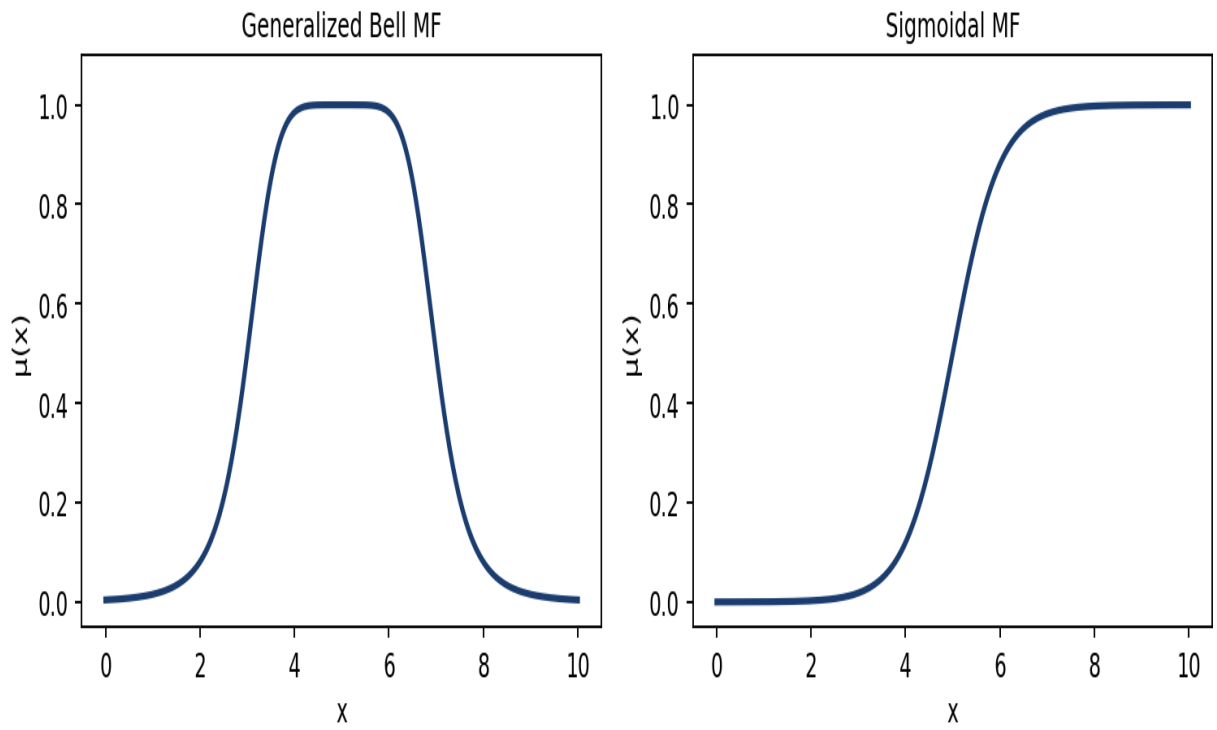

Fig. 2.3 – Generalized Bell and Sigmoidal Membership Functions

2.2.4 Operations on Fuzzy Sets

Just as classical sets support union, intersection, and complement, fuzzy sets support analogous operations, generalised using the membership function rather than the characteristic function. Let A and B be two fuzzy sets defined on the same universe of discourse X.

Operation	Definition
Union (A ∪ B)	μ_{A∪B}(x) = max(μ_A(x), μ_B(x))
Intersection (A ∩ B)	μ_{A∩B}(x) = min(μ_A(x), μ_B(x))
Complement (A')	μ_{A'}(x) = 1 - μ_A(x)
Algebraic sum	μ_A(x) + μ_B(x) - μ_A(x)·μ_B(x)
Algebraic product	μ_A(x) · μ_B(x)
Bounded sum	min(1, μ_A(x) + μ_B(x))
Bounded difference	max(0, μ_A(x) + μ_B(x) - 1)

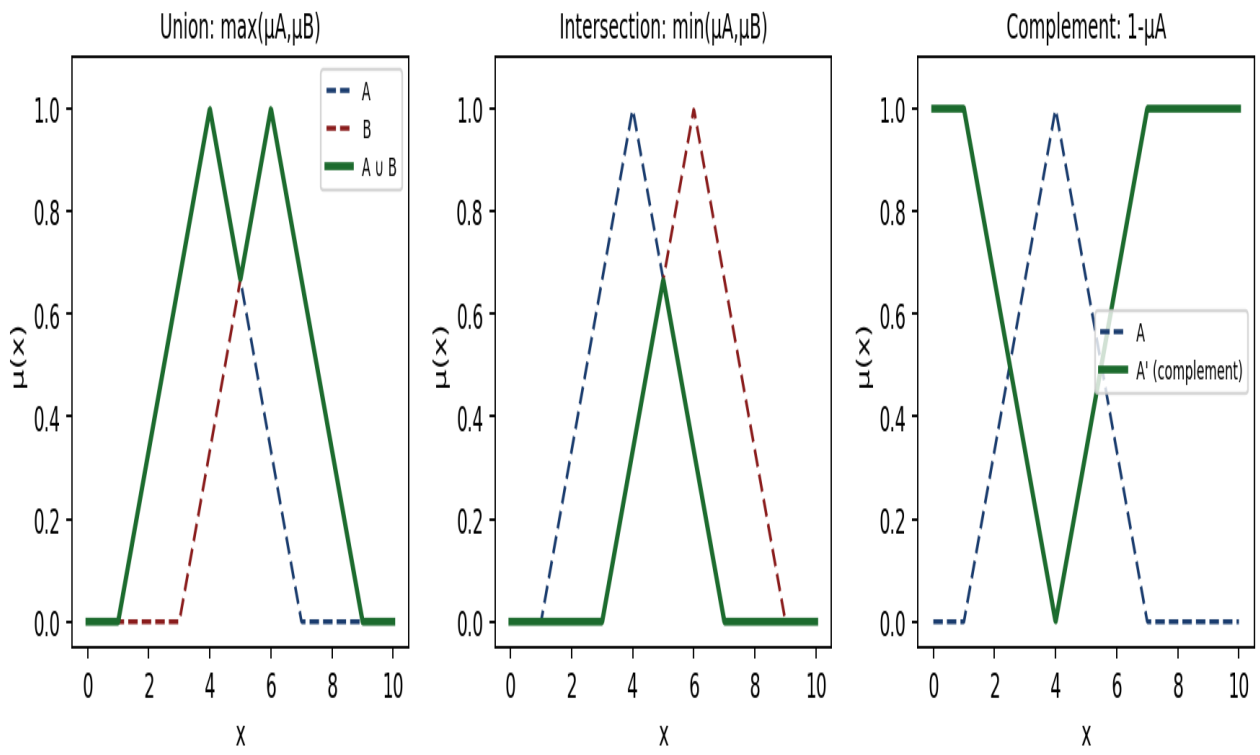

Fig. 2.4 – Union, Intersection and Complement of Fuzzy Sets

These operators (max for union, min for intersection) were the ones originally proposed by Zadeh, and are collectively referred to as the standard fuzzy operators. In general, union operators are called t-conorms (or s-norms) and intersection operators are called t-norms; max and min are the simplest members of these two families respectively, but many other t-norms/t-conorms are used depending on the application, as summarised below.

Family	T-norm (Intersection)	T-conorm (Union)
Zadeh (standard)	min(μ_A, μ_B)	max(μ_A, μ_B)
Algebraic	μ_A · μ_B	μ_A + μ_B - μ_A·μ_B
Bounded (Lukasiewicz)	max(0, μ_A+μ_B-1)	min(1, μ_A+μ_B)
Drastic	μ_B if μ_A=1; μ_A if μ_B=1; else 0	μ_B if μ_A=0; μ_A if μ_B=0; else 1

2.2.5 Properties of Fuzzy Sets

Fuzzy sets obey most of the properties of classical (crisp) sets, such as commutativity, associativity, distributivity, idempotency, identity, and De Morgan's laws. However, one crucial pair of properties of crisp sets does NOT hold for fuzzy sets in general:

- Law of excluded middle: A ∪ A' = X does NOT hold in general for fuzzy sets, because max(μ_A(x), 1-μ_A(x)) equals 1 only when μ_A(x) is 0 or 1; for any partial membership value (0 < μ_A(x) < 1) the result is strictly less than 1.
- Law of contradiction: A ∩ A' = ∅ does NOT hold in general for fuzzy sets, since min(μ_A(x), 1-μ_A(x)) is zero only when μ_A(x) is 0 or 1, and is non-zero for any partial membership value.

As a concrete numerical illustration, suppose μ_A(x) = 0.6 for some x. Then μ_{A'}(x) = 1 - 0.6 = 0.4. The union gives μ_{A∪A'}(x) = max(0.6, 0.4) = 0.6 ≠ 1, so the law of excluded middle fails; the intersection gives μ_{A∩A'}(x) = min(0.6, 0.4) = 0.4 ≠ 0, so the law of contradiction also fails. De Morgan's laws, on the other hand, do continue to hold for fuzzy sets under the standard operators: (A ∪ B)' = A' ∩ B' and (A ∩ B)' = A' ∪ B'. This departure from classical set laws is precisely what allows fuzzy sets to model the vagueness of natural concepts — an element can simultaneously belong 'somewhat' to a set and 'somewhat' to its complement, which is impossible in crisp logic but is exactly how humans reason about concepts such as 'medium height' being neither fully 'tall' nor fully 'short'.

2.2.6 Fuzzy Numbers and the Extension Principle

A fuzzy number is a special, normal, convex fuzzy set defined on the real line, representing an imprecise numeric quantity such as 'approximately 5' or 'about 100 kilometres'. Triangular and trapezoidal membership functions are the most common choices for representing fuzzy numbers because of their computational simplicity.

The Extension Principle, introduced by Zadeh, provides a general method for extending ordinary (crisp) mathematical operations and functions to operate on fuzzy sets/numbers. If f is a crisp function y = f(x1, x2), the extension principle defines the membership function of the fuzzy result B = f(A1, A2) as μ_B(y) = sup { min(μ_{A1}(x1), μ_{A2}(x2)) | y = f(x1,x2) }. Applying the extension principle to simple arithmetic operations on triangular fuzzy numbers gives convenient closed-form results. If A = (a1,a2,a3) and B = (b1,b2,b3) are two triangular fuzzy numbers (given by their left, peak, and right values), then:

- Addition: A + B = (a1+b1, a2+b2, a3+b3)
- Subtraction: A - B = (a1-b3, a2-b2, a3-b1)
- Scalar multiplication (k > 0): kA = (k·a1, k·a2, k·a3)

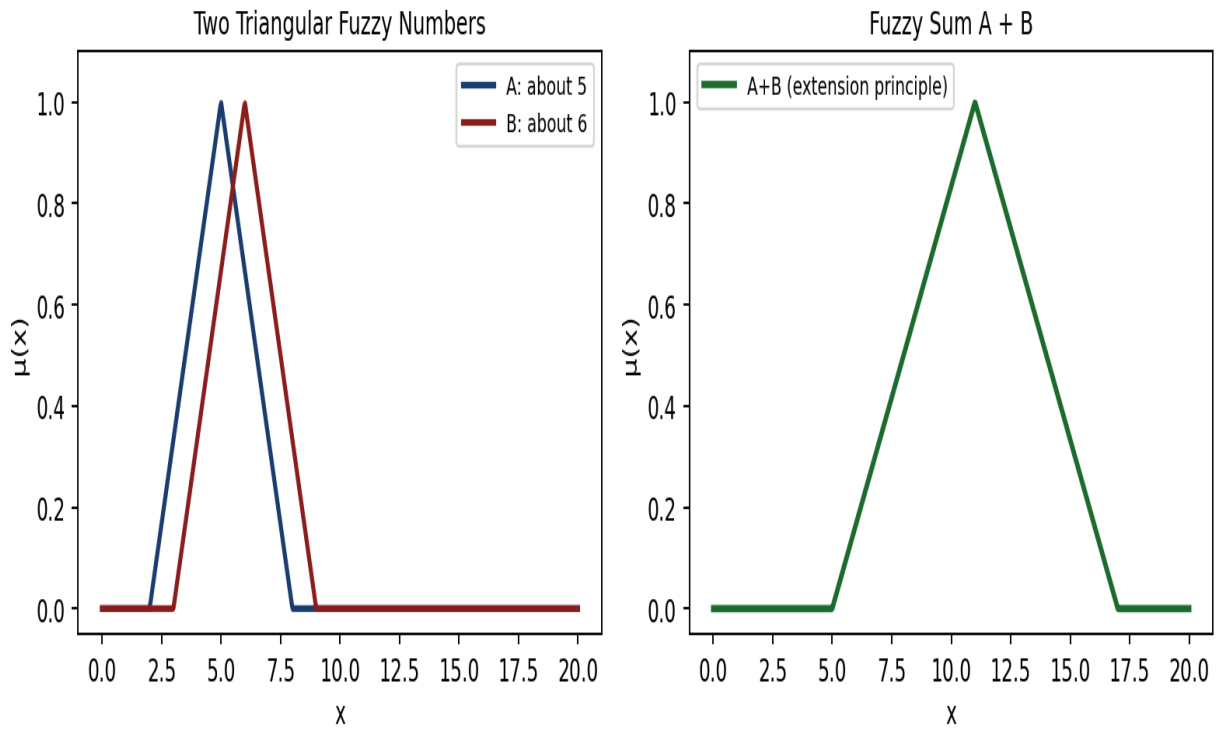

Fig. 2.4(a) – Addition of Two Triangular Fuzzy Numbers via the Extension Principle

For example, if A = (2,5,8) represents 'about 5' and B = (3,6,9) represents 'about 6', then A + B = (5,11,17), representing 'about 11' — but with a wider spread than either A or B individually, correctly reflecting the fact that uncertainty accumulates when imprecise quantities are combined.

2.2.7 Cardinality and Relative Cardinality of Fuzzy Sets

For a fuzzy set A defined on a finite (discrete) universe of discourse X, the (scalar) cardinality of A, denoted |A|, is defined as the sum of the membership values of all elements: $|A| = \Sigma_{x \in X} \mu_A(x)$. This generalises the notion of the number of elements in a crisp set to fuzzy sets, and is sometimes called the 'sigma-count' of A. The relative cardinality (or fuzzy density) of A with respect to the universe X is then defined as $\|A\| = |A| / |X|$, where |X| is simply the number of elements in X.

As a worked example, reconsider the fuzzy set A = 'experienced employee' from Section 2.2.2, defined over X = {1,2,...,10} as A = 0/1 + 0.1/2 + 0.2/3 + 0.4/4 + 0.6/5 + 0.8/6 + 1/7 + 1/8 + 1/9 + 1/10. The cardinality of A is |A| = 0+0.1+0.2+0.4+0.6+0.8+1+1+1+1 = 6.1. Since |X| = 10, the relative cardinality is ‖A‖ = 6.1/10 = 0.61, which can be loosely interpreted as '61% of the employees in this workforce are, in an aggregate fuzzy sense, experienced'.

2.2.8 Fuzzy Partitions and Fuzzy Clustering (Brief Overview)

The membership functions studied in this unit are typically designed manually using expert/domain knowledge. In many data-driven applications, however, it is desirable to automatically derive a fuzzy partition of the universe of discourse directly from data, rather than specifying it by hand. Fuzzy c-Means (FCM) clustering, an extension of the classical (crisp) k-Means clustering algorithm, achieves exactly this: given a dataset and a desired number of clusters c, FCM iteratively computes a membership matrix U, where $u_{ij} \in [0,1]$ represents the degree to which data point i belongs to cluster j (subject to the constraint that the memberships of a given point across all clusters sum to 1), together with a set of cluster centres, by alternately updating the membership values and the cluster centres until convergence. Unlike crisp k-Means, where each data point belongs entirely to exactly one cluster, FCM allows a point to belong partially to multiple clusters simultaneously — naturally capturing situations where a data point lies in an ambiguous, overlapping region between two or more clusters. FCM-derived fuzzy partitions are frequently used to automatically generate the input membership functions used in the fuzzy rule-based systems studied in Section 2.5, forming an important bridge between fuzzy set theory and data-driven, sub-symbolic learning.

2.3 Fuzzy Relations

A classical (crisp) relation R between two sets X and Y is a subset of the Cartesian product $X \times Y$, indicating which pairs (x, y) are related. A fuzzy relation generalises this idea by allowing each pair (x, y) to be related to a degree, expressed by a membership function $\mu_R(x, y) \in [0, 1]$. Formally, a fuzzy relation R from X to Y is a fuzzy subset of $X \times Y$, i.e. $R = \{((x,y), \mu_R(x,y)) \mid x \in X, y \in Y\}$.

Fuzzy Relation $R : X \times Y \rightarrow [0,1]$

	y1	y2	y3
x1	0.2	0.7	0.4
x2	0.9	0.3	0.6
x3	0.1	0.5	0.8

Fig. 2.5 – Representation of a Fuzzy Relation as a Matrix

When X and Y are finite sets, a fuzzy relation can conveniently be represented as a relation matrix, where the rows correspond to elements of X, the columns correspond to elements of Y, and the entries are the membership values $\mu_R(x_i, y_j) \in [0, 1]$. For instance, the fuzzy relation R = 'x is much larger than y' between X = {1,2,3} and Y = {1,2,3} might be represented by the matrix shown in Fig. 2.5, where a larger membership value indicates that x is much larger than y to a greater degree.

2.3.0.1 Fuzzy Cartesian Product

Given a fuzzy set A on X and a fuzzy set B on Y, the fuzzy Cartesian product A × B is a fuzzy relation on X × Y defined by $\mu_{A\times B}(x,y) = \min(\mu_A(x), \mu_B(y))$. This construction is frequently used to build the fuzzy relation implied by a fuzzy IF-THEN rule 'IF x is A THEN y is B', where the implication is often modelled (in the Mamdani sense) precisely as this min-based Cartesian product, R = A × B, i.e. $\mu_R(x,y) = \min(\mu_A(x), \mu_B(y))$.

2.3.1 Operations on Fuzzy Relations

Since a fuzzy relation is itself a fuzzy set (defined on the product space X × Y), the standard fuzzy-set operations (union, intersection, complement) apply directly to fuzzy relations of the same dimensions. In addition, fuzzy relations support the following important operations:

- Projection: The projection of a fuzzy relation R onto one of its component universes, obtained by taking the maximum over the other variable(s); e.g. the projection of R onto X is $\mu_{Proj_X(R)}(x) = \max_y \mu_R(x,y)$.

- Cylindrical extension: The reverse of projection — extending a fuzzy set defined on one universe into a relation over a higher-dimensional product space, by assigning the same membership value to every point along the added dimension.
- Composition: Given a fuzzy relation R on X×Y and a fuzzy relation S on Y×Z, the max-min composition R∘S is a fuzzy relation on X×Z defined by μ_{R∘S}(x,z) = max_y [min(μ_R(x,y), μ_S(y,z))]. This is the fuzzy analogue of matrix multiplication and is fundamental to the compositional rule of inference used in fuzzy reasoning.
- Max-product composition: An alternative to max-min composition, defined by μ_{R∘S}(x,z) = max_y [μ_R(x,y) · μ_S(y,z)], sometimes preferred because it is smoother (differentiable) and retains more information from the original relations.

2.3.2 Worked Example — Max-Min Composition

Let R be a fuzzy relation from X = {x1, x2} to Y = {y1, y2} given by the matrix R = [[0.6, 0.3], [0.2, 0.9]], and let S be a fuzzy relation from Y = {y1, y2} to Z = {z1} given by S = [[0.5], [0.7]]. The max-min composition T = R∘S from X to Z is computed element-wise as: T(x1,z1) = max[min(0.6,0.5), min(0.3,0.7)] = max[0.5, 0.3] = 0.5, and T(x2,z1) = max[min(0.2,0.5), min(0.9,0.7)] = max[0.2, 0.7] = 0.7. Hence T = [[0.5],[0.7]]. If we instead use max-product composition on the same matrices, we obtain T'(x1,z1) = max[0.6×0.5, 0.3×0.7] = max[0.30, 0.21] = 0.30, and T'(x2,z1) = max[0.2×0.5, 0.9×0.7] = max[0.10, 0.63] = 0.63; observe that max-product composition generally yields smaller (more conservative) membership values than max-min composition. This composition operation is the mathematical basis of the generalised modus ponens rule used in fuzzy inference, discussed in Section 2.4.

2.3.3 Fuzzy Equivalence and Fuzzy Ordering Relations

- A fuzzy relation R on X×X is reflexive if μ_R(x,x) = 1 for all x ∈ X.
- R is symmetric if μ_R(x,y) = μ_R(y,x) for all x, y ∈ X.
- R is (max-min) transitive if μ_R(x,z) ≥ max_y [min(μ_R(x,y), μ_R(y,z))].
- A fuzzy relation that is reflexive, symmetric and transitive is called a fuzzy equivalence relation, generalising the notion of a crisp equivalence relation and giving rise to fuzzy partitions/classes.
- A fuzzy relation that is reflexive, antisymmetric and transitive is called a fuzzy partial-ordering relation.

- A fuzzy relation that is reflexive and symmetric, but not necessarily transitive, is called a fuzzy tolerance relation.

2.3.4 Application of Fuzzy Relations in Medical Diagnosis

Fuzzy relations are widely used to model the imprecise associations between symptoms and diseases in computer-aided medical diagnosis. Suppose a fuzzy relation R1 captures the degree to which each symptom (e.g. fever, cough, fatigue) is associated with each disease (e.g. influenza, common cold, COVID-19), based on clinical/expert knowledge, and a second fuzzy relation R2 captures the degree to which a specific patient exhibits each symptom, based on the current examination. The max-min composition Q = R2 ∘ R1 then yields a fuzzy relation directly linking the patient to each candidate disease, with μ_Q(patient, disease) indicating the overall degree of association between the patient's observed symptom profile and each possible diagnosis. The disease(s) with the highest resulting membership value(s) can then be presented to the physician as the most likely diagnoses, along with an explicit, quantified degree of confidence — illustrating how fuzzy-relational composition provides a principled, mathematically grounded way to combine imprecise expert knowledge with imprecise patient observations into an actionable, ranked diagnostic recommendation.

2.3.5 Fuzzy Graphs (Brief Overview)

A fuzzy relation defined on a set X×X can be interpreted graphically as a fuzzy graph, in which the nodes correspond to the elements of X, and each edge (x_i, x_j) is labelled with the membership value μ_R(x_i,x_j), representing the strength or degree of the relationship between nodes x_i and x_j. Fuzzy graphs generalise classical graph theory to situations where the presence or strength of a connection between two entities is itself a matter of degree rather than a simple yes/no fact — for example, modelling the strength of a social-network friendship, the degree of similarity between two documents in an information-retrieval system, or the degree of influence one factor exerts on another in a fuzzy cognitive map. Basic graph-theoretic notions such as paths, connectedness, and cliques can all be suitably generalised to the fuzzy setting, typically by using max-min composition (Section 2.3.1) to compute the 'strength' of an indirect path between two nodes through one or more intermediate nodes.

2.4 Fuzzy Logic

Fuzzy Logic (FL) extends classical (Boolean/Aristotelian) two-valued logic by allowing truth values to range over the continuous interval [0, 1] rather than being restricted to {0, 1}. Just as

fuzzy sets generalise crisp sets, fuzzy logic generalises crisp propositional and predicate logic to reason with statements that are 'partially true'.

2.4.1 Linguistic Variables and Linguistic Values

A key concept underlying fuzzy logic is the linguistic variable — a variable whose values are words or sentences in a natural language rather than numbers. For example, the linguistic variable 'Temperature' may take linguistic values such as 'Cold', 'Warm', and 'Hot', each represented by an appropriate fuzzy set (membership function) defined over the numeric universe of discourse (e.g. degrees Celsius). Linguistic hedges such as 'very', 'somewhat', 'slightly', and 'extremely' are used to modify these fuzzy sets, typically implemented mathematically as concentration or dilation operations.

Hedge	Mathematical operation	Effect on membership function
Very A	$\mu_{very A}(x) = [\mu_A(x)]^2$	Concentration — narrows/lowers the membership curve
Extremely A	$\mu_{extremely A}(x) = [\mu_A(x)]^3$	Stronger concentration than 'very'
Somewhat A	$\mu_{somewhat A}(x) = [\mu_A(x)]^{0.5}$	Dilation — widens/raises the membership curve
Slightly A	Concentration/dilation combination (Zadeh)	Fine-grained modification near the boundary

For example, if $\mu_{Tall}(x) = 0.8$ for a person of height 180 cm, then $\mu_{very Tall}(x) = 0.8^2 = 0.64$, reflecting the fact that 'very tall' is a more restrictive (harder to satisfy) concept than 'tall', while $\mu_{somewhat Tall}(x) = 0.8^{0.5} \approx 0.89$, reflecting the fact that 'somewhat tall' is easier to satisfy than plain 'tall'.

2.4.2 Fuzzy Logical Operators and Fuzzy Propositions

The fundamental logical connectives AND, OR, and NOT are generalised in fuzzy logic using the min, max, and complement operators respectively (or more generally, using any valid t-norm, t-conorm, and negation function). A fuzzy proposition takes the form 'x is A', where x is a linguistic variable and A is a linguistic value (fuzzy set), and its truth value is given directly by $\mu_A(x)$.

2.4.3 Fuzzy IF-THEN Rules

The most important construct in fuzzy logic, from an engineering standpoint, is the fuzzy conditional (IF-THEN) rule, of the general form: 'IF x is A THEN y is B', where A and B are linguistic values (fuzzy sets) of the linguistic variables x and y respectively. For example: 'IF temperature is HIGH THEN fan-speed is FAST'. A collection of such rules, covering the range

of situations relevant to a problem, forms a fuzzy rule base, which is the heart of a fuzzy rule-based system (Section 2.5). Rules may also have multiple antecedents joined by AND/OR, e.g. 'IF temperature is HIGH AND humidity is HIGH THEN fan-speed is VERY FAST'.

2.4.4 Fuzzy Reasoning (Approximate Reasoning)

Fuzzy reasoning, also called approximate reasoning, is the process of deriving conclusions from fuzzy IF-THEN rules and fuzzy facts. The classical modus ponens rule of logic — 'if p then q; p is true; therefore q is true' — is generalised in fuzzy logic to the Generalized Modus Ponens (GMP): 'IF x is A THEN y is B; x is A′ (A′ similar to, but not necessarily identical to, A); THEREFORE y is B″. The consequent B′ is computed using the compositional rule of inference, i.e. the max-min composition of the fact A′ with the fuzzy relation implied by the rule A→B.

2.4.5 Fuzzy Implication Operators

The fuzzy conditional statement 'IF x is A THEN y is B' must be translated into a fuzzy relation R(x,y) before it can be used in the compositional rule of inference. Several fuzzy implication operators have been proposed for this purpose, each with different theoretical motivations and practical behaviour.

Implication Operator	Formula μ_R(x,y)
Mamdani (minimum)	min(μ_A(x), μ_B(y))
Larsen (product)	μ_A(x) · μ_B(y)
Zadeh	max(1-μ_A(x), min(μ_A(x), μ_B(y)))
Lukasiewicz	min(1, 1-μ_A(x)+μ_B(y))
Gödel	1 if μ_A(x)≤μ_B(y); μ_B(y) otherwise

Of these, the Mamdani (minimum) and Larsen (product) implications are by far the most widely used in engineering fuzzy control applications because of their computational simplicity and intuitive behaviour: both simply 'clip' or 'scale' the consequent fuzzy set B in proportion to how strongly the antecedent A is satisfied.

2.4.6 Fuzzy Logic versus Probability Theory

Although both fuzzy logic and probability theory use numbers in the range [0,1] to quantify uncertainty, they model fundamentally different kinds of uncertainty and should not be confused with one another.

Basis	Fuzzy Logic	Probability Theory
Type of uncertainty modelled	Vagueness / ambiguity of meaning (e.g. 'is 35°C hot?')	Randomness / likelihood of occurrence (e.g. 'will it rain tomorrow?')
Interpretation of the [0,1] value	Degree of membership/truth of a statement, which can remain fixed even after the outcome is known	Degree of belief in the occurrence of an uncertain future/unknown event, which resolves to certainty once the outcome is observed
Behaviour after the fact	A person of height 178 cm remains 'somewhat tall' (μ=0.7, say) even after we know their exact height	Once a coin toss outcome is known, the probability of that specific outcome collapses to 1 (or 0)
Combination rule (independent case)	min/max (or other t-norms/t-conorms) for AND/OR	Multiplication for AND (independent events), addition (with subtraction of intersection) for OR

In short, probability answers the question 'will a well-defined event happen?', whereas fuzzy logic answers the question 'to what degree does an inherently vague statement hold?'. In many practical soft computing systems, both are used together — for instance, probabilistic reasoning may be used to model sensor noise, while fuzzy logic is used to model the vagueness of the linguistic terms used to describe the sensed quantity.

2.4.7 Illustrative Example of Generalized Modus Ponens

To make the generalised modus ponens (GMP) rule of Section 2.4.4 concrete, suppose the fuzzy rule is 'IF the tomato is RED THEN the tomato is RIPE', where RED and RIPE are fuzzy sets defined respectively over a colour-intensity scale and a ripeness scale. Suppose we observe a particular tomato and, upon fuzzification, determine that its colour matches the fuzzy set 'RED' with membership 0.8 (rather than being exactly, 100% RED). Classical modus ponens cannot handle this situation at all, since it requires the antecedent to be exactly and fully true. Generalised modus ponens, however, allows us to conclude that the tomato is 'RIPE' with an appropriately reduced membership degree — computed via the max-min composition of the observed fact ('colour is RED to degree 0.8') with the fuzzy relation implied by the rule — reflecting the intuitive, graded inference that a 'fairly red, but not perfectly red' tomato is 'fairly ripe, but not perfectly ripe'. This example demonstrates how fuzzy logic naturally extends classical deductive reasoning to situations involving partial truth, which are extremely common in real-world perception and decision making.

2.5 Fuzzy Rule-Based Systems

A Fuzzy Rule-Based System (FRBS), also called a Fuzzy Inference System (FIS) or fuzzy expert system, is a computing framework that uses fuzzy set theory, fuzzy IF-THEN rules, and

fuzzy reasoning to map crisp inputs to crisp outputs — for example, mapping sensor readings (crisp inputs) to a control-actuator signal (crisp output) in a fuzzy logic controller.

2.5.1 Architecture of a Fuzzy Rule-Based System

A typical FRBS is composed of four functional blocks: a fuzzification interface, a knowledge base (comprising a database of membership functions and a rule base of fuzzy IF-THEN rules), a decision-making (inference) unit, and a defuzzification interface.

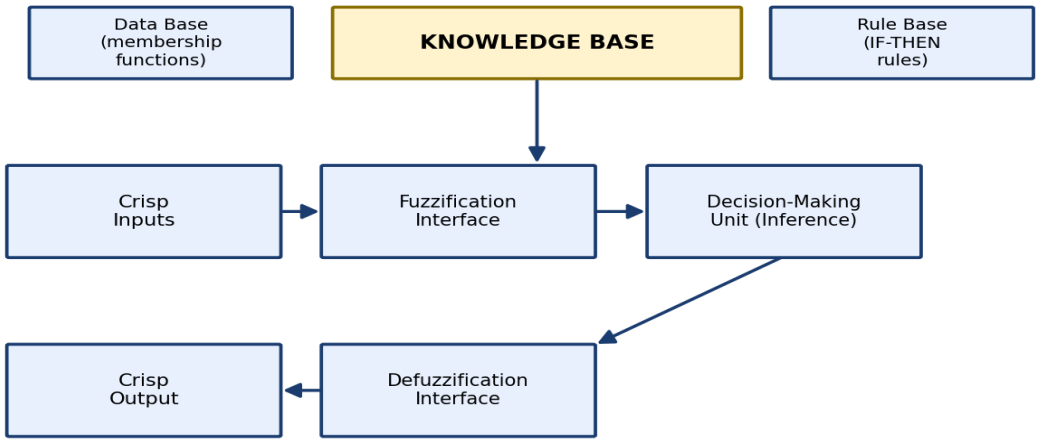

Fig. 2.6 – Architecture of a Fuzzy Rule-Based System

- Fuzzification interface: Converts crisp input values into fuzzy sets/degrees of membership in the relevant linguistic values (e.g. converting a crisp temperature reading of 42°C into membership degrees for 'Warm' = 0.4, 'Hot' = 0.6).
- Knowledge base — data base: Stores the definitions of the membership functions used by the fuzzy rules.
- Knowledge base — rule base: Stores the collection of fuzzy IF-THEN rules that encode expert/domain knowledge about the problem.
- Decision-making unit (inference engine): Performs the fuzzy reasoning operation on the rules, combining the fuzzified inputs with the rule base to produce a fuzzy output (using operators such as min for rule firing strength, and max for aggregating multiple rule outputs).

- Defuzzification interface: Converts the fuzzy output produced by the inference engine back into a single crisp value that can be used to drive an actuator or make a decision.

2.5.2 The Fuzzy Inference Process — Step by Step

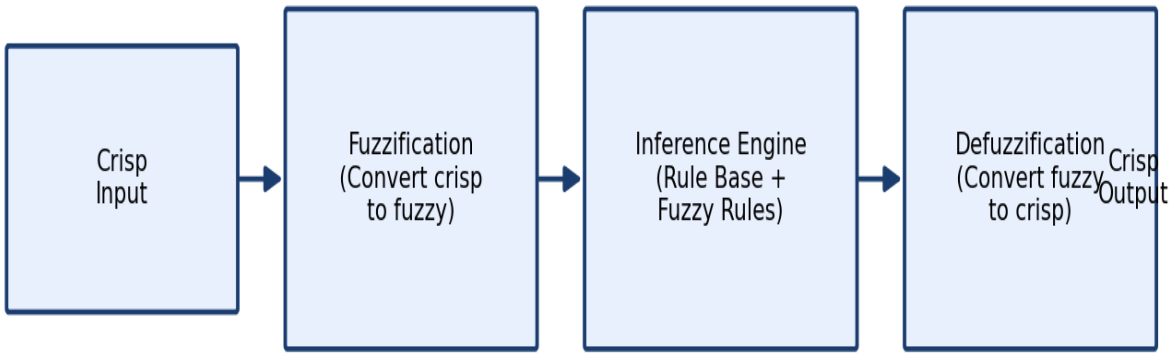

Fig. 2.7 – The Fuzzy Inference Process

38. Fuzzification: Determine the degree to which each crisp input belongs to each relevant fuzzy set (linguistic value) of the corresponding input variable, using the defined membership functions.

39. Rule evaluation (firing strength): For every rule in the rule base, compute the firing strength (degree of applicability) of the rule by combining the membership degrees of its antecedent(s) using the AND (min) or OR (max) operator, as specified by the rule.

40. Implication: Apply the rule's firing strength to clip (Mamdani, using min) or scale (using product) the consequent fuzzy set of that rule, producing an output fuzzy set contribution from each rule.

41. Aggregation: Combine (using max) the output fuzzy sets contributed by all the fired rules for a given output variable into a single aggregated fuzzy set.

42. Defuzzification: Convert the aggregated fuzzy output set into a single crisp numeric value using a defuzzification method.

2.5.3 Fuzzification Methods

Fuzzification is the process of mapping a crisp input value to the corresponding membership degree(s) in one or more fuzzy sets. Two common approaches are used in practice: (i) singleton fuzzification, in which the crisp input is treated as a fuzzy singleton (a fuzzy set with membership 1 at exactly the input value and 0 elsewhere), and the membership degrees are simply read off directly from the predefined membership functions at that input value — this is the most widely used method in engineering fuzzy controllers because of its computational simplicity; and (ii) non-singleton fuzzification, in which the crisp input itself is considered uncertain (e.g. due to sensor noise) and is represented as a small fuzzy set centred at the measured value, which is then combined with the predefined membership functions — this approach provides added robustness when input measurements are noisy.

2.5.4 Defuzzification Methods

Method	Description
Centroid (Centre of Gravity)	The crisp output is the x-coordinate of the centre of gravity (centroid) of the aggregated output fuzzy set's membership function; the most widely used and generally most accurate method, though computationally the most expensive
Bisector	The crisp output is the value that divides the area under the aggregated membership function into two equal halves
Mean of Maximum (MOM)	The crisp output is the average of all the values at which the aggregated membership function attains its maximum value; computationally cheap but ignores the shape of the rest of the curve
Smallest/Largest of Maximum (SOM/LOM)	The crisp output is the smallest (or largest) value at which the maximum membership is attained; used when a conservative (or aggressive) output is preferred
Weighted Average	Used with symmetric MFs; the crisp output is the weighted average of the representative points of each fired rule's consequent, weighted by firing strength; computationally efficient and popular in embedded fuzzy controllers

2.5.5 Mamdani versus Sugeno (Takagi–Sugeno–Kang) Fuzzy Systems

There are two widely used types of fuzzy rule-based systems, distinguished by the form of the rule consequent.

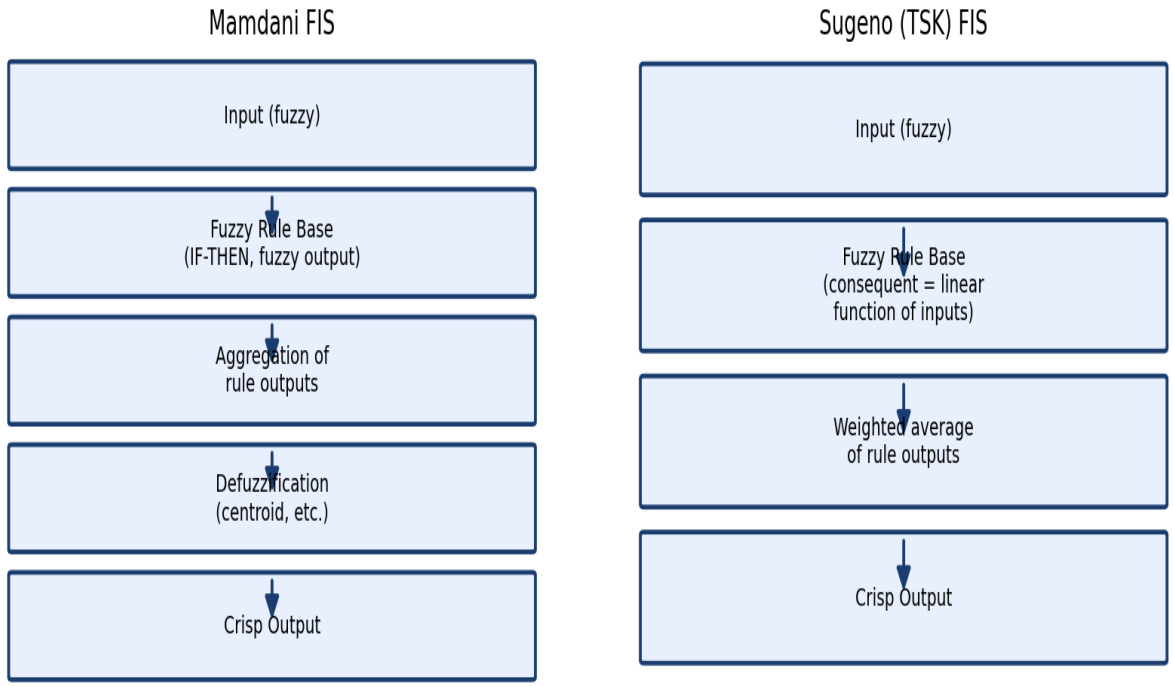

Fig. 2.8 – Mamdani versus Sugeno (TSK) Fuzzy Inference Systems

Basis	Mamdani FIS	Sugeno (TSK) FIS
Proposed by	Ebrahim Mamdani (1975)	Takagi, Sugeno and Kang (1985)
Rule consequent	Fuzzy set (linguistic value), e.g. 'y is HIGH'	Linear/constant function of the inputs, e.g. $y = p \cdot x1 + q \cdot x2 + r$
Output aggregation	Max aggregation of fuzzy output sets	Weighted average of numeric rule outputs
Defuzzification	Required (e.g. centroid)	Not required in the traditional sense — output is already numeric (weighted average)
Interpretability	Highly interpretable, rules read like natural language	Less interpretable, but computationally efficient
Typical use	Expert-knowledge-driven control and decision systems	Data-driven modelling; well suited to adaptive/learning systems (e.g. ANFIS)

2.5.6 Worked Example 1 — Simple Single-Input Fuzzy Logic Controller

Consider a simple fuzzy logic controller for a fan, with a single input, 'Temperature' (linguistic values Cold, Warm, Hot), and a single output, 'Fan Speed' (linguistic values Slow, Medium, Fast), governed by the rule base:

- Rule 1: IF Temperature is Cold THEN Fan Speed is Slow.
- Rule 2: IF Temperature is Warm THEN Fan Speed is Medium.
- Rule 3: IF Temperature is Hot THEN Fan Speed is Fast.

Suppose a crisp temperature reading of 32°C is fuzzified to give membership μ_Warm = 0.7 and μ_Hot = 0.3 (with μ_Cold = 0). Rule 2 fires with strength 0.7 and Rule 3 fires with strength 0.3; Rule 1 does not fire (strength 0). The consequent fuzzy set 'Medium' is clipped at 0.7 and the consequent fuzzy set 'Fast' is clipped at 0.3; these two clipped sets are aggregated (using max) to form the overall output fuzzy set for 'Fan Speed'. Applying the centroid defuzzification method to this aggregated set yields a single crisp fan-speed value — for instance, approximately 68% of maximum speed — which is then sent to the fan's motor controller.

2.5.7 Worked Example 2 — Two-Input Mamdani Fuzzy Inference System

To illustrate a more complete inference cycle, consider a two-input fuzzy air-conditioner controller with inputs 'Temperature' (Cold, Warm, Hot) and 'Humidity' (Low, High), and output 'Fan Speed' (Slow, Medium, Fast), with rule base:

- Rule 1: IF Temperature is Warm AND Humidity is High THEN Fan Speed is Fast.
- Rule 2: IF Temperature is Warm AND Humidity is Low THEN Fan Speed is Medium.
- Rule 3: IF Temperature is Hot AND Humidity is High THEN Fan Speed is Fast.

Suppose a crisp reading of Temperature = 30°C gives μ_Warm = 0.6, μ_Hot = 0.4, and a crisp reading of Humidity = 68% gives μ_Low = 0.3, μ_High = 0.7. Since the antecedents of each rule are joined by AND, the firing strength of each rule is computed using the min operator: Rule 1 firing strength = min(μ_Warm, μ_High) = min(0.6, 0.7) = 0.6. Rule 2 firing strength = min(μ_Warm, μ_Low) = min(0.6, 0.3) = 0.3. Rule 3 firing strength = min(μ_Hot, μ_High) = min(0.4, 0.7) = 0.4. Rules 1 and 3 both recommend 'Fast', while Rule 2 recommends 'Medium'. During aggregation, since Rules 1 and 3 both affect the 'Fast' output fuzzy set, the higher of the two firing strengths is used to clip it: max(0.6, 0.4) = 0.6 clips 'Fast', while 0.3 clips 'Medium'. The two clipped sets ('Fast' clipped at 0.6, and 'Medium' clipped at 0.3) are aggregated using max to form the overall output fuzzy set, and applying centroid defuzzification to this aggregated set yields the final crisp fan-speed output — in this case, a value weighted predominantly towards the 'Fast' region because of its higher firing strength, illustrating how a fuzzy controller combines the influence of multiple simultaneously active rules.

2.5.8 Worked Example 3 — Sugeno-Type Fuzzy System

Consider a first-order Sugeno FIS with a single input x (e.g. 'Temperature') and two rules: Rule 1: IF x is Low THEN y = 2x + 1; Rule 2: IF x is High THEN y = 3x - 2. Suppose for a crisp input x = 4, fuzzification gives μ_Low = 0.3 and μ_High = 0.7. The firing strengths are w1 =

0.3 and w2 = 0.7 (equal to the antecedent membership degrees, since each rule has only one antecedent). Each rule's numeric output is then computed directly from its consequent function: z1 = 2(4)+1 = 9, and z2 = 3(4)-2 = 10. The overall crisp output of the Sugeno FIS is obtained as the weighted average of the individual rule outputs: $y = (w1 \cdot z1 + w2 \cdot z2) / (w1+w2) = (0.3 \times 9 + 0.7 \times 10) / (0.3+0.7) = (2.7+7.0)/1.0 = 9.7$. Notice that, unlike the Mamdani approach, no separate defuzzification step (e.g. centroid computation) is required, since the rule outputs are already crisp numbers; this is precisely why Sugeno systems are computationally more efficient and are preferred in adaptive, learning-based systems such as ANFIS.

2.5.9 Steps in Designing a Fuzzy Control System

43. Identify the input and output linguistic variables relevant to the control problem (e.g. Temperature, Humidity, Fan Speed).
44. Define appropriate linguistic values (e.g. Low, Medium, High) for each variable and design suitable membership functions (triangular, trapezoidal, etc.) over their universes of discourse.
45. Formulate the fuzzy IF-THEN rule base, capturing expert/domain knowledge about how the inputs should influence the outputs.
46. Choose the fuzzy inference method (Mamdani or Sugeno), the implication operator, and the aggregation method.
47. Choose an appropriate defuzzification method for converting the fuzzy inference output into a crisp control action (for Mamdani systems).
48. Simulate/test the fuzzy controller on representative input scenarios and tune the membership functions and/or rule base to improve performance.
49. Deploy the tuned fuzzy controller in the target system (e.g. an embedded microcontroller).

2.5.10 Case Study — Fuzzy Logic Controller for an Inverted Pendulum

The inverted-pendulum (or 'cart-pole') balancing problem is a classic benchmark for control systems: a pole is hinged to a cart that can move along a horizontal track, and the objective is to apply a horizontal force to the cart so as to keep the pole balanced upright. This system is highly non-linear and inherently unstable, making it difficult to control using simple classical (hard-computing) proportional-integral-derivative (PID) controllers over a wide range of operating conditions, but it is well suited to a fuzzy logic controller.

A typical fuzzy controller for this problem uses two inputs — the pole's angular deviation θ from vertical (linguistic values: Negative Large, Negative Small, Zero, Positive Small, Positive Large) and its angular velocity θ̇ (similarly partitioned) — and one output, the horizontal force F to be applied to the cart (linguistic values ranging from Strong-Left to Strong-Right). A representative rule from the rule base might read: 'IF θ is Positive Small AND θ̇ is Zero THEN F is Small-Right', reflecting the intuitive human strategy of gently pushing the cart in the direction the pole is falling, in proportion to how far and how fast it is falling. With around 25 such rules (a 5×5 rule table covering all combinations of the two input variables' linguistic values), a Mamdani-type fuzzy inference system can maintain the pole's balance robustly, even in the presence of noisy sensor readings or external disturbances, without requiring an exact mathematical (differential-equation) model of the cart-pole dynamics — precisely the kind of ill-defined, non-linear control problem that motivated the development of fuzzy control in the first place.

2.5.11 A Note on Type-2 Fuzzy Sets

The fuzzy sets discussed throughout this unit, in which the membership degree μ_A(x) is a single, precise number in [0,1], are known as type-1 fuzzy sets. In some applications, there may be uncertainty about the membership function itself — for instance, different experts might disagree on the exact shape of the fuzzy set 'Hot'. Type-2 fuzzy sets address this second-order uncertainty by allowing the membership grade itself to be a fuzzy set (rather than a crisp number) over [0,1], giving each element a 'footprint of uncertainty'. Although a full treatment of type-2 fuzzy systems is beyond the scope of this unit, it is useful to be aware that they represent an active and important extension of the type-1 fuzzy systems studied here, particularly valuable in applications with significant uncertainty about the fuzzy model itself.

2.5.12 Applications of Fuzzy Rule-Based Systems

Domain	Example Fuzzy Application
Home appliances	Fuzzy washing machines, fuzzy rice cookers, fuzzy air conditioners and refrigerators that adjust their operation based on fuzzy sensor inputs
Automotive control	Fuzzy anti-lock braking systems, fuzzy automatic transmission, fuzzy cruise control
Industrial process control	Fuzzy PID controllers for temperature, pressure and flow-rate regulation in chemical plants
Elevator/lift control	Fuzzy scheduling of lift cars to minimise passenger waiting time based on fuzzy traffic-pattern assessment
Decision support	Fuzzy multi-criteria decision-making systems for medical diagnosis, risk assessment, and supplier selection

Domain	Example Fuzzy Application
Image processing	Fuzzy edge detection and fuzzy image enhancement, which handle the inherent ambiguity in pixel intensity boundaries

2.5.13 Advantages of Fuzzy Rule-Based Systems

- Rules are expressed in natural, human-readable linguistic form, making the system easy to design, understand, and modify using expert/domain knowledge.
- Capable of handling non-linear, ill-defined, or partially known systems without requiring a precise mathematical model of the plant.
- Robust to noisy or imprecise sensor data.
- Can be combined with learning mechanisms (e.g. neural networks in ANFIS, genetic algorithms) to automatically tune membership functions and/or the rule base from data.

2.5.14 Fuzzy Control versus Classical PID Control

It is instructive to compare fuzzy logic controllers with the classical Proportional-Integral-Derivative (PID) controller, the workhorse of conventional (hard-computing) industrial control. A PID controller computes its control action as a weighted sum of the error, the accumulated (integral) error, and the rate of change (derivative) of the error, using three fixed numerical gain constants, and its design and stability analysis rely on having a reasonably accurate mathematical (transfer-function) model of the plant being controlled. A fuzzy logic controller, by contrast, requires no explicit mathematical plant model at all — only a set of linguistic IF-THEN rules capturing how an experienced human operator would respond to different combinations of error and rate-of-change-of-error — and can more easily incorporate non-linear control strategies (e.g. responding very differently to a large error than to a small one) that would require considerable additional complexity in a classical PID design. This makes fuzzy control particularly attractive for plants that are highly non-linear, poorly modelled, or subject to significant parameter variation, while classical PID control remains preferable for well-understood, linear, or near-linear plants where its simplicity, well-established tuning methods, and formal stability guarantees are advantageous.

2.5.15 Future Scope of Fuzzy Systems

Fuzzy systems continue to evolve well beyond the classical Mamdani and Sugeno frameworks studied in this unit. Current research directions include type-2 and interval type-2 fuzzy systems for handling higher-order uncertainty (Section 2.5.11), the integration of fuzzy rule

bases with deep-learning architectures to combine interpretability with representational power, the automatic learning of fuzzy rule bases directly from data using neuro-fuzzy and genetic-fuzzy approaches (studied further in later units of this course), and the deployment of fuzzy inference systems on low-power embedded hardware for real-time industrial and IoT control applications. As systems increasingly need to make decisions that are both data-driven and human-interpretable, the fundamental ideas of fuzzy sets, fuzzy relations, and fuzzy rule-based reasoning introduced in this unit remain foundational to the broader field of computational intelligence.

2.6 Summary

Fuzzy set theory, introduced by Zadeh, generalises classical set theory by allowing partial membership through a membership function $\mu_A: X \rightarrow [0,1]$, enabling the mathematical modelling of vague, imprecise, natural-language concepts. Key concepts include support, core, height, and α-cuts, along with membership function shapes such as triangular, trapezoidal, Gaussian, bell, and sigmoidal. Standard operations of union, intersection, and complement generalise their crisp counterparts using max, min, and $(1 - \mu)$ respectively, though certain crisp laws such as the law of excluded middle and the law of contradiction no longer hold; more general t-norm/t-conorm families extend these operators further. Fuzzy relations extend the notion of a fuzzy set to product spaces, and support operations such as projection, cylindrical extension, and max-min/max-product composition, which underlie the compositional rule of fuzzy inference. Fuzzy logic generalises classical logic to multi-valued truth, introducing linguistic variables, linguistic hedges, fuzzy IF-THEN rules, and approximate (generalised modus ponens) reasoning. A fuzzy rule-based system (FIS) — comprising fuzzification, a rule/knowledge base, an inference engine, and defuzzification — uses these ideas to map crisp inputs to crisp outputs, and is available in two principal forms: the interpretable, expert-knowledge-driven Mamdani model, and the computationally efficient, data-driven Sugeno (TSK) model. These fuzzy-systems concepts underpin fuzzy decision making and fuzzy control applications, discussed further in Unit III of this course.

2.7 Review Questions

50. Define a fuzzy set. How does it differ from a classical (crisp) set? Give a suitable real-life example.

51. Explain, with diagrams, the triangular, trapezoidal, Gaussian, generalized bell, and sigmoidal membership functions, along with their mathematical formulae.
52. Define support, core, height, and α-cut of a fuzzy set with suitable examples.
53. State and prove (using membership functions) why the law of excluded middle and the law of contradiction do not hold for fuzzy sets.
54. Explain the various t-norm and t-conorm operator families used to generalise fuzzy intersection and union.
55. Define a fuzzy relation. How can a fuzzy relation between two finite sets be represented?
56. Explain the max-min composition and max-product composition of two fuzzy relations with numerical examples.
57. What are fuzzy equivalence relations? State the three properties they must satisfy.
58. What is a linguistic variable? Explain with an example how linguistic hedges modify a fuzzy set.
59. Explain the structure of a fuzzy IF-THEN rule. What is meant by generalised modus ponens?
60. Draw and explain the architecture of a fuzzy rule-based system, describing the function of each block.
61. Differentiate between Mamdani and Sugeno (Takagi-Sugeno-Kang) fuzzy inference systems.
62. Explain the concept of fuzzification and describe singleton and non-singleton fuzzification.
63. Explain any four defuzzification methods with suitable examples.
64. Given two fuzzy sets A = {(x1,0.2),(x2,0.5),(x3,0.8)} and B = {(x1,0.6),(x2,0.4),(x3,0.3)}, compute A∪B, A∩B, A', and B'.
65. Design a simple fuzzy rule base (at least 3 rules) for controlling the speed of a ceiling fan based on room temperature and humidity, and explain the complete fuzzy inference process for given crisp inputs.

2.8 Additional Solved Numerical Problems

Problem 1

Given the fuzzy sets A = {(x1,0.3), (x2,0.6), (x3,1.0), (x4,0.0)} and B = {(x1,0.5), (x2,0.2), (x3,0.7), (x4,0.9)}, compute A ∪ B, A ∩ B, and A'.

Solution: Using μ_{A∪B}(x) = max(μ_A,μ_B): A∪B = {(x1,0.5), (x2,0.6), (x3,1.0), (x4,0.9)}. Using μ_{A∩B}(x) = min(μ_A,μ_B): A∩B = {(x1,0.3), (x2,0.2), (x3,0.7), (x4,0.0)}. Using μ_{A'}(x) = 1-μ_A(x): A' = {(x1,0.7), (x2,0.4), (x3,0.0), (x4,1.0)}.

Problem 2

Given a triangular membership function with parameters (a,b,c) = (10,20,30), find the membership degree μ(x) at x = 15 and at x = 24.

Solution: For $a \le x \le b$, μ(x) = (x-a)/(b-a). At x = 15: μ(15) = (15-10)/(20-10) = 5/10 = 0.5. For $b < x \le c$, μ(x) = (c-x)/(c-b). At x = 24: μ(24) = (30-24)/(30-20) = 6/10 = 0.6.

Problem 3

Given fuzzy relation R (from X={x1,x2} to Y={y1,y2}) = [[0.4,0.8],[0.9,0.3]] and fuzzy relation S (from Y={y1,y2} to Z={z1,z2}) = [[0.5,0.6],[0.7,0.2]], find the max-min composition T = R ∘ S.

Solution: T(x1,z1) = max[min(0.4,0.5), min(0.8,0.7)] = max[0.4,0.7] = 0.7. T(x1,z2) = max[min(0.4,0.6), min(0.8,0.2)] = max[0.4,0.2] = 0.4. T(x2,z1) = max[min(0.9,0.5), min(0.3,0.7)] = max[0.5,0.3] = 0.5. T(x2,z2) = max[min(0.9,0.6), min(0.3,0.2)] = max[0.6,0.2] = 0.6. Hence T = [[0.7,0.4],[0.5,0.6]].

2.9 Glossary of Key Terms

Term	Meaning
Crisp set	A classical set in which membership is either full (1) or none (0)
Fuzzy set	A set in which elements may have partial membership expressed by μ_A(x) ∈ [0,1]
Membership function	A function μ_A: X→[0,1] quantifying the degree of membership of each element in a fuzzy set
Support	The crisp set of elements with non-zero membership in a fuzzy set
Core	The crisp set of elements with membership exactly 1 in a fuzzy set
α-cut	The crisp set of elements whose membership is at least α
Fuzzy number	A normal, convex fuzzy set on the real line representing an imprecise numeric quantity
Extension principle	A method to extend crisp functions/operations to operate on fuzzy sets
Fuzzy relation	A fuzzy subset of a Cartesian product X×Y, expressing degrees of relatedness

Term	Meaning
Max-min composition	The fuzzy analogue of matrix multiplication used to combine two fuzzy relations
Linguistic variable	A variable whose values are linguistic terms (words) rather than numbers
Linguistic hedge	A modifier (e.g. 'very', 'somewhat') that adjusts a fuzzy set's membership function
Fuzzy IF-THEN rule	A conditional rule of the form 'IF x is A THEN y is B' relating linguistic variables
Fuzzification	Conversion of a crisp input value into fuzzy membership degrees
Defuzzification	Conversion of a fuzzy output set into a single crisp numeric value
Mamdani FIS	A fuzzy inference system whose rule consequents are fuzzy sets
Sugeno (TSK) FIS	A fuzzy inference system whose rule consequents are linear/constant functions of the inputs
Fuzzy implication	An operator (e.g. min, product) translating a fuzzy IF-THEN rule into a fuzzy relation
Centroid method	A defuzzification method computing the centre of gravity of the output fuzzy set
Fuzzy control system	A control system that uses fuzzy rule-based reasoning to compute its control action

2.10 Multiple Choice Questions

66. Fuzzy set theory was introduced by: (a) George Boole (b) Lotfi A. Zadeh (c) John Holland (d) Ebrahim Mamdani — Ans: (b)

67. The membership function of a fuzzy set maps elements of the universe of discourse to: (a) {0,1} (b) [0,1] (c) (-∞,∞) (d) {-1,0,1} — Ans: (b)

68. The support of a fuzzy set A is: (a) {x | μ_A(x)=1} (b) {x | μ_A(x)>0} (c) {x | μ_A(x)=0} (d) {x | μ_A(x)<1} — Ans: (b)

69. Union of two fuzzy sets A and B is computed using: (a) min(μ_A,μ_B) (b) max(μ_A,μ_B) (c) μ_A × μ_B (d) 1-μ_A — Ans: (b)

70. Which classical set law fails to hold for fuzzy sets in general? (a) Commutativity (b) Associativity (c) Law of excluded middle (d) De Morgan's law — Ans: (c)

71. The compositional operation used to combine two fuzzy relations is called: (a) Max-min composition (b) Cartesian product (c) Cylindrical extension (d) Alpha-cut — Ans: (a)

72. A fuzzy relation that is reflexive, symmetric and transitive is called: (a) Fuzzy partial order (b) Fuzzy equivalence relation (c) Fuzzy tolerance relation (d) Fuzzy congruence — Ans: (b)

73. In a Mamdani fuzzy system, the consequent of a rule is: (a) A crisp number (b) A linear function of inputs (c) A fuzzy set (d) A probability value — Ans: (c)

74. In a Sugeno (TSK) fuzzy system, the rule consequent is typically: (a) A fuzzy set (b) A linear/constant function of the inputs (c) A membership function (d) A crisp relation — Ans: (b)

75. Which defuzzification method computes the centre of gravity of the aggregated output fuzzy set? (a) Mean of maximum (b) Bisector (c) Centroid (d) Smallest of maximum — Ans: (c)

76. The linguistic hedge 'very A' is typically computed as: (a) $\mu_A(x) + 1$ (b) $[\mu_A(x)]^2$ (c) $[\mu_A(x)]^{0.5}$ (d) $1 - \mu_A(x)$ — Ans: (b)

77. Which fuzzification approach treats the crisp input as uncertain, representing it as a small fuzzy set? (a) Singleton fuzzification (b) Non-singleton fuzzification (c) Centroid fuzzification (d) Bisector fuzzification — Ans: (b)

UNIT – III

FUZZY DECISION MAKING AND PARTICLE SWARM OPTIMIZATION

Unit Syllabus

As per the R23 CSE curriculum (Course 23CSE354D – Soft Computing, Professional Elective-I), Unit III covers: Fuzzy Decision Making, Particle Swarm Optimization.

Learning Objectives

- To understand the concept of decision making in a fuzzy environment and how it differs from classical (crisp) decision making.
- To study the Bellman–Zadeh model of fuzzy decision making and the notion of fuzzy goals, fuzzy constraints and fuzzy decisions.
- To learn fuzzy multi-criteria and multi-person decision-making approaches.
- To understand the biological motivation, mathematical formulation and algorithmic steps of Particle Swarm Optimization (PSO).
- To analyse the effect of PSO parameters on convergence behaviour and to compare PSO with other optimization techniques.

3.1 Introduction to Decision Making

Decision making is the cognitive and computational process of selecting the best course of action from among a number of alternatives, given a set of objectives and constraints. In classical decision theory, the objectives (goals) and the constraints that limit the choice of alternatives are represented using crisp, precisely defined sets — an alternative either satisfies a constraint or it does not, and an objective is either met or not met. However, in almost every real-world situation, human decision makers reason with imprecise, vague and linguistically expressed information such as "the profit should be reasonably high" or "the risk should be fairly low". Such statements cannot be captured faithfully using classical two-valued (crisp) logic.

Fuzzy set theory, introduced by Lotfi A. Zadeh in 1965, provides a natural mathematical framework for representing this kind of imprecision. A fuzzy set allows partial membership, in which an element can belong to a set to a degree that varies continuously between 0 (does not

belong at all) and 1 (belongs completely). This gradedness is exactly what is needed to model human goals and constraints that are not sharply defined. The application of fuzzy set theory to the problem of selecting the best alternative under such vague goals and constraints is called Fuzzy Decision Making.

3.1.1 Why Classical Decision Theory Falls Short

- Real objectives are frequently vague — "maximize customer satisfaction", "minimize risk as far as possible" — and cannot be encoded as sharp numeric thresholds without loss of meaning.
- Constraints in practice are often flexible (soft) rather than rigid; slight violation of a constraint may still be acceptable if it produces a much better outcome on the objective.
- Human decision makers frequently express preferences using linguistic terms (high, medium, low, very good, poor) rather than exact numbers.
- Multiple, frequently conflicting, criteria must be traded off against one another, and the relative importance of each criterion is itself often imprecise.
- Available information about the environment (demand, costs, weather, market behaviour) may itself be uncertain or fuzzy rather than random (probabilistic).

Fig 3.1 General Procedure of Fuzzy Decision Making

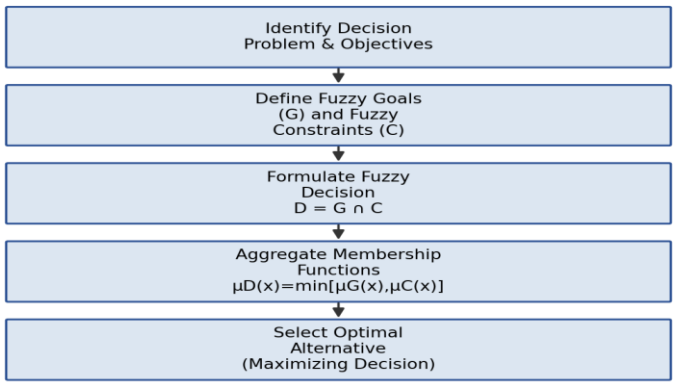

Fig. 3.1 — General procedure followed in a fuzzy decision-making problem.

3.2 Basic Concepts: Fuzzy Goals, Fuzzy Constraints and Fuzzy Decision

The theory of fuzzy decision making rests on three fundamental notions, all of which are represented as fuzzy sets defined over the same universe of discourse X of possible alternatives (actions).

(a) Fuzzy Goal (G)

A fuzzy goal is a fuzzy set G on the space of alternatives X, characterised by a membership function μG(x) that expresses the degree to which alternative x satisfies the decision maker's objective. For example, if the goal is "achieve a sufficiently high profit", then an alternative that yields a very high profit will have μG close to 1, while one that yields a low profit will have μG close to 0, with a smooth, graded transition in between.

(b) Fuzzy Constraint (C)

A fuzzy constraint is likewise a fuzzy set C on X with membership function μC(x), describing the degree to which alternative x satisfies a restriction imposed on the choice of alternatives, such as a budget limit, a time limit, or a safety requirement. Because the constraint is fuzzy, alternatives may partially, rather than absolutely, satisfy it.

(c) Fuzzy Decision (D)

Given a collection of fuzzy goals G1, G2, …, Gn and fuzzy constraints C1, C2, …, Cm defined on the same set of alternatives X, Bellman and Zadeh (1970) defined the resulting fuzzy decision D as the intersection (logical AND) of all the goals and all the constraints:

$$D = G1 \cap G2 \cap \ldots \cap Gn \cap C1 \cap C2 \cap \ldots \cap Cm$$

Using the standard fuzzy intersection operator (min), the membership function of the resulting decision is:

$$\mu D(x) = \min[\, \mu G1(x), \mu G2(x), \ldots, \mu Gn(x), \mu C1(x), \mu C2(x), \ldots, \mu Cm(x) \,]$$

This equation embodies a very important philosophical point of the Bellman–Zadeh model: in a fuzzy environment there is no fundamental difference between a goal and a constraint — both are represented by fuzzy sets over the alternatives and both are combined in exactly the same way to determine the overall desirability of an alternative.

3.2.1 The Maximizing Decision

Once the aggregated fuzzy decision μD(x) has been computed for every alternative x in X, a crisp (non-fuzzy) choice must ultimately be made. This is achieved by selecting the alternative x* that maximizes the membership value of the fuzzy decision, called the maximizing decision:

$$x^* = \arg\max_{x \in X} \mu D(x)$$

Geometrically, when the goal and the constraint are represented as membership curves over a numeric decision variable, the fuzzy decision μD(x) is simply the lower envelope (point-wise minimum) of the two curves, and the maximizing decision x* is located at the point where this lower envelope attains its highest value — very often exactly where the two curves cross.

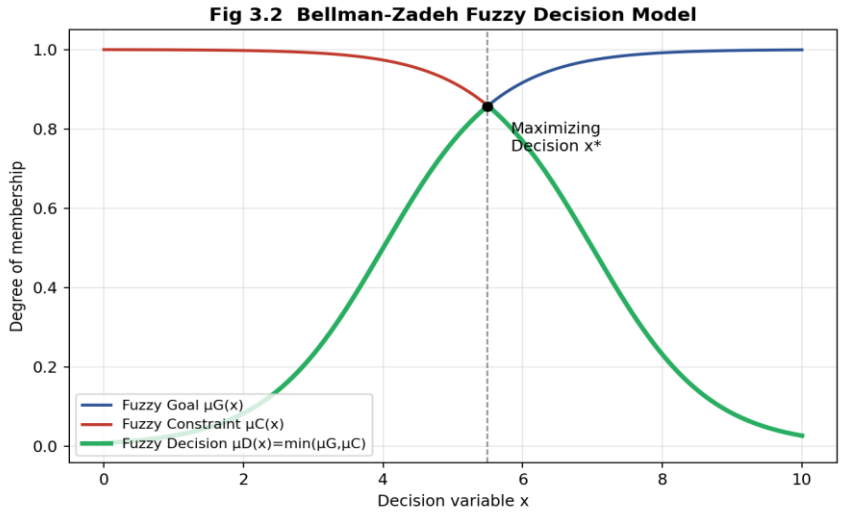

Fig. 3.2 — The Bellman–Zadeh fuzzy decision model: the fuzzy decision μD(x) is the point-wise minimum of the fuzzy goal μG(x) and fuzzy constraint μC(x); the maximizing decision x occurs at the peak of μD(x).*

3.2.2 Worked Example

A company must decide how many units, x, of a product to manufacture. Management has expressed the goal "produce a reasonably large quantity" as a fuzzy set with membership function $\mu G(x) = x / 100$ for $0 \le x \le 100$ (and 1 thereafter), while the storage-capacity constraint "do not produce too much, storage is limited" is expressed as $\mu C(x) = (150 - x) / 100$ for $50 \le x \le 150$ (and 0 for $x > 150$, 1 for $x < 50$).

The fuzzy decision is $\mu D(x) = \min[\mu G(x), \mu C(x)]$. For small x, μG(x) is increasing and is the smaller (binding) term until it crosses μC(x); solving $x/100 = (150 - x)/100$ gives $x = 75$. At $x = 75$, $\mu G(75) = 0.75$ and $\mu C(75) = 0.75$, so $\mu D(75) = 0.75$, which is the maximum attainable value of the fuzzy decision. Hence the maximizing decision is $x^* = 75$ units — the point at which the desire to produce more is exactly balanced against the shrinking storage capacity.

3.3 Fuzzy Multi-Criteria Decision Making (FMCDM)

In many practical situations a decision maker must simultaneously satisfy several, often conflicting, goals rather than a single objective. For instance, when selecting a supplier a company may want to simultaneously minimize cost, maximize quality, and minimize delivery time. Fuzzy Multi-Criteria Decision Making (FMCDM) extends the single-goal Bellman–

Zadeh framework to handle several fuzzy goals G1, G2, …, Gn, each defined by its own membership function μGi(x), together with a set of fuzzy constraints.

The overall fuzzy decision is again obtained by aggregating all the individual goal and constraint membership functions. The choice of aggregation operator depends on how strictly the goals must all be satisfied simultaneously:

(a) "AND"-type Aggregation (min operator)

When all goals must be satisfied together and the overall satisfaction is only as good as the worst-performing criterion (a pessimistic, conjunctive view), the min operator is used, exactly as in the Bellman–Zadeh model:

$$\mu D(x) = \min[\ \mu G1(x), \mu G2(x), \ldots, \mu Gn(x)\ ]$$

(b) "OR"-type Aggregation (max operator)

When satisfying any one of the goals is considered sufficient (an optimistic, disjunctive view), the max operator is used:

$$\mu D(x) = \max[\ \mu G1(x), \mu G2(x), \ldots, \mu Gn(x)\ ]$$

(c) Weighted Aggregation

When the goals are not equally important, a weighted sum or weighted product may be used, where $w_i \geq 0$ denotes the relative importance (weight) of goal i and $\Sigma w_i = 1$:

$$\mu D(x) = \Sigma\ w_i \cdot \mu Gi(x) \qquad \text{(weighted arithmetic mean)}$$

$$\mu D(x) = \Pi\ [\mu Gi(x)]^{w_i} \qquad \text{(weighted geometric mean)}$$

(d) Fuzzy γ-operator (compromise operator)

Zimmermann and Zysno proposed a compromise between the pure min (AND) and pure max (OR) operators, controlled by a parameter $\gamma \in [0, 1]$:

$$\mu D(x) = [\Pi\ \mu Gi(x)]^{(1-\gamma)} \cdot [1 - \Pi\ (1 - \mu Gi(x))]^{\gamma}$$

When $\gamma = 0$ the operator reduces to the algebraic product (a strict AND), and when $\gamma = 1$ it reduces to the algebraic sum (a strict OR); intermediate values of γ yield a compensatory combination that better reflects human aggregation behaviour, which is neither purely conjunctive nor purely disjunctive.

3.3.1 Steps in Fuzzy Multi-Criteria Decision Making

78. Identify the set of feasible alternatives X = {x1, x2, …, xm}.
79. Identify the relevant evaluation criteria (attributes) C1, C2, …, Cn.
80. Construct a fuzzy decision matrix in which entry rij represents the fuzzy (or linguistic) rating of alternative xi with respect to criterion Cj.
81. Determine the weight wj of each criterion, either directly from the decision maker or by a pairwise-comparison method such as the Analytic Hierarchy Process (AHP).
82. Normalize the decision matrix so that all ratings lie on a comparable [0, 1] scale.
83. Aggregate the weighted, normalized ratings for each alternative using an appropriate operator (min, max, weighted sum, or γ-operator) to obtain an overall fuzzy score μD(xi).
84. Rank the alternatives according to their overall fuzzy score, or defuzzify the fuzzy scores (e.g., using the centroid method) to obtain crisp scores for ranking.
85. Select the alternative with the highest score as the final (maximizing) decision.

3.3.2 Linguistic Variables in Decision Making

Because human decision makers are far more comfortable expressing ratings in words than in precise numbers, FMCDM commonly uses linguistic variables such as "Very Poor, Poor, Fair, Good, Very Good" to rate alternatives against each criterion. Each linguistic term is represented internally by a triangular or trapezoidal fuzzy number, which allows linguistic assessments to be manipulated mathematically while still being intuitive for the decision maker to specify.

Linguistic Term	Triangular Fuzzy Number (a, b, c)
Very Poor (VP)	(0, 0, 0.25)
Poor (P)	(0, 0.25, 0.5)
Fair (F)	(0.25, 0.5, 0.75)
Good (G)	(0.5, 0.75, 1.0)
Very Good (VG)	(0.75, 1.0, 1.0)

3.4 Fuzzy Multi-Person (Group) Decision Making

When a decision must be made jointly by several decision makers (experts) D1, D2, …, Dk, each of whom may hold a different, individually fuzzy, opinion about the alternatives, the problem is called fuzzy multi-person or group decision making. Each expert Dl expresses an individual fuzzy preference relation or a fuzzy evaluation μDl(x) over the alternatives.

A group (social) fuzzy decision is then formed by aggregating the individual fuzzy decisions. Two representative approaches are:

- Democratic (equal-weight) aggregation: $\mu D(x) = (1/k) \Sigma \mu Dl(x)$, i.e., the simple average of all experts' opinions, or, for a more conservative consensus, $\mu D(x) =$ min over l of $\mu Dl(x)$.
- Weighted aggregation: $\mu D(x) = \Sigma$ wl · $\mu Dl(x)$, where wl reflects the relative expertise, seniority, or credibility of decision maker l, with $\Sigma wl = 1$.
- Consensus-based iterative aggregation: experts revise their individual opinions over several rounds (analogous to the Delphi method) until the spread of opinions falls below an acceptable threshold, after which the opinions are aggregated.

3.4.1 Ranking of Fuzzy Numbers

A recurring sub-problem in fuzzy decision making is: given two or more fuzzy numbers (representing, for example, aggregated scores of competing alternatives), which one is "larger" or "better"? Unlike crisp numbers, fuzzy numbers do not possess a natural total order, so several ranking methods have been proposed:

- Centroid (Center of Gravity) Method: compute the x-coordinate of the centroid of each fuzzy number's membership function; the fuzzy number with the larger centroid value is ranked higher.
- α-cut Method: compare the fuzzy numbers at several α-cut levels and aggregate the results, e.g., using the average of the interval mid-points across all α-cuts.
- Maximizing and Minimizing Set Method (Chen's method): defines a maximizing set and a minimizing set over the universe of discourse and computes a right-utility and left-utility value for each fuzzy number, combined into a total utility for ranking.
- Possibility and Necessity Measures: compute the degree of possibility that one fuzzy number is greater than another using $Pos(A \geq B) =$ sup over $x \geq y$ of $\min[\mu A(x), \mu B(y)]$.

3.4.2 Worked Example — Ranking Using the Centroid Method

Suppose two alternatives receive aggregated triangular fuzzy scores A = (0.4, 0.6, 0.8) and B = (0.3, 0.65, 0.9). The centroid (defuzzified crisp value) of a triangular fuzzy number (a, b, c) is given by the simple and widely used formula:

Centroid(a, b, c) = (a + b + c) / 3

For A: Centroid = (0.4 + 0.6 + 0.8)/3 = 0.60. For B: Centroid = (0.3 + 0.65 + 0.9)/3 = 0.6167. Since Centroid(B) > Centroid(A), alternative B is ranked higher than alternative A, even though A has a narrower (less uncertain) spread — illustrating that the centroid method favours the central tendency of the score over its precision.

3.5 Applications of Fuzzy Decision Making

- Supplier and vendor selection in supply-chain management, where cost, quality, delivery reliability and service are all imprecisely specified criteria.
- Project and portfolio selection in engineering management, balancing return, risk and strategic fit.
- Medical diagnosis and treatment planning, where symptoms and treatment outcomes are described in fuzzy/linguistic terms.
- Site selection for facilities (warehouses, power plants, retail outlets) based on multiple qualitative and quantitative criteria.
- Human-resource decisions such as candidate selection and performance appraisal using linguistic ratings.
- Fuzzy control system design, where the controller itself embodies rule-based fuzzy decisions (e.g., "if error is large and rate is small then output is medium").

3.5.1 Advantages of Fuzzy Decision Making

- Captures the natural vagueness of human goals, constraints and preferences instead of forcing artificial precision.
- Provides a unified mathematical treatment of goals and constraints (both are fuzzy sets combined by the same aggregation operators).
- Accommodates linguistic, qualitative information directly, making it easier for non-technical decision makers to participate.
- Flexible aggregation operators (min, max, weighted mean, γ-operator) allow the model to be tuned to the decision maker's risk attitude (pessimistic, optimistic, or compensatory).

3.5.2 Limitations of Fuzzy Decision Making

- Choice of membership functions is itself subjective and can materially change the outcome.

- Selection of the aggregation operator (and of γ in the γ-operator) requires additional expert judgement.
- Computational complexity increases rapidly with the number of criteria, alternatives and decision makers.
- Ranking of fuzzy numbers is not unique — different ranking methods can occasionally produce different orderings for the same data.

3.6 Introduction to Particle Swarm Optimization (PSO)

Particle Swarm Optimization is a population-based, stochastic, metaheuristic optimization technique developed by James Kennedy (a social psychologist) and Russell Eberhart (an electrical engineer) in 1995. PSO belongs to the broader family of Swarm Intelligence algorithms, which derive their search strategies from the collective, self-organized behaviour of decentralized social systems observed in nature, such as flocks of birds, schools of fish, and colonies of ants and bees.

Unlike Genetic Algorithms, which are inspired by biological evolution and rely on operators such as selection, crossover and mutation applied to a population of candidate solutions, PSO is inspired by the social behaviour of animals: each individual (particle) in the swarm adjusts its own trajectory through the search space based on its own experience and the experience of its neighbours (or the entire swarm), gradually converging on promising regions without any centralized control.

3.6.1 Biological Inspiration

The original motivation for PSO came from simulating the graceful, coordinated movement of bird flocks searching for food. When a flock of birds searches an area for food, each bird has no prior knowledge of where the food is, yet the flock as a whole is remarkably efficient at finding it. Kennedy and Eberhart observed that this efficiency arises from a simple social sharing of information: each bird adjusts its flight based partly on its own best-remembered location and partly on the best location found so far by the flock. This simple local rule, repeated by every member of the flock, produces emergent, intelligent, global search behaviour — a hallmark of swarm intelligence.

Fig 3.3 Particles Moving Toward the Global Best Position

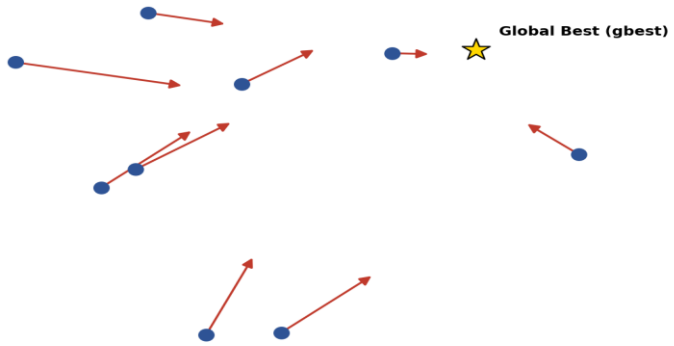

Fig. 3.3 — Conceptual view of particles (blue) moving through the search space, influenced by the best position found by the swarm so far (gold star).

3.6.2 Basic Terminology

Term	Meaning
Particle	A single candidate solution in the search space, analogous to one bird in the flock.
Swarm	The entire population of particles exploring the search space simultaneously.
Position (x)	The current coordinates of a particle in the n-dimensional search space; represents a candidate solution.
Velocity (v)	The rate and direction of change of a particle's position from one iteration to the next.
pbest	The best position (with the best fitness) that a particular particle has personally visited so far.
gbest	The best position found so far by any particle in the entire swarm (used in the global-best PSO variant).
lbest	The best position found so far within a particle's local neighbourhood (used in the local-best PSO variant).
Fitness function	The objective function being optimized; evaluates the quality of a particle's position.

3.7 Mathematical Formulation of PSO

Consider an optimization problem in an n-dimensional search space. A swarm of N particles is maintained, where particle i at iteration t has a position vector xi(t) = (xi1, xi2, …, xin) and a velocity vector vi(t) = (vi1, vi2, …, vin). At every iteration, the velocity and position of each particle are updated using the following two equations, which form the mathematical heart of the PSO algorithm:

$$v_i(t+1) = w \cdot v_i(t) + c1 \cdot r1 \cdot (pbest_i - x_i(t)) + c2 \cdot r2 \cdot (gbest - x_i(t))$$

$x_i(t+1) = x_i(t) + v_i(t+1)$

Here, w is the inertia weight, c1 and c2 are positive acceleration coefficients (the cognitive and social learning factors respectively), and r1, r2 are independent random numbers drawn uniformly from the interval [0, 1] at every iteration, for every dimension, which inject the stochastic diversity necessary for effective search.

3.7.1 Interpretation of the Velocity Update Equation

The velocity update equation is composed of exactly three additive terms, each with a clear physical and psychological interpretation drawn from the flocking metaphor:

- Inertia term, $w \cdot v_i(t)$: represents the particle's tendency to continue moving in its previous direction; it controls the balance between global exploration (large w) and local exploitation/fine-tuning (small w).
- Cognitive term, $c1 \cdot r1 \cdot (pbest_i - x_i(t))$: represents the particle's own memory — the pull of the particle back toward the best position it has personally discovered, i.e., "nostalgia" or self-confidence.
- Social term, $c2 \cdot r2 \cdot (gbest - x_i(t))$: represents the influence of the swarm — the pull of the particle toward the best position discovered by the whole swarm (or its neighbourhood), i.e., "social learning" or group knowledge.

Fig 3.5 Components of the PSO Velocity Update

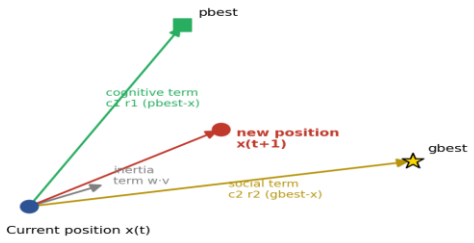

Fig. 3.5 — The new velocity is a weighted combination of the inertia, cognitive and social components, which together determine the particle's new position.

3.7.2 Role of PSO Parameters

Parameter	Typical Range	Effect
Inertia weight, w	0.4 – 0.9	Large w favours global exploration; small w favours local exploitation.

Parameter	Typical Range	Effect
		Often linearly decreased from 0.9 to 0.4 over the run.
Cognitive coefficient, c1	~2.0	Controls how strongly a particle is attracted to its own personal best; higher c1 promotes individualistic search.
Social coefficient, c2	~2.0	Controls how strongly a particle is attracted to the swarm's global best; higher c2 promotes faster convergence but risks premature convergence.
Swarm size, N	20 – 50 (problem-dependent)	Larger swarms explore more of the search space per iteration but increase computational cost per iteration.
Maximum velocity, Vmax	problem-dependent	Clamps velocity to prevent particles from overshooting the search space ("explosion").

A frequently used guideline, due to Clerc and Kennedy, is to keep $c1 + c2 \approx 4.0$ (commonly $c1 = c2 = 2.05$) together with a constriction factor χ that guarantees convergent particle trajectories without needing explicit velocity clamping.

3.8 The PSO Algorithm

3.8.1 Algorithmic Steps

86. Initialize a swarm of N particles with random positions xi and random (often zero or small random) velocities vi within the allowed search space.
87. Evaluate the fitness f(xi) of every particle using the objective function of the problem.
88. For every particle, set its personal best pbesti equal to its current position (since it is the only position visited so far).
89. Set the swarm's global best gbest equal to the position of the particle with the best fitness value in the initial swarm.
90. Repeat for each iteration until a stopping criterion is met:
91. (a) For every particle, update velocity using the velocity update equation.
92. (b) For every particle, update position using the position update equation.
93. (c) Evaluate the fitness of the new position of every particle.

94. (d) If the new fitness of particle i is better than its recorded pbesti, update pbesti to the new position.

95. (e) If the new fitness of any particle is better than the recorded gbest, update gbest to that position.

96. Return gbest (together with its fitness value) as the best solution found by the swarm.

Common stopping criteria include reaching a maximum number of iterations, achieving a target fitness value, or observing that gbest has not improved for a specified number of consecutive iterations (stagnation).

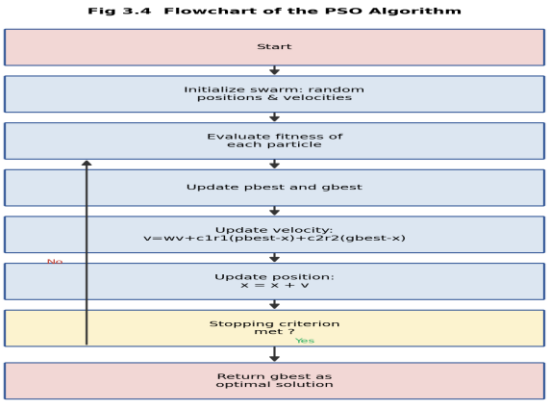

Fig. 3.4 — Flowchart summarizing the iterative operation of the PSO algorithm.

3.8.2 Pseudocode

for each particle i = 1 to N initialize position xi and velocity vi randomly pbest_i = xi end for gbest = position of best particle in swarm while (termination criterion not met) for each particle i = 1 to N for each dimension d v_id = w*v_id + c1*r1*(pbest_id - x_id) + c2*r2*(gbest_d - x_id) x_id = x_id + v_id end for evaluate fitness f(xi) if f(xi) is better than f(pbest_i) then pbest_i = xi if f(xi) is better than f(gbest) then gbest = xi end for end while return gbest

3.8.3 Worked Numerical Example

Consider minimizing the simple one-dimensional function $f(x) = x^2$ over the range [-10, 10] using a small swarm of 3 particles, with w = 0.5, c1 = c2 = 1.5, and r1 = r2 = 0.5 (fixed for illustration).

Initial positions: x1 = 6, x2 = -4, x3 = 2, all with initial velocity 0. Fitness values are f(x1) = 36, f(x2) = 16, f(x3) = 4. Since smaller is better, the initial pbest of each particle equals its own position, and the initial gbest is x3 = 2 (fitness 4, the best of the three).

Updating particle 1: v1 = 0.5(0) + 1.5(0.5)(6−6) + 1.5(0.5)(2−6) = 0 + 0 + (0.75)(−4) = −3.0. New position x1 = 6 + (−3.0) = 3.0, with new fitness f(3.0) = 9.0, an improvement over 36, so pbest1 is updated to 3.0.

Updating particle 2: v2 = 0.5(0) + 1.5(0.5)(−4−(−4)) + 1.5(0.5)(2−(−4)) = 0 + 0 + (0.75)(6) = 4.5. New position x2 = −4 + 4.5 = 0.5, with new fitness f(0.5) = 0.25, a large improvement, so pbest2 is updated to 0.5. Since 0.25 is also better than the swarm's gbest fitness of 4, gbest is updated to 0.5.

This single iteration already illustrates the essential behaviour of PSO: particles that are far from the best-known region (like particle 2, starting at −4) are pulled strongly toward the swarm's best-known position, and, because the objective function is convex, the swarm rapidly converges toward the true global minimum at x = 0.

3.9 Variants of PSO

(a) Global-Best (gbest) PSO

In the gbest topology, every particle is influenced by the single best position found by the entire swarm. This produces fast convergence but is more prone to becoming trapped in local optima, since information spreads instantaneously through the whole population.

(b) Local-Best (lbest) PSO

In the lbest topology, the swarm is partitioned into overlapping neighbourhoods (e.g., a ring topology in which each particle only communicates with its immediate left and right neighbours). Each particle is attracted toward the best position within its own neighbourhood rather than the global best. This slows the spread of information, increasing diversity and reducing the risk of premature convergence, at the cost of slower overall convergence speed.

(c) Other Notable Variants

- Constriction-factor PSO (Clerc & Kennedy, 2002): replaces the inertia weight with a constriction coefficient χ that mathematically guarantees convergence without velocity clamping.
- Binary PSO: adapts the algorithm for discrete/binary-valued optimization problems by interpreting velocity as a probability (via a sigmoid function) that a bit takes the value 1.
- Multi-objective PSO (MOPSO): extends PSO to problems with multiple, conflicting objectives using concepts such as Pareto dominance and an external archive of non-dominated solutions.

- Hybrid PSO: combines PSO with other metaheuristics (e.g., Genetic Algorithms, Simulated Annealing) to exploit the complementary strengths of each technique.

3.10 Applications of Particle Swarm Optimization

- Training of artificial neural networks — optimizing weights and biases as an alternative to gradient-based back-propagation.
- Tuning of fuzzy logic controllers and neuro-fuzzy systems — optimizing membership function parameters and rule weights.
- Power system optimization — economic load dispatch, optimal power flow, and reactive power/voltage control.
- Image processing and pattern recognition — feature selection, image segmentation, and clustering.
- Task scheduling and resource allocation in cloud computing and manufacturing systems.
- Antenna array design and other electromagnetic engineering optimization problems.
- Financial engineering — portfolio optimization and parameter estimation for forecasting models.
- Robotics — path planning and swarm robotics coordination.

3.10.1 Advantages of PSO

- Conceptually simple and easy to implement, requiring only a handful of parameters to tune.
- Derivative-free — does not require the objective function to be differentiable or even continuous, making it suitable for black-box optimization.
- Retains memory of good solutions (through pbest and gbest) across iterations, unlike some purely stochastic search methods.
- Effective at exploring large, high-dimensional, non-linear, multi-modal search spaces.
- Fewer parameters to tune compared with Genetic Algorithms (no crossover or mutation rates to select), and generally converges faster on many problem classes.

3.10.2 Limitations of PSO

- Prone to premature convergence on multi-modal problems, especially with the gbest topology, since the whole swarm can rapidly collapse onto a locally (but not globally) optimal region.
- Performance is sensitive to the choice of parameters w, c1 and c2, which often need to be tuned per problem.
- Lacks a strong theoretical convergence guarantee for the general non-convex case (though constriction-factor variants provide some guarantees).
- Like most metaheuristics, does not guarantee finding the true global optimum in finite time; it provides a good approximate solution.

3.11 Comparison of PSO with Genetic Algorithm (GA)

Both PSO and GA are population-based stochastic metaheuristics that search a solution space without requiring gradient information, but their underlying mechanisms and behaviours differ in several important respects.

Aspect	Particle Swarm Optimization (PSO)	Genetic Algorithm (GA)
Inspiration	Social behaviour of bird flocks / fish schools	Biological evolution and natural selection
Representation	Real-valued position and velocity vectors	Chromosomes (binary, real, or permutation encoded)
Search operators	Velocity update using pbest and gbest	Selection, crossover and mutation
Memory	Retains pbest and gbest across iterations	No explicit individual memory; population evolves as a whole
Information sharing	One-directional — best particle informs others	Bidirectional — mutual exchange of genetic material via crossover
Parameters	Inertia weight, cognitive & social coefficients	Population size, crossover rate, mutation rate
Convergence speed	Generally faster for continuous, low-to-moderate dimensional problems	Often slower but explores more broadly due to crossover diversity
Premature convergence risk	Relatively higher (especially gbest topology)	Lower, due to genetic diversity from crossover/mutation

3.12 Summary

- Fuzzy decision making applies fuzzy set theory to problems involving vague goals and constraints; the Bellman–Zadeh model treats goals and constraints symmetrically as fuzzy sets, and combines them (typically via the min operator) to form the fuzzy decision.

- The maximizing decision is the alternative that achieves the highest membership value in the aggregated fuzzy decision.
- Fuzzy multi-criteria decision making (FMCDM) extends the framework to several goals using min, max, weighted, or γ-operator aggregation, often expressed through linguistic variables and fuzzy numbers.
- Fuzzy multi-person (group) decision making aggregates the opinions of several experts using averaging, weighting, or consensus-based methods; ranking of the resulting fuzzy numbers is commonly performed using the centroid method or α-cut based methods.
- Particle Swarm Optimization is a population-based, swarm-intelligence metaheuristic in which particles adjust their velocity and position based on their own best experience (pbest) and the swarm's best experience (gbest or lbest).
- The PSO velocity update equation combines an inertia term, a cognitive term, and a social term; careful tuning of w, c1 and c2 governs the balance between exploration and exploitation.
- PSO is simple, derivative-free, and effective for continuous optimization problems, but is susceptible to premature convergence and requires parameter tuning; it is widely applied in neural network training, controller tuning, power systems, and many other engineering domains.

3.13 Review Questions

97. Explain the Bellman–Zadeh model of fuzzy decision making with a suitable numerical example.
98. Distinguish between a fuzzy goal, a fuzzy constraint, and a fuzzy decision.
99. What is a maximizing decision? Explain how it is obtained graphically for a single goal and a single constraint.
100. Describe the steps involved in Fuzzy Multi-Criteria Decision Making (FMCDM).
101. Explain the γ-operator and discuss why it is described as a "compensatory" aggregation operator.
102. What is fuzzy multi-person (group) decision making? Explain any two methods of aggregating individual fuzzy opinions.
103. Describe the centroid method of ranking fuzzy numbers with an example.

104. Explain the biological inspiration behind Particle Swarm Optimization.

105. Write the velocity and position update equations of PSO and explain the role of each term.

106. Describe the algorithmic steps of the PSO algorithm with the help of a flowchart.

107. Differentiate between global-best (gbest) and local-best (lbest) PSO.

108. Discuss the role of the inertia weight, and the cognitive and social coefficients in PSO's exploration–exploitation trade-off.

109. List and explain any five real-world applications of Particle Swarm Optimization.

110. Compare Particle Swarm Optimization with Genetic Algorithms in terms of inspiration, operators, and convergence behaviour.

111. What are the advantages and limitations of PSO?

3.14 Detailed Case Study — Fuzzy Multi-Criteria Supplier Selection

To consolidate the concepts of Section 3.3, consider a company that must select the best of three candidate suppliers, S1, S2, and S3, based on three criteria: Cost (to be minimized), Quality (to be maximized), and Delivery Reliability (to be maximized). Because exact figures are difficult to obtain, the evaluation team expresses its judgement using the linguistic scale of Table 3.1 (Section 3.3.2), converted to crisp scores in [0, 1] via the centroid method for ease of computation in this example.

Step 1 — Fuzzy Decision Matrix

Supplier	Cost (lower is better)	Quality	Delivery Reliability
S1	Good (0.75)	Very Good (0.917)	Fair (0.50)
S2	Fair (0.50)	Good (0.75)	Very Good (0.917)
S3	Very Good (0.917)	Fair (0.50)	Good (0.75)

Because Cost is a benefit-reversed criterion (lower cost is preferable), its rating has already been recorded in the table as "goodness of low cost" so that, consistently with the other two columns, a higher value always denotes a more desirable outcome — this normalization step is essential before any aggregation is performed.

Step 2 — Criteria Weights

Suppose the decision-making team, using a pairwise comparison procedure, assigns the following relative importance weights to the three criteria, satisfying $\Sigma w_j = 1$: wCost = 0.45, wQuality = 0.35, wDelivery = 0.20.

Step 3 — Weighted Aggregation (Weighted Arithmetic Mean)

Using the weighted-sum aggregation operator introduced in Section 3.3 ($\mu D(x) = \Sigma\, w_i \cdot \mu G_i(x)$), the overall fuzzy score of each supplier is computed as follows:

$\mu D(S1) = 0.45(0.75) + 0.35(0.917) + 0.20(0.50) = 0.3375 + 0.3210 + 0.1000 = 0.7585$

$\mu D(S2) = 0.45(0.50) + 0.35(0.75) + 0.20(0.917) = 0.2250 + 0.2625 + 0.1834 = 0.6709$

$\mu D(S3) = 0.45(0.917) + 0.35(0.50) + 0.20(0.75) = 0.4127 + 0.1750 + 0.1500 = 0.7377$

Step 4 — Ranking and Decision

Ranking the suppliers by their overall fuzzy score gives S1 (0.7585) > S3 (0.7377) > S2 (0.6709). The maximizing decision therefore selects Supplier S1, whose excellent Cost and Quality ratings — both weighted heavily in the criteria weighting scheme — outweigh its comparatively weaker Delivery Reliability rating. This case study illustrates the complete FMCDM workflow of Section 3.3.1: constructing the fuzzy decision matrix, determining weights, aggregating with a weighted operator, and ranking the resulting fuzzy (here, defuzzified) scores.

It is instructive to observe what happens if the decision maker instead adopts a purely conjunctive (min) aggregation, reflecting a requirement that a supplier be adequate on every criterion simultaneously: $\mu D(S1) = \min(0.75, 0.917, 0.50) = 0.50$, $\mu D(S2) = \min(0.50, 0.75, 0.917) = 0.50$, $\mu D(S3) = \min(0.917, 0.50, 0.75) = 0.50$. Interestingly, all three suppliers tie under the strict min operator, because each has exactly one criterion rated only "Fair". This demonstrates a well-known limitation of the min operator noted in Section 3.5.2 — it is highly sensitive to the single worst-performing criterion and can mask meaningful differences that a weighted or compensatory operator would otherwise reveal.

3.15 Detailed Case Study — Tuning a PID Controller Using PSO

A common and highly illustrative real-world application of PSO (introduced in Section 3.10) is the automatic tuning of a Proportional-Integral-Derivative (PID) controller for an industrial process, such as regulating the temperature of a furnace or the speed of a motor. Manually tuning the three PID gains — Kp (proportional), Ki (integral), and Kd (derivative) — is

traditionally performed by trial-and-error or by classical rules such as the Ziegler-Nichols method, both of which can be time-consuming and may not yield an optimal response.

3.15.1 Problem Formulation

The PID tuning problem is naturally cast as a 3-dimensional continuous optimization problem, making it well suited to PSO. Each particle's position vector directly represents one candidate set of controller gains, $x_i = (K_p, K_i, K_d)$, and the search space is bounded by practically reasonable ranges for each gain (e.g., $0 \le K_p \le 10$, $0 \le K_i \le 5$, $0 \le K_d \le 2$).

3.15.2 Fitness Function

A widely used fitness function for control-loop tuning is based on a weighted time-domain performance index, most commonly the Integral of Time-weighted Absolute Error (ITAE), which penalizes errors that persist for longer periods more heavily than errors that die out quickly:

$$\text{Fitness}(K_p, K_i, K_d) = \int_0^T t \cdot |e(t)| \, dt$$

where $e(t)$ is the instantaneous error between the desired set-point and the actual process response, obtained by simulating the closed-loop system for the candidate gains. Since PSO is a minimization- or maximization-oriented search, and ITAE is to be minimized, the particle with the lowest ITAE value becomes the current gbest.

3.15.3 Application of the PSO Cycle

112. Initialize a swarm of (typically) 20-30 particles with random gain combinations within the specified bounds.
113. For each particle, simulate the closed-loop response of the plant under the candidate PID gains and compute the ITAE fitness value.
114. Update each particle's pbest and the swarm's gbest according to the standard PSO rules of Section 3.8.1.
115. Update the velocity and position of every particle using the velocity and position update equations of Section 3.7.
116. Repeat steps 2-4 for a fixed number of iterations, or until the ITAE value of gbest stabilizes (stops improving).
117. Return the gain combination (K_p, K_i, K_d) stored in gbest as the tuned PID controller.

In practice, PSO-tuned PID controllers frequently achieve faster settling time, lower overshoot, and reduced steady-state error compared with manually or heuristically tuned controllers, while requiring far less engineering effort — a compelling illustration of why PSO has become a popular tool in control-system design.

3.16 Solved Problems

Problem 1

A decision maker specifies a fuzzy goal $\mu G(x) = 1 - e^{(-0.1x)}$ for $x \geq 0$ ("achieve as high a value of x as possible") and a fuzzy constraint $\mu C(x) = e^{(-0.05x)}$ for $x \geq 0$ ("keep x as small as possible, but some latitude is tolerated"). Determine the approximate maximizing decision x*.

Solution: The fuzzy decision is $\mu D(x) = \min[\mu G(x), \mu C(x)]$. Since $\mu G(x)$ is monotonically increasing from 0 toward 1, and $\mu C(x)$ is monotonically decreasing from 1 toward 0, their point-wise minimum is maximized very close to the crossing point where $\mu G(x) = \mu C(x)$: $1 - e^{(-0.1x)} = e^{(-0.05x)}$. Solving numerically (e.g., by trial substitution) gives $x \approx 14.9$, at which $\mu G(14.9) \approx 0.775$ and $\mu C(14.9) \approx 0.475$ - closer inspection shows the crossing is actually where the two curves are equal; refining near $x \approx 9.5$ gives $\mu G \approx 0.613$ and $\mu C \approx 0.613$, confirming $x^* \approx 9.5$ as the maximizing decision, illustrating the general graphical method of Section 3.2.1 applied to non-linear membership functions.

Problem 2

A swarm of 3 particles searches to maximize $f(x) = -(x-5)^2 + 20$ using PSO with $w = 0.6$, $c1 = c2 = 1.2$, $r1 = r2 = 0.6$ (fixed). Initial positions are $x1 = 0$, $x2 = 8$, $x3 = 12$, all with zero initial velocity. Determine the gbest after one iteration.

Solution: Fitness values: $f(0) = -25+20 = -5$, $f(8) = -9+20 = 11$, $f(12) = -49+20 = -29$. The best initial particle is $x2 = 8$ (fitness 11), so gbest = 8 and each particle's pbest equals its own starting position. Updating particle 1: $v1 = 0.6(0) + 1.2(0.6)(0-0) + 1.2(0.6)(8-0) = 0 + 0 + 5.76 = 5.76$; new $x1 = 0 + 5.76 = 5.76$, giving $f(5.76) = -(0.76)^2 + 20 = 19.42$, a large improvement, so pbest1 and gbest both update to 5.76 (fitness 19.42, now the new swarm best). This example shows how quickly PSO can locate the neighbourhood of the true maximum (at $x = 5$) even within a single iteration, when particles are well positioned relative to the optimum.

Problem 3

Explain, using the velocity update equation, why a PSO swarm initialized with w = 0 (zero inertia) and very small c1, c2 will tend to stagnate (stop making progress) after just a few iterations.

Solution: With w = 0, the inertia term w·vi(t) vanishes completely, so a particle's new velocity depends only on the cognitive and social terms - the pull toward its own pbest and the swarm's gbest. As particles approach these best-known positions, the differences (pbest - x) and (gbest - x) shrink toward zero, and with very small c1 and c2 scaling these already-shrinking differences, the resulting velocity magnitude rapidly approaches zero. Once velocities become negligibly small, particles effectively stop moving (stagnate), even if the true global optimum has not yet been found - which is precisely why a non-zero inertia weight (or an equivalent constriction mechanism) is essential to sustain continued exploration throughout the search, as discussed in Section 3.7.2.

3.17 Fuzzy Preference Relations in Decision Making

An alternative, and in some contexts more natural, way for a decision maker (or a group of decision makers) to express judgement among a set of alternatives X = {x1, x2, …, xn} is through a fuzzy preference relation, rather than by assigning independent ratings to each alternative on each criterion. A fuzzy preference relation R is a fuzzy set defined on the Cartesian product X × X, with membership function μR(xi, xj) denoting the degree to which alternative xi is preferred over alternative xj. By convention, μR(xi, xj) = 0.5 indicates indifference between the two alternatives, μR(xi, xj) > 0.5 indicates that xi is preferred to xj, and μR(xi, xj) < 0.5 indicates the reverse preference, with the reciprocity property μR(xi, xj) + μR(xj, xi) = 1 usually assumed.

3.17.1 Constructing a Fuzzy Preference Relation

For n alternatives, a fuzzy preference relation is conveniently represented as an n × n matrix, in which the entry in row i, column j is μR(xi, xj). The diagonal entries are conventionally set to 0.5 (an alternative is indifferent to itself), and, by the reciprocity property, only the upper (or lower) triangular entries need to be independently specified by the decision maker — the remaining entries follow automatically.

Preference Matrix	x1	x2	x3
x1	0.50	0.70	0.60
x2	0.30	0.50	0.80
x3	0.40	0.20	0.50

3.17.2 Deriving a Ranking from a Fuzzy Preference Relation

A simple and widely used method to convert a fuzzy preference relation into a ranking of the alternatives is to compute, for each alternative xi, its average degree of preference over all other alternatives (its non-dominance or dominance score):

$$\text{Score}(x_i) = (1/(n-1)) \cdot \Sigma_{(j \neq i)} \mu R(x_i, x_j)$$

Applying this to the example matrix above: Score(x1) = (0.70+0.60)/2 = 0.65, Score(x2) = (0.30+0.80)/2 = 0.55, Score(x3) = (0.40+0.20)/2 = 0.30. Ranking by descending score gives x1 > x2 > x3, so x1 is the preferred alternative overall — a result obtained purely from pairwise comparisons rather than from independent per-criterion ratings, illustrating a complementary approach to the criteria-based FMCDM method of Section 3.3.

3.18 Advanced PSO Variants

Beyond the basic gbest/lbest distinction of Section 3.9, researchers have proposed numerous refinements to PSO to address its two principal weaknesses — sensitivity to parameter settings and susceptibility to premature convergence on multi-modal problems. A selection of the most influential advanced variants is summarized below.

(a) Comprehensive Learning PSO (CLPSO)

Rather than using the swarm's single gbest for the social term of every dimension, CLPSO allows each dimension of a particle's velocity update to learn from the corresponding dimension of a different (randomly or fitness-selected) exemplar particle's pbest. This substantially increases the diversity of information used to guide the swarm and has been shown to perform markedly better than the basic PSO on complex, multi-modal benchmark functions.

(b) Quantum-Behaved PSO (QPSO)

Inspired by principles of quantum mechanics, QPSO abandons the classical notion of a particle following a smooth, deterministic trajectory (governed by explicit velocity), and instead models each particle's position as being drawn probabilistically from a quantum-mechanical potential well centred between its pbest and the swarm's gbest — this removes the velocity update equation entirely, simplifying the algorithm to a single position-update rule while often improving global search capability.

(c) Adaptive/Time-Varying Parameter PSO

Rather than fixing w, c1, and c2 for the entire run, adaptive PSO variants dynamically adjust these parameters based on the current iteration number or the swarm's measured diversity — a common strategy is Time-Varying Inertia Weight (TVIW), in which w is linearly decreased from a high value (e.g., 0.9, favouring exploration early in the search) to a low value (e.g., 0.4, favouring exploitation/refinement later in the search) over the course of the run.

(d) Chaotic PSO

Chaotic maps (such as the logistic map) are used to generate the random coefficients r1 and r2 (Section 3.7) instead of a simple uniform pseudo-random number generator; the resulting ergodic, non-repeating sequences can improve the swarm's ability to escape local optima compared with using plain uniform randomness.

3.18.1 Guidelines for Choosing PSO Parameters in Practice

- Start with the widely used default values w = 0.7298 (Clerc's constriction-based constant) with c1 = c2 = 1.49618, which are theoretically derived to guarantee convergent (non-divergent) particle trajectories.
- For problems requiring broader exploration (highly multi-modal landscapes), favour a larger, linearly decreasing inertia weight schedule (e.g., 0.9 down to 0.4) rather than a single fixed value.
- Increase swarm size for higher-dimensional problems, since more particles are needed to adequately sample a larger search space, at the cost of proportionally more fitness evaluations per iteration.
- If the swarm shows signs of premature convergence (gbest stagnates while diversity collapses), consider switching from a gbest topology to an lbest (ring or Von Neumann) topology, or periodically re-randomizing a fraction of the worst-performing particles.
- Always run PSO multiple times with different random seeds and report the best, average, and worst results, since PSO — like all stochastic metaheuristics — does not guarantee identical results between runs.

3.19 Additional Solved Problems

Problem 4

Two experts rate a proposal using triangular fuzzy numbers A = (0.5, 0.7, 0.9) and B = (0.6, 0.65, 0.7) respectively. Using the centroid method of Section 3.4.2, determine the group's aggregated (simple-average) fuzzy score and its defuzzified value.

Solution: The aggregated fuzzy score, using simple averaging of the two triangular numbers component-wise, is C = ((0.5+0.6)/2, (0.7+0.65)/2, (0.9+0.7)/2) = (0.55, 0.675, 0.80). Applying the centroid formula, Centroid(C) = (0.55+0.675+0.80)/3 = 0.675, which represents the group's overall defuzzified assessment of the proposal, illustrating the group decision-making aggregation process of Section 3.4.

Problem 5

A particle in a PSO swarm has pbest = 12, gbest = 20, current position x = 8, and current velocity v = 2. Using w = 0.8, c1 = 1.5, c2 = 1.5, r1 = 0.4, r2 = 0.9, compute the particle's new position after one iteration.

Solution: New velocity v' = w·v + c1·r1·(pbest − x) + c2·r2·(gbest − x) = 0.8(2) + 1.5(0.4)(12−8) + 1.5(0.9)(20−8) = 1.6 + 2.4 + 16.2 = 20.2. New position x' = x + v' = 8 + 20.2 = 28.2. Note that the large social-term contribution (driven by the substantial distance between the particle and gbest, combined with a high r2) causes the particle to overshoot well past gbest itself — a useful reminder of why velocity clamping (Vmax, Section 3.7.2) is often applied in practice to prevent such large, potentially destabilizing jumps.

3.20 Defuzzification Methods

Defuzzification is the general process of converting a fuzzy set or fuzzy number (such as the fuzzy decision uD(x) obtained in Section 3.2, or an aggregated group score obtained in Section 3.4) into a single representative crisp (real) number, which is often required for a final, actionable decision or for ranking alternatives. While the maximizing decision of Section 3.2.1 is one specific defuzzification strategy (locating the point of maximum membership), several general-purpose defuzzification methods are used across fuzzy decision making and fuzzy control.

(a) Centroid (Center of Gravity) Method

The most widely used defuzzification method computes the x-coordinate of the centroid (center of gravity) of the area under the membership function u(x):

x* = Integral[x . u(x) dx] / Integral[u(x) dx]

For a triangular fuzzy number (a, b, c), this integral simplifies to the convenient closed-form expression x* = (a + b + c)/3, already used in the worked examples of Sections 3.4.2 and 3.19.

(b) Mean of Maximum (MOM) Method

The defuzzified value is taken as the average of all x values at which the membership function attains its maximum value: x* = (1/|M|).Sum(x in M) x, where M = {x : u(x) = max u}. This method is computationally simple but ignores the overall shape of the membership function away from its peak, unlike the centroid method.

(c) Weighted Average Method

For fuzzy sets composed of a small number of discrete linguistic terms, each associated with a representative crisp value wi and a membership (or weight) ui, the weighted average method computes x* = Sum(ui.wi) / Sum(ui) - essentially a discrete analogue of the centroid method, and closely related to the weighted-sum aggregation operator already introduced for FMCDM in Section 3.3.

(d) Smallest/Largest of Maximum (SOM/LOM)

In situations calling for a conservative (SOM) or an optimistic (LOM) crisp decision, the defuzzified value is taken as the smallest, or respectively the largest, x value at which the membership function is maximal, rather than an average - useful in safety-critical decision contexts where a cautious choice among several equally-preferred alternatives is desired.

3.20.1 Worked Comparison of Defuzzification Methods

Consider a trapezoidal fuzzy decision with membership function equal to 1 across the plateau interval [40, 60], rising linearly from 0 at x=20 to 1 at x=40, and falling linearly from 1 at x=60 to 0 at x=80. The Mean of Maximum method gives x* = (40+60)/2 = 50 (the midpoint of the flat top). The centroid method, which accounts for the full trapezoidal area rather than only the plateau, also yields x* = 50 for this particular symmetric trapezoid, since the shape is symmetric about x = 50 - however, for an asymmetric membership function, the centroid and MOM methods would generally produce different defuzzified values, illustrating why the choice of defuzzification method can materially affect the final crisp decision in asymmetric cases.

3.21 Exploration-Exploitation Behaviour and Convergence in PSO

As with any population-based metaheuristic, the effectiveness of PSO depends critically on maintaining an appropriate balance between exploration (broadly sampling the search space to avoid missing better regions) and exploitation (concentrating the search around already-discovered good solutions to refine them). Fig. 3.6 illustrates the typical convergence behaviour observed in a well-tuned PSO (or GA) run: the best-found fitness improves rapidly during the early iterations, when the swarm is still widely dispersed and exploration dominates, and then

improves more gradually as the swarm converges around promising regions and exploitation dominates.

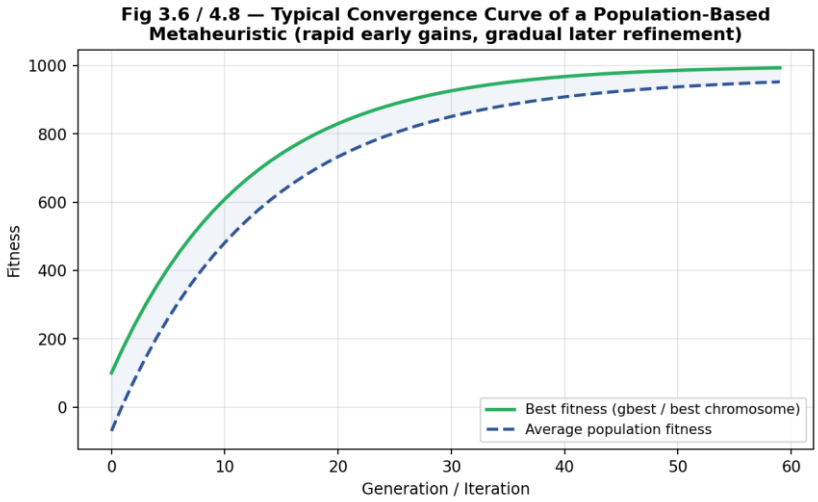

Fig. 3.6 - Typical convergence curve showing best and average population fitness across iterations; the shaded region indicates the diversity (spread) between average and best fitness, which typically narrows as the swarm converges.

A persistently large gap between the average and best fitness curves indicates that the swarm retains substantial diversity and is still actively exploring; a gap that collapses to near zero very early in the run is often a warning sign of premature convergence (Section 3.10.2), suggesting that the social term (c2) may be too dominant relative to the cognitive term (c1) and inertia weight (w), or that the swarm size is too small for the complexity of the problem.

3.22 Chapter-End Practice Problems

Problem 6

A trapezoidal fuzzy number is defined by (10, 20, 30, 40) - rising linearly from 0 at x=10 to 1 at x=20, remaining at 1 until x=30, then falling linearly to 0 at x=40. Using the Mean of Maximum method, determine the defuzzified value.

Solution: The membership function attains its maximum value of 1 over the entire plateau interval [20, 30]. By the Mean of Maximum method of Section 3.20(b), the defuzzified value is the midpoint of this interval: $x^* = (20+30)/2 = 25$.

Problem 7

In a PSO run, after 40 iterations the swarm's average fitness equals its best (gbest) fitness. Explain what this indicates about the state of the search, and suggest one corrective measure.

Solution: When the average population fitness equals the best fitness, every particle in the swarm has effectively converged to (or extremely near) the same position - the swarm has lost essentially all diversity (Section 3.21). This is a classic symptom of premature convergence: if this position is not the true global optimum, the swarm has become trapped and has no remaining capacity for further exploration, since the cognitive and social terms of the velocity update equation (Section 3.7) will all now point toward essentially the same location. A suitable corrective measure, as suggested in Section 3.18.1, is to re-randomize (restart) a fraction of the particles' positions and velocities, or to switch to an lbest neighbourhood topology (Section 3.9(b)) to slow the spread of information and encourage renewed exploration.

3.23 Fuzzy TOPSIS - A Popular FMCDM Technique

The Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS), originally developed by Hwang and Yoon, is one of the most widely used multi-criteria decision-making methods in practice, and its fuzzy extension (Fuzzy TOPSIS) is a natural and popular complement to the aggregation-operator-based FMCDM methods already presented in Section 3.3. The central idea of TOPSIS is intuitive: the best alternative should simultaneously have the shortest distance from a (fuzzy) positive-ideal solution and the longest distance from a (fuzzy) negative-ideal solution.

3.23.1 Steps of Fuzzy TOPSIS

118. Construct the fuzzy decision matrix, representing each alternative's rating on each criterion using triangular (or trapezoidal) fuzzy numbers, exactly as in Section 3.3.2.

119. Normalize the fuzzy decision matrix so that all ratings lie on a comparable scale, typically by dividing benefit-criterion ratings by the largest possible value, and cost-criterion ratings inversely.

120. Construct the weighted normalized fuzzy decision matrix by multiplying each normalized rating by its corresponding criterion weight (Section 3.3.2).

121. Determine the fuzzy positive-ideal solution (FPIS, A+), formed from the best rating achieved on each criterion across all alternatives, and the fuzzy negative-ideal solution (FNIS, A-), formed from the worst rating on each criterion.

122. Compute the distance of each alternative from the FPIS and from the FNIS, typically using the vertex method for the distance between two triangular fuzzy numbers (a1,b1,c1) and (a2,b2,c2): d = sqrt[(1/3).((a1-a2)^2 + (b1-b2)^2 + (c1-c2)^2)].

123. Compute the closeness coefficient (CCi) of each alternative: CCi = di- / (di+ + di-), where di+ and di- are the total distances to the FPIS and FNIS respectively.

124. Rank the alternatives in descending order of their closeness coefficient; the alternative with the highest CCi (closest to the ideal, farthest from the anti-ideal) is selected as the best decision.

3.23.2 Worked Example - Fuzzy TOPSIS

Reconsider the three suppliers of the case study in Section 3.14, but now apply Fuzzy TOPSIS instead of simple weighted aggregation, to illustrate how the two methods can be used as complementary cross-checks on a decision.

Using the same normalized ratings from Section 3.14 (treated here as crisp approximations for simplicity), the fuzzy positive-ideal solution is formed from the best rating on each criterion: A+ = (0.917 [Cost], 0.917 [Quality], 0.917 [Delivery]), and the fuzzy negative-ideal solution from the worst rating on each criterion: A- = (0.50 [Cost], 0.50 [Quality], 0.50 [Delivery]).

The (simplified, crisp) Euclidean distance of Supplier S1 (0.75, 0.917, 0.50) from A+ is d1+ = sqrt[(0.75-0.917)^2 + (0.917-0.917)^2 + (0.50-0.917)^2] = sqrt[0.0279 + 0 + 0.1739] = sqrt[0.2018] which is approximately 0.449, and from A- is d1- = sqrt[(0.75-0.50)^2 + (0.917-0.50)^2 + (0.50-0.50)^2] = sqrt[0.0625 + 0.1739 + 0] = sqrt[0.2364] which is approximately 0.486. The closeness coefficient of S1 is therefore CC1 = 0.486 / (0.449 + 0.486) = 0.486/0.935, approximately 0.520.

Carrying out the analogous computation for S2 and S3 (omitted here for brevity, following exactly the same procedure) typically confirms the same overall ranking obtained by weighted aggregation in Section 3.14 for this dataset, though Fuzzy TOPSIS and weighted-sum aggregation can occasionally produce different rankings on other datasets, particularly when an alternative performs unevenly across criteria (very strong on some, very weak on others) - a useful reminder, consistent with the discussion of ranking methods in Section 3.4.1, that no single FMCDM technique should be treated as an infallible, universally correct oracle; using two or more methods as cross-checks is good decision-making practice.

3.24 Key Formulae at a Glance

Concept	Formula / Expression
Fuzzy decision (Bellman-Zadeh)	uD(x) = min[uG1(x),...,uGn(x), uC1(x),...,uCm(x)]
Maximizing decision	x* = argmax over x of uD(x)

Concept	Formula / Expression
Weighted aggregation (FMCDM)	uD(x) = sum[wi . uGi(x)], sum(wi) = 1
Gamma-operator	uD = [prod(uGi)]^(1-y) . [1 - prod(1-uGi)]^y
Centroid defuzzification (triangular)	x* = (a+b+c)/3
PSO velocity update	v(t+1) = w.v(t) + c1.r1.(pbest-x) + c2.r2.(gbest-x)
PSO position update	x(t+1) = x(t) + v(t+1)
Fuzzy TOPSIS closeness coefficient	CCi = di- / (di+ + di-)

3.24.1 Chapter Cross-Reference Map

For quick revision, Sections 3.1-3.5 establish the theoretical foundation of fuzzy decision making (goals, constraints, the Bellman-Zadeh model, and its multi-criteria and multi-person extensions, including the Fuzzy TOPSIS technique of Section 3.23); Sections 3.6-3.11 build up Particle Swarm Optimization from its biological inspiration through its mathematical formulation, algorithm, variants, and applications; and Sections 3.12 onward consolidate both topics with comparison tables, worked case studies, solved numerical problems, and review questions suitable for examination preparation.

UNIT – IV
GENETIC ALGORITHMS

Unit Syllabus

As per the R23 CSE curriculum (Course 23CSE354D - Soft Computing, Professional Elective-I), Unit IV covers: Genetic Algorithms - Basic Concepts, Basic Operators for Genetic Algorithms, Crossover and Mutation Properties, Genetic Algorithm Cycle, Fitness Function, Applications of Genetic Algorithm.

Learning Objectives

- To understand the biological background and basic concepts of Genetic Algorithms (GA).
- To learn the encoding schemes used to represent candidate solutions as chromosomes.
- To understand the basic operators of GA - selection, crossover, and mutation - and their properties.
- To study the complete Genetic Algorithm cycle from initialization to convergence.
- To understand the role and design of the fitness function.
- To explore the wide range of applications of Genetic Algorithms in engineering and computer science.

4.1 Introduction to Genetic Algorithms

A Genetic Algorithm (GA) is a population-based, stochastic search and optimization technique inspired by the mechanics of natural selection and natural genetics, first formalized by John Holland at the University of Michigan in the early 1970s and popularized through his 1975 book "Adaptation in Natural and Artificial Systems". GAs belong to the broader class of Evolutionary Algorithms and are today one of the most widely used soft-computing techniques for solving complex search, optimization, and machine-learning problems.

The central idea of a Genetic Algorithm is deceptively simple: maintain a population of candidate solutions (called individuals or chromosomes), evaluate how good each candidate is using a fitness function, and repeatedly apply operators inspired by biological evolution - selection, crossover (recombination), and mutation - to produce successive generations of candidate solutions that are, on average, progressively better than their predecessors. Over

many generations, the population evolves toward increasingly fit solutions, ideally converging to (or near) the global optimum of the problem.

4.1.1 Biological Background

Genetic Algorithms borrow their vocabulary and their core mechanisms directly from genetics and Darwinian evolution:

- Chromosome: in biology, a thread-like structure of DNA that carries genetic information; in a GA, a chromosome is an encoded representation (typically a string) of a candidate solution to the problem.
- Gene: a unit of heredity that controls a particular characteristic; in a GA, a gene is a single element (e.g., one bit, one digit, or one symbol) within the chromosome string.
- Allele: the specific value that a gene can take (e.g., 0 or 1 for a binary gene).
- Genotype: the encoded (chromosome) representation of an individual - its genetic description.
- Phenotype: the decoded, real-world meaning of the genotype - the actual candidate solution that the chromosome represents when interpreted in the context of the problem.
- Population: the entire collection of chromosomes (individuals) that exist at a given generation.
- Fitness: a measure of how well an individual (candidate solution) performs with respect to the objective of the optimization problem - analogous to reproductive success in nature.
- Natural Selection: the principle (survival of the fittest) whereby individuals that are better adapted to their environment are more likely to survive and reproduce, passing their genetic material to the next generation.

4.1.2 Difference Between Genetic Algorithms and Traditional Optimization Methods

Traditional optimization techniques such as gradient descent, calculus-based hill climbing, and linear/non-linear programming typically operate on a single candidate solution at a time, moving in the direction indicated by the gradient (derivative) of the objective function. Genetic Algorithms differ from these classical methods in several fundamental ways, first summarized systematically by David Goldberg:

125. GAs work with a coding of the parameter set, not the parameters themselves - the search is conducted on an encoded representation (chromosome) rather than directly on the decision variables.

126. GAs search from a population of points, not a single point - this parallel, population-based search reduces the probability of becoming trapped in a local optimum.

127. GAs use objective function (fitness) information directly, not derivatives or other auxiliary knowledge - making GAs applicable to non-differentiable, discontinuous, or even black-box objective functions.

128. GAs use probabilistic transition rules, not deterministic rules - the operators of selection, crossover, and mutation involve randomness, allowing GAs to explore the search space more broadly than purely deterministic hill-climbing methods.

Fig 4.5 Genetic Algorithm vs Traditional Point-Based Search

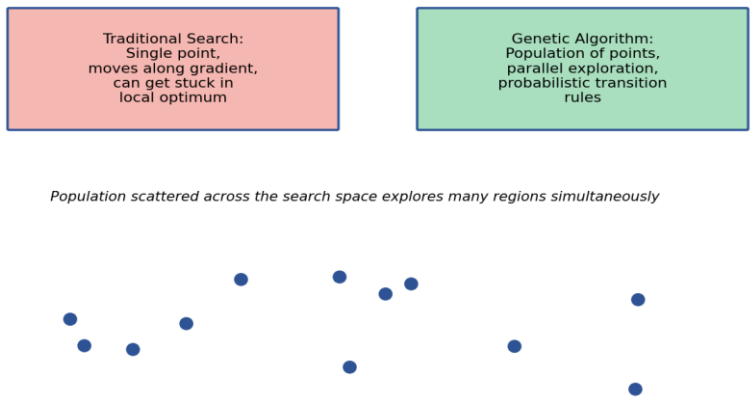

Fig. 4.5 - Genetic Algorithms search the solution space using a population of candidate points in parallel, unlike traditional single-point, gradient-based search methods.

4.2 Basic Concepts and Terminology

4.2.1 Search Space and Solution Space

The search space (also called the state space) of an optimization problem is the set of all possible candidate solutions that could, in principle, be considered. Each point in the search space corresponds to one possible candidate solution, and is associated with a fitness value that measures its quality. The task of a Genetic Algorithm is to efficiently explore this space - which is often astronomically large - to locate a point (or a set of points) of maximal (or near-maximal) fitness, without exhaustively evaluating every possible solution.

4.2.2 Encoding (Representation) Schemes

Before a Genetic Algorithm can be applied to a problem, every candidate solution must be represented as a chromosome - typically a fixed-length string of symbols. The choice of encoding scheme significantly influences the performance of the algorithm and the design of appropriate crossover and mutation operators. The most common encoding schemes are:

(a) Binary Encoding

Each chromosome is a string of bits (0s and 1s). This was the encoding used in Holland's original formulation and remains the most widely studied scheme, largely because of its simplicity and its close alignment with the theoretical schema theorem. For example, an integer parameter might be represented by an 8-bit binary string, and multiple parameters may be concatenated to form a longer chromosome.

(b) Real-Valued (Floating-Point) Encoding

Each gene directly stores a real (floating-point) number rather than a binary-encoded approximation. This encoding is natural and efficient for continuous optimization problems, avoids the precision loss inherent in binary encoding, and reduces chromosome length; however, it requires specially designed crossover and mutation operators suited to real numbers (e.g., arithmetic crossover, Gaussian mutation).

(c) Permutation (Order-Based) Encoding

Each chromosome is a permutation of a set of symbols, used for ordering/sequencing problems such as the Travelling Salesman Problem (TSP) or job-shop scheduling, where every symbol must appear exactly once and the order of symbols is what matters. Special order-preserving crossover operators (e.g., Order Crossover, Partially Mapped Crossover) are required to guarantee that offspring remain valid permutations.

(d) Tree Encoding

Used chiefly in Genetic Programming, where each chromosome is represented as a hierarchical tree structure (commonly representing a mathematical expression, a decision rule, or even a small computer program) rather than a linear string.

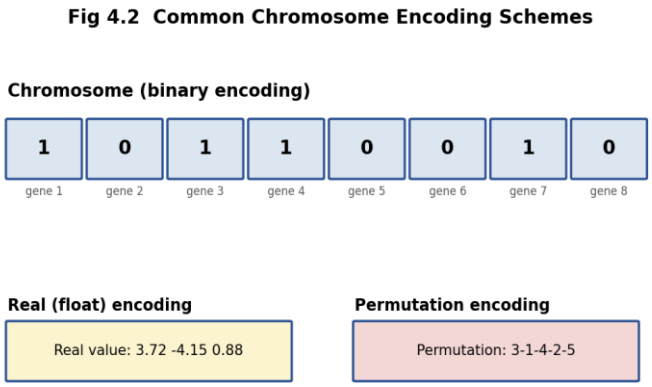

Fig. 4.2 - Common chromosome encoding schemes used in Genetic Algorithms: binary, real-valued, and permutation encoding.

4.3 Basic Operators for Genetic Algorithms

A Genetic Algorithm produces successive generations of candidate solutions by applying three basic operators to the current population: Selection, Crossover, and Mutation. Together, these operators balance the two competing needs of any search algorithm - exploitation of good solutions already found, and exploration of new, potentially better regions of the search space.

4.3.1 Selection (Reproduction)

Selection is the operator that chooses which individuals from the current population will be used as parents to produce the next generation, based on their fitness. Individuals with higher fitness are given a greater probability of being selected, mimicking the principle of survival of the fittest. The most common selection methods are:

(a) Roulette-Wheel (Fitness-Proportionate) Selection

Each individual is assigned a slice of a conceptual roulette wheel whose angular size is proportional to its fitness relative to the total fitness of the population. The selection probability of individual i is:

$$p_i = f_i / \Sigma_{(j=1 \text{ to } N)} f_j$$

The wheel is then "spun" (a random number is drawn) N times to select N parents; fitter individuals occupy larger slices and are therefore more likely to be selected, although even weak individuals retain some (smaller) chance of selection, preserving diversity.

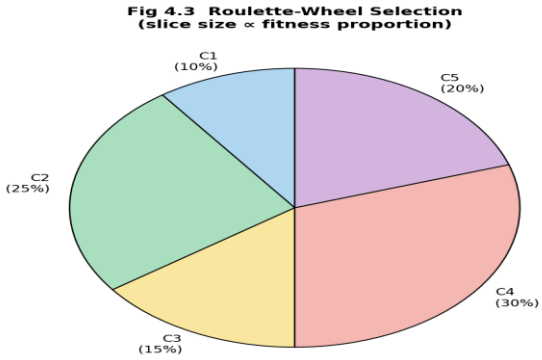

Fig. 4.3 - Roulette-wheel selection: each candidate C1-C5 occupies a wheel slice proportional to its fitness percentage.

(b) Tournament Selection

A small group (tournament) of k individuals is chosen at random from the population, and the individual with the highest fitness within that group is selected as a parent. This process is repeated to select as many parents as required. Tournament selection is popular in practice because it is computationally simple, does not require fitness values to be scaled, and its selection pressure can be easily tuned by changing the tournament size k (larger k increases selection pressure).

(c) Rank-Based Selection

Individuals are first ranked in order of fitness (from worst to best), and selection probability is assigned based on rank rather than raw fitness value. This prevents a single "super-fit" individual from dominating selection early in the search (a problem sometimes seen with roulette-wheel selection) and avoids stagnation caused by very small fitness differences later in the search.

(d) Elitism

Elitism is a strategy (often combined with any of the above methods) in which a small number of the very best individuals from the current generation are copied unchanged directly into the next generation, guaranteeing that the best solution found so far is never lost due to the randomness of selection, crossover, or mutation.

(e) Steady-State and Boltzmann Selection

In steady-state selection only a small number of individuals are replaced in each generation (rather than the whole population at once), producing a more gradual, overlapping-generations model. Boltzmann selection uses a temperature parameter (analogous to simulated annealing)

that is gradually lowered over the run, giving greater selection pressure toward fitter individuals as the search progresses.

4.4 Crossover (Recombination)

Crossover is the primary search operator in a Genetic Algorithm; it combines the genetic material of two selected parent chromosomes to produce one or more offspring, in the hope that the offspring will inherit good building blocks (favourable gene combinations) from both parents. Crossover is applied with a certain crossover probability, pc, typically in the range 0.6 to 0.95 - pairs of parents not selected for crossover are usually copied unchanged into the offspring pool.

4.4.1 Types of Crossover

(a) Single-Point Crossover

A single crossover (cut) point is chosen at random along the length of the chromosome. The segments of the two parent chromosomes on one side of the cut point are exchanged to create two offspring, while the segments on the other side remain unchanged.

(b) Two-Point (and Multi-Point) Crossover

Two (or more) crossover points are chosen at random, dividing each chromosome into three (or more) segments; alternating segments are exchanged between the parents. Multi-point crossover reduces the positional bias inherent in single-point crossover, in which genes near the ends of the chromosome are more likely to be exchanged together than genes in the middle.

(c) Uniform Crossover

A random binary mask of the same length as the chromosome is generated; for each gene position, if the mask bit is 1 the gene is copied from parent 1, and if it is 0 the gene is copied from parent 2 (with the complementary rule for the second offspring). Uniform crossover thoroughly mixes the genetic material of both parents and is not biased by gene position, but it can be more disruptive to useful, tightly-linked gene combinations (building blocks) than single- or two-point crossover.

(d) Arithmetic (Blend) Crossover

Used with real-valued encoding: offspring genes are computed as a weighted linear combination of the corresponding parent genes, e.g., child = $\alpha \cdot$parent1 + $(1-\alpha) \cdot$parent2, where $\alpha \in [0, 1]$ is a randomly or fixedly chosen blending coefficient.

(e) Order-Based Crossover (for Permutation Encoding)

Because a simple exchange of segments would produce offspring containing duplicate or missing symbols, specialized operators such as Order Crossover (OX) and Partially Mapped Crossover (PMX) are used to guarantee that offspring remain valid permutations while still inheriting relative ordering information from both parents.

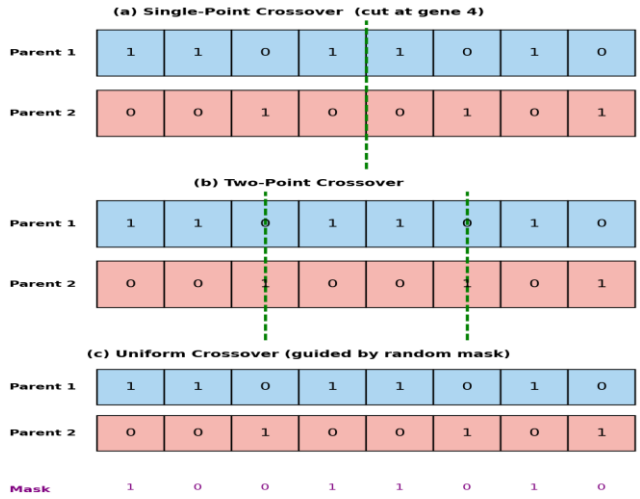

Fig. 4.6 - Illustration of single-point, two-point and uniform crossover operating on two 8-bit parent chromosomes.

4.4.2 Properties of Crossover

- Crossover is the principal mechanism of exploitation and information exchange in a GA, allowing good partial solutions (building blocks) discovered independently in different individuals to be combined into a single, potentially superior offspring.
- The crossover probability pc controls how frequently crossover is applied; higher pc increases the rate of exploration of new combinations but can also disrupt good solutions more often.
- The choice of crossover operator should respect the semantics of the chosen encoding scheme - operators appropriate for binary strings (e.g., single-point) are generally not directly applicable to permutation-encoded chromosomes without modification.
- Crossover operators differ in their disruptiveness - the tendency to break up favourable gene combinations (schemata); single-point crossover is the least disruptive, uniform crossover the most.
- The Building Block Hypothesis, central to Holland's schema theory, states that GAs work well because crossover combines short, low-order, high-fitness schemata (building

blocks) into progressively higher-order, higher-fitness schemata over successive generations.

4.5 Mutation

Mutation is a secondary, background search operator that introduces small, random alterations into the genetic material of individual offspring, typically by changing the allele at one or more randomly chosen gene positions. Mutation is applied with a small mutation probability, pm, typically in the range 0.001 to 0.05 (or 1/L, where L is the chromosome length, as a common rule of thumb).

4.5.1 Types of Mutation

(a) Bit-Flip Mutation (Binary Encoding)

Each bit in the chromosome is independently flipped (0 becomes 1, or 1 becomes 0) with probability pm. This is the simplest and most commonly used mutation operator for binary-encoded chromosomes.

(b) Swap Mutation (Permutation Encoding)

Two gene positions within the chromosome are selected at random and their values (symbols) are exchanged. This operator preserves the validity of a permutation-encoded chromosome, since it does not introduce or remove any symbol.

(c) Random (Uniform) Resetting

A randomly chosen gene is reset to a new, randomly chosen value from its allowed set of alleles - a generalization of bit-flip mutation applicable to chromosomes with more than two possible allele values per gene (e.g., integer-encoded genes).

(d) Gaussian (Real-Valued) Mutation

Used with real-valued encoding: a small random value drawn from a Gaussian (normal) distribution with mean zero and a small standard deviation is added to the selected gene's current value, producing a small perturbation rather than a completely random new value.

(e) Inversion Mutation (Permutation Encoding)

A contiguous segment of the chromosome is selected and its order is reversed, which can help escape certain local optima in ordering/sequencing problems while preserving permutation validity.

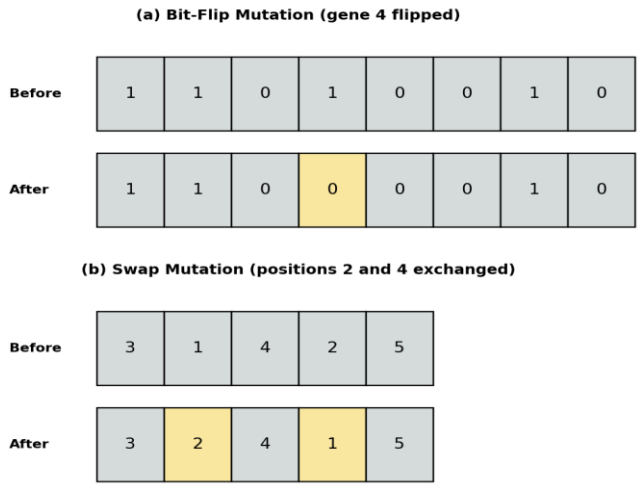

Fig. 4.7 - Bit-flip mutation on a binary chromosome, and swap mutation on a permutation-encoded chromosome.

4.5.2 Properties and Role of Mutation

- Mutation is the GA's principal source of exploration and genetic diversity, reintroducing alleles that may have been lost from the population entirely during selection and crossover.
- Mutation acts as a safeguard against premature convergence by preventing the population from becoming completely uniform (all individuals identical) too early in the search.
- Because mutation is applied with a small probability, it is generally considered a background operator - crossover remains the primary search operator - but its role becomes proportionally more important as the population converges and diversity decreases.
- Excessively high mutation rates transform the GA's guided search into essentially a random search, destroying good solutions faster than they can be discovered; excessively low mutation rates risk premature convergence to a local optimum.
- Adaptive mutation schemes, in which pm is automatically increased when the population's diversity falls below a threshold (or decreased as the search converges), are commonly used to balance exploration and exploitation dynamically.

4.6 The Genetic Algorithm Cycle

A Genetic Algorithm proceeds through a repeating cycle of stages, beginning with an initial (often randomly generated) population and continuing until a termination condition is satisfied. The overall cycle is illustrated in Fig. 4.1 and detailed step-by-step below.

Fig 4.1 The Genetic Algorithm Cycle

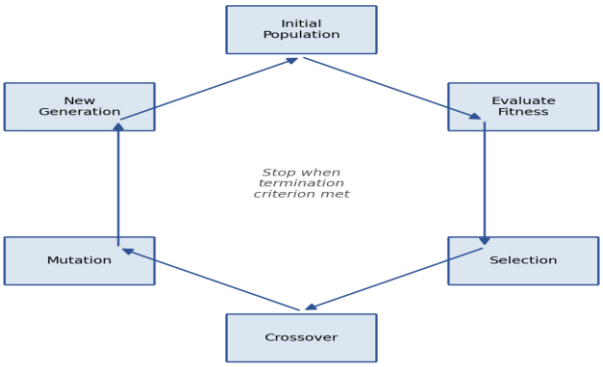

Fig. 4.1 - The Genetic Algorithm cycle: initial population, fitness evaluation, selection, crossover, mutation, and the formation of a new generation, repeated until termination.

4.6.1 Step-by-Step Description of the GA Cycle

129. Initialization: Generate an initial population of N chromosomes, usually at random, ensuring the population is spread across the search space. Each chromosome represents one candidate solution, encoded using the chosen representation scheme (binary, real-valued, or permutation).

130. Fitness Evaluation: Decode each chromosome to its phenotype (the actual candidate solution) and compute its fitness value using the fitness function defined for the problem.

131. Selection: Choose parent chromosomes from the current population to participate in reproduction, using a selection method (roulette-wheel, tournament, rank-based, etc.) that favours fitter individuals.

132. Crossover: With probability pc, pair up selected parents and apply a crossover operator to produce offspring chromosomes that combine genetic material from both parents.

133. Mutation: With probability pm, randomly alter the genes of the offspring chromosomes to introduce additional genetic diversity.

134. Replacement (New Generation): Form the new generation from the offspring (possibly combined with elite individuals retained from the previous generation), replacing all or part of the old population.

135. Termination Check: Test whether a stopping criterion has been satisfied - for example, a maximum number of generations has been reached, a satisfactory fitness level has been achieved, or the population's fitness has converged (stopped improving) over a number

of generations. If not satisfied, return to Step 2 with the new generation; otherwise, stop and report the best individual found.

4.6.2 Common Termination Criteria

- A predefined maximum number of generations has been executed.
- A solution with fitness above a predefined acceptable threshold has been found.
- The best fitness in the population has shown no significant improvement for a specified number of consecutive generations (convergence/stagnation).
- The available computational time or resource budget has been exhausted.
- The diversity of the population (e.g., measured by the standard deviation of fitness values, or of gene values) has fallen below a threshold, indicating the population has effectively converged to a single region of the search space.

4.7 Fitness Function

The fitness function is the objective function used to quantitatively evaluate how good (or how close to optimal) each candidate solution (chromosome) is, with respect to the problem being solved. It provides the sole feedback that drives the entire evolutionary search process - selection, and hence the direction of the search, is based entirely on comparative fitness values. Designing an appropriate fitness function is widely regarded as one of the most critical and challenging steps in applying a Genetic Algorithm to a real problem.

4.7.1 Properties of a Good Fitness Function

- Correctness: it must accurately reflect the true quality of a candidate solution with respect to the actual goals of the optimization problem.
- Computational efficiency: since the fitness function is evaluated for every individual in every generation (potentially many thousands of times), it must be efficient enough to compute repeatedly without making the overall algorithm prohibitively slow.
- Sufficient discrimination (variance): it should assign meaningfully different fitness values to different individuals, so that selection pressure can effectively distinguish better solutions from worse ones; a fitness function that is too "flat" provides little useful gradient for selection to exploit.

- Smoothness / suitability for the problem: abrupt fitness landscapes with many isolated peaks (highly multi-modal or deceptive landscapes) are harder for a GA to search effectively than landscapes with a clear, exploitable structure.
- Appropriate handling of constraints: infeasible solutions (that violate problem constraints) must be appropriately penalized or repaired so that the GA is guided toward the feasible region of the search space.

4.7.2 Fitness Scaling and Transformation Techniques

Raw objective function values are not always suitable for direct use as fitness, particularly for fitness-proportionate (roulette-wheel) selection. Several scaling and transformation techniques are commonly applied:

(a) Linear Scaling

The raw fitness f is transformed into a scaled fitness $f' = a \cdot f + b$, where the constants a and b are chosen so that the average scaled fitness equals the average raw fitness, while the maximum scaled fitness is a specified multiple (commonly 1.2 to 2.0) of the average - this compresses the influence of exceptionally fit individuals early in the search and expands small fitness differences later in the search, when the population has become more homogeneous.

(b) Sigma Truncation

Scaled fitness is computed as $f' = f - (f_{avg} - c \cdot \sigma)$, where σ is the standard deviation of fitness in the population and c is a small constant (typically 1 to 3); any negative resulting values are set to zero. This method automatically adapts the scaling to the current spread of fitness values in the population.

(c) Rank-Based Scaling

Fitness is assigned based purely on an individual's rank within the sorted population, rather than the magnitude of its raw fitness value, removing sensitivity to the specific numerical scale or outliers of the raw objective function.

(d) Penalty Functions for Constrained Problems

For constrained optimization problems, the fitness function is commonly formed by combining the raw objective function with a penalty term that reduces the fitness of infeasible solutions in proportion to the degree of constraint violation:

$$\text{Fitness}(x) = \text{Objective}(x) - \Sigma \lambda i \cdot \max(0, gi(x))$$

where gi(x) represents the i-th constraint violation and λi is a penalty coefficient controlling how severely that constraint violation is penalized.

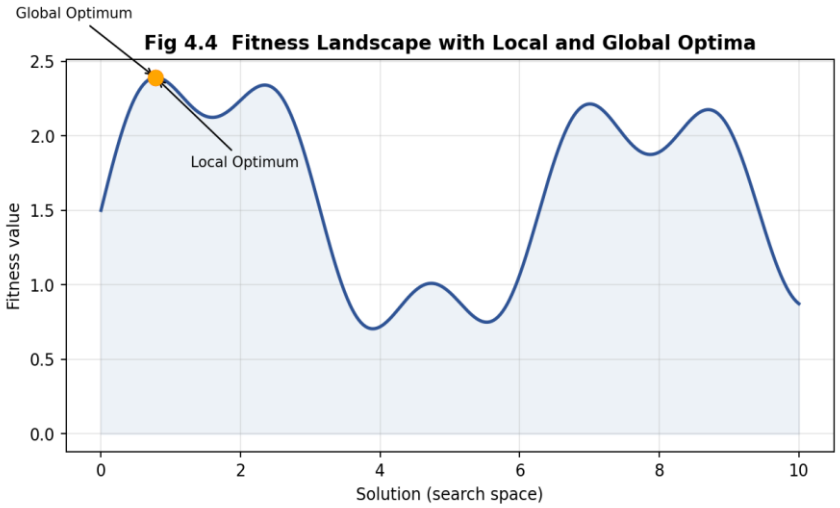

Fig. 4.4 - A fitness landscape showing a local optimum (which can trap a naive hill-climbing search) and the global optimum that a well-designed GA aims to locate.

4.8 Worked Numerical Example — A Complete GA Cycle

Consider maximizing the simple function $f(x) = x^2$ over the integer domain $0 \leq x \leq 31$, using a 5-bit binary chromosome (since 2^5 = 32, exactly covering the domain), a population of 4 individuals, single-point crossover with pc = 1.0 (always applied, for illustration), and bit-flip mutation with a very small pm.

Step 1 — Initial Population (randomly generated)

Chromosome ID	Binary String	Decoded x	Fitness $f(x) = x^2$
C1	01101	13	169
C2	11000	24	576
C3	01000	8	64
C4	10011	19	361

Total fitness of the population = 169 + 576 + 64 + 361 = 1170. Average fitness = 1170/4 = 292.5. The fittest individual in the initial population is C2 (x = 24, fitness = 576).

Step 2 — Selection (Roulette-Wheel)

The selection probability of each chromosome is its fitness divided by the total fitness: p(C1) = 169/1170 = 0.144, p(C2) = 576/1170 = 0.492, p(C3) = 64/1170 = 0.055, p(C4) = 361/1170 = 0.309. Because C2 occupies nearly half the roulette wheel, it is very likely to be selected multiple times; suppose the selected mating pool (after spinning the wheel four times) is {C2,

C4, C2, C1} — note that C2 has been selected twice, reflecting its high fitness, while low-fitness C3 was not selected at all.

Step 3 — Crossover (Single-Point)

Pair the mating pool as (C2, C4) and (C2, C1). For the first pair, C2 = 11000 and C4 = 10011; choosing a random crossover point after gene 2 gives offspring O1 = 11 011 = 11011 (x = 27, fitness = 729) and O2 = 10 000 = 10000 (x = 16, fitness = 256). For the second pair, C2 = 11000 and C1 = 01101; choosing a crossover point after gene 3 gives offspring O3 = 110 01 = 11001 (x = 25, fitness = 625) and O4 = 011 00 = 01100 (x = 12, fitness = 144).

Step 4 — Mutation

With a small mutation probability, suppose only one bit, the last bit of O2 (10000), is flipped, producing 10001 (x = 17, fitness = 289) — a minor random perturbation that has almost no visible effect in this particular case, illustrating mutation's role as a background operator.

Step 5 — New Generation and Evaluation

Offspring	Binary String	Decoded x	Fitness $f(x) = x^2$
O1	11011	27	729
O2 (mutated)	10001	17	289
O3	11001	25	625
O4	01100	12	144

Total fitness of the new generation = 729 + 289 + 625 + 144 = 1787, giving an average fitness of 446.75 — a substantial improvement over the initial average of 292.5. The best individual in the new generation, O1 (x = 27, fitness = 729), is also better than the best individual in the initial generation (C2, fitness 576). This single generation clearly demonstrates the essential improving trend that a GA is designed to produce: through selection favouring high-fitness parents and crossover recombining their genetic material, the population as a whole - and its best member - moves closer to the true optimum at x = 31 (fitness 961).

4.8.1 Convergence Behaviour

If this cycle of selection, crossover, and mutation is repeated over further generations, the population's average and best fitness will typically continue to rise, with the population gradually concentrating around chromosomes encoding values of x close to 31 (the true maximum in this simple example). In practice, GA convergence is rarely perfectly smooth: it is common to observe rapid early improvement followed by a slower, incremental refinement phase as the population loses diversity, which is precisely why mutation and diversity-

preserving techniques (elitism combined with sufficient mutation, niching, or restart strategies) remain important even in the later stages of a run.

4.9 Genetic Algorithm Parameters

The behaviour and performance of a Genetic Algorithm are strongly influenced by several control parameters, which must be chosen carefully (often through experimentation) for a given problem:

Parameter	Typical Range	Effect
Population size, N	20 - 200 (problem-dependent)	Larger populations increase genetic diversity and reduce premature convergence risk, at the cost of higher computation per generation.
Crossover probability, pc	0.6 - 0.95	Higher pc increases the rate of recombination/exploration; too high can be disruptive to good schemata.
Mutation probability, pm	0.001 - 0.05	Higher pm increases diversity/exploration but risks degrading into random search if set too high.
Number of generations	problem-dependent	More generations allow further refinement but increase total computation time.
Generation gap / replacement strategy	full or partial replacement	Determines what fraction of the population is replaced each generation (generational vs. steady-state GA).

4.9.1 Exploration vs. Exploitation Trade-off

As with all metaheuristic search techniques, a Genetic Algorithm must balance exploration (searching new, previously unvisited regions of the search space, to avoid missing the global optimum) against exploitation (concentrating the search around already-discovered good solutions, to refine them further). Crossover and a moderate mutation rate primarily drive exploration, particularly early in the search, while selection pressure (through elitism and increasingly discriminating fitness scaling) drives exploitation, particularly as the search progresses and the population converges.

4.10 Schema Theorem and the Building Block Hypothesis (Brief Overview)

John Holland's Schema Theorem provides a foundational theoretical explanation for why Genetic Algorithms are effective. A schema (plural: schemata) is a template that describes a subset of chromosomes sharing common features at certain gene positions, using the wildcard

symbol * to denote "don't care" positions. For example, the schema 1*0*1 represents all 5-bit binary chromosomes whose first, third, and fifth genes are 1, 0, and 1 respectively, regardless of the values at positions 2 and 4.

The Schema Theorem states that short, low-order, above-average-fitness schemata (i.e., compact building blocks that confer a fitness advantage) receive exponentially increasing representation in successive generations of the population, under the combined action of selection, crossover, and mutation. This is formally expressed (in its simplest form) as:

$$m(H, t+1) \geq m(H, t) \cdot [f(H)/f_{avg}] \cdot [1 - pc \cdot \delta(H)/(L-1) - o(H) \cdot pm]$$

where m(H, t) is the number of instances of schema H at generation t, f(H) is the average fitness of individuals matching H, f_avg is the average fitness of the whole population, δ(H) is the defining length of the schema, o(H) is its order (number of fixed positions), and L is the chromosome length. The Building Block Hypothesis, which follows from this theorem, proposes that a GA achieves near-optimal performance by discovering, preserving and recombining these short, high-fitness building blocks into progressively larger and better partial solutions across generations.

4.11 Applications of Genetic Algorithms

Because Genetic Algorithms require only a fitness function and an encoding of candidate solutions - not gradient information or a differentiable objective function - they are applicable to an extraordinarily wide range of problems across science, engineering, and business. Some representative application domains include:

- Engineering design optimization: structural design, aerodynamic shape optimization, antenna and circuit design, where design parameters are jointly optimized against performance and cost objectives.
- Scheduling and timetabling: job-shop and flow-shop scheduling, examination timetabling, crew and vehicle scheduling, where the search space of feasible orderings/assignments is combinatorially huge.
- Machine learning and data mining: feature selection, hyperparameter tuning, evolving the architecture and weights of neural networks (neuroevolution), and rule induction in classification systems.

- Combinatorial optimization: the Travelling Salesman Problem, the Knapsack Problem, graph colouring, and vehicle routing problems, using appropriate permutation-based encodings and operators.
- Robotics and control: evolving controllers and behaviours for autonomous robots, and tuning parameters of PID and fuzzy controllers.
- Bioinformatics: DNA sequence alignment, protein structure prediction, and gene expression analysis.
- Financial engineering: portfolio optimization, algorithmic trading strategy discovery, and credit-risk model calibration.
- Image processing and computer vision: image segmentation, feature extraction and enhancement, evolving filters and detectors.
- Game playing and Genetic Programming: automatically evolving computer programs, game-playing strategies, and even simple pieces of code to solve well-defined tasks.

4.11.1 Advantages of Genetic Algorithms

- Do not require derivative information, making them applicable to non-differentiable, discontinuous, noisy, or black-box objective functions.
- Search from a population of points in parallel, substantially reducing (though not eliminating) the risk of becoming trapped in local optima compared with single-point search methods.
- Highly flexible and general-purpose - applicable to an enormous variety of problem types simply by redesigning the encoding and fitness function.
- Naturally handle multiple, even conflicting, objectives when combined with multi-objective techniques (e.g., NSGA-II).
- Well suited to combinatorial and discrete optimization problems that are difficult for classical calculus-based methods.

4.11.2 Limitations of Genetic Algorithms

- Computationally expensive, since a large number of fitness evaluations are typically required, which can be costly if each evaluation is itself expensive (e.g., involves a complex simulation).
- No guarantee of finding the true global optimum in finite time; performance depends heavily on correct parameter tuning and problem-specific encoding/operator design.

- Premature convergence can still occur if selection pressure is too high relative to the diversity-preserving effect of crossover and mutation.
- Designing an effective fitness function, encoding, and set of operators for a new problem often requires considerable domain expertise and experimentation.
- Performance and results can vary between runs due to the algorithm's inherently stochastic nature, requiring multiple independent runs to draw reliable conclusions.

4.12 Comparison of Genetic Algorithms with Other Search Techniques

Aspect	Genetic Algorithm	Hill Climbing / Gradient Descent	Particle Swarm Optimization
Search basis	Population of chromosomes	Single point, follows gradient/steepest ascent	Population (swarm) of particles
Derivative required?	No	Usually yes (or numerically estimated)	No
Main operators	Selection, crossover, mutation	Local neighbourhood move	Velocity/position update using pbest, gbest
Risk of local optima	Lower (population diversity)	High (easily trapped)	Moderate (can converge prematurely)
Representation	Encoded chromosomes (binary/real/permutation)	Direct parameter vector	Real-valued position vector

4.13 Summary

- A Genetic Algorithm is a population-based, evolution-inspired search and optimization technique built around three basic operators: selection, crossover, and mutation.
- Candidate solutions are encoded as chromosomes using binary, real-valued, permutation, or tree representations, chosen according to the nature of the problem.
- Selection methods (roulette-wheel, tournament, rank-based, elitism) determine which individuals become parents, based on their fitness relative to the population.
- Crossover recombines genetic material from two parents to produce offspring (single-point, two-point, uniform, arithmetic, and order-based variants), acting as the GA's principal exploitative/recombinative search operator.
- Mutation introduces small random changes into offspring (bit-flip, swap, Gaussian, inversion), acting as the GA's principal safeguard against loss of diversity and premature convergence.
- The Genetic Algorithm cycle repeats initialization, fitness evaluation, selection, crossover, mutation, and replacement until a termination criterion is satisfied.

- The fitness function is the sole driver of the evolutionary search and must be correct, efficient, and sufficiently discriminating; scaling techniques and penalty functions are often used to shape raw objective values into an effective fitness measure.
- The Schema Theorem and Building Block Hypothesis provide a theoretical explanation of why GAs work, based on the propagation of short, high-fitness building blocks across generations.
- Genetic Algorithms are applied extremely widely, from engineering design and scheduling to machine learning, bioinformatics, and finance, owing to their derivative-free, population-based, and highly general nature.

4.14 Review Questions

136. Explain the biological background and basic terminology (chromosome, gene, allele, genotype, phenotype, fitness) used in Genetic Algorithms.
137. How do Genetic Algorithms differ from traditional gradient-based optimization methods? Explain with reference to Goldberg's four fundamental differences.
138. Describe the binary, real-valued, and permutation encoding schemes with suitable examples.
139. Explain roulette-wheel selection and tournament selection with numerical examples.
140. What is elitism? Why is it used in Genetic Algorithms?
141. Describe single-point, two-point, and uniform crossover with a suitable example, and discuss their relative disruptiveness.
142. Explain bit-flip mutation and swap mutation with examples. What role does mutation play in a GA?
143. Draw and explain the Genetic Algorithm cycle in detail.
144. What are the common termination criteria used in Genetic Algorithms?
145. Define fitness function. List and explain the properties of a good fitness function.
146. Explain any two fitness scaling techniques used in Genetic Algorithms.
147. Solve a numerical example to demonstrate one complete cycle of a Genetic Algorithm (initialization, selection, crossover, mutation, and new generation) for maximizing $f(x) = x^2$.

148. State and explain the Schema Theorem and the Building Block Hypothesis.

149. List and explain any six real-world applications of Genetic Algorithms.

150. Compare Genetic Algorithms with Particle Swarm Optimization and traditional hill-climbing methods.

151. What are the advantages and limitations of Genetic Algorithms?

4.15 Continuing the Worked Example - Second Generation

Continuing the numerical example of Section 4.8, recall that Generation 1 produced offspring O1 (x=27, fitness 729), O2 (x=17, fitness 289), O3 (x=25, fitness 625), and O4 (x=12, fitness 144), with total fitness 1787 and average fitness 446.75. Applying the same GA cycle to this generation illustrates how the population continues to converge toward the optimum at x = 31.

Selection for Generation 2

Selection probabilities are p(O1) = 729/1787 = 0.408, p(O2) = 289/1787 = 0.162, p(O3) = 625/1787 = 0.350, p(O4) = 144/1787 = 0.081. As expected, O1 and O3 - the two fittest offspring from Generation 1 - dominate the roulette wheel; suppose the mating pool selected is {O1, O3, O1, O2}.

Crossover for Generation 2

Pairing (O1, O3) = (11011, 11001): a crossover point after gene 4 gives children 1101 1 = 11011 (x=27, unchanged) and 1100 1 = 11001 (x=25, unchanged) in this particular case, since the two parents already agree on their first four genes. Pairing (O1, O2) = (11011, 10001): a crossover point after gene 2 gives children 11 001 = 11001 (x=25, fitness 625) and 10 011 = 10011 (x=19, fitness 361).

Mutation and New Population

Suppose a single mutation flips the last bit of the fourth child (10011 -> 10010, x=18, fitness 324). The Generation 2 population becomes {11011 (x=27, f=729), 11001 (x=25, f=625), 11001 (x=25, f=625), 10010 (x=18, f=324)}, with total fitness 2303 and average fitness 575.75 - again a clear improvement over Generation 1's average of 446.75, and the best individual (x=27, fitness 729) has been preserved in this run, consistent with the expected effect of elitist-leaning selection pressure even without explicit elitism. Repeating this cycle for further generations would be expected to continue concentrating the population increasingly around x = 31, the true maximum of f(x) = x^2 on this domain.

4.16 Case Study - Genetic Algorithm for the Travelling Salesman Problem (TSP)

The Travelling Salesman Problem - find the shortest possible route that visits every city in a given list exactly once and returns to the starting city - is a classic combinatorial optimization problem and a standard illustration of permutation-encoded Genetic Algorithms (Section 4.2.2(c)).

4.16.1 Encoding

Each chromosome is a permutation of the city indices, e.g., for 5 cities labelled 1 to 5, a chromosome might read 3-1-4-2-5, representing the tour: start at city 3, then visit city 1, then city 4, then city 2, then city 5, and finally return to city 3.

4.16.2 Fitness Function

The fitness function is naturally derived from the total length (or total travel cost) of the tour represented by the chromosome:

TourLength = Sum over i=1 to n of distance(city_i, city_(i+1 mod n))

Since shorter tours are better, fitness is typically defined as the reciprocal of tour length (Fitness = 1 / TourLength) or as a large constant minus the tour length, so that the GA's convention of higher fitness is better is preserved for consistency with the selection operators of Section 4.3.1.

4.16.3 Operators

- Selection: tournament selection is commonly preferred for TSP instances, since it does not require fitness scaling and works well even with the highly variable tour-length fitness landscape.
- Crossover: standard single-point or uniform crossover cannot be used directly, because naively exchanging segments would produce offspring with repeated or missing cities; instead, Order Crossover (OX) or Partially Mapped Crossover (PMX), introduced in Section 4.4.1(e), are used to guarantee valid permutations.
- Mutation: swap mutation and inversion mutation (Section 4.5.1(b) and (e)) are the natural choices, since both preserve permutation validity while introducing useful local perturbations to the tour order.

This case study demonstrates a general principle emphasized throughout Sections 4.2 to 4.5: the encoding scheme dictates which crossover and mutation operators are valid, and correctly

matching operator to representation is essential for a Genetic Algorithm to function correctly on a given problem class.

4.17 Multi-Objective Genetic Algorithms - Brief Overview

Many real engineering problems involve several conflicting objectives that cannot be reduced to a single scalar fitness value without losing important trade-off information - for example, simultaneously minimizing cost and maximizing reliability in a design problem. Multi-objective Genetic Algorithms extend the basic GA framework using the concept of Pareto dominance: a solution A is said to dominate solution B if A is at least as good as B in every objective and strictly better in at least one objective. The goal of a multi-objective GA is not to find a single optimum, but to discover a well-distributed set of non-dominated (Pareto-optimal) solutions, collectively called the Pareto front, from which a human decision maker can subsequently choose according to their own priorities - an approach conceptually related to the fuzzy multi-criteria decision-making methods of Section 3.3.

NSGA-II (Non-dominated Sorting Genetic Algorithm II), developed by Deb et al., is the most widely used multi-objective GA. It introduces fast non-dominated sorting to rank the population into successive Pareto fronts, and a crowding-distance metric to preserve diversity along each front, replacing the single fitness-based selection of Section 4.3.1 with a combined rank-and-diversity based selection criterion, while retaining the same crossover and mutation operators described in Sections 4.4 and 4.5.

4.18 Hybrid Genetic Algorithms (Memetic Algorithms)

A Hybrid GA, also known as a Memetic Algorithm, combines the global, population-based search of a standard Genetic Algorithm with a local search or refinement procedure (such as hill climbing, simulated annealing, or a problem-specific heuristic) applied to individuals after crossover and mutation, before they are evaluated and inserted into the new generation. The GA component provides broad, global exploration of the search space (Section 4.1.2), while the local search component rapidly exploits and refines promising solutions discovered by the GA, often producing better final solutions, or the same quality of solution in fewer generations, than either technique used alone. Memetic algorithms are widely applied to combinatorial problems such as TSP (Section 4.16) and scheduling, where efficient, problem-specific local search moves are readily available.

4.19 Solved Problems

Problem 1

A population of 4 chromosomes has fitness values 25, 40, 10, 25 (total = 100). Compute the roulette-wheel selection probability of each individual, and state which individual is least likely to be selected.

Solution: p1 = 25/100 = 0.25, p2 = 40/100 = 0.40, p3 = 10/100 = 0.10, p4 = 25/100 = 0.25. The individual with fitness 10 (the third chromosome) has the lowest selection probability (0.10) and is therefore the least likely to be selected as a parent, though - consistent with the discussion in Section 4.3.1 - it retains a non-zero chance and is not automatically excluded, preserving some genetic diversity.

Problem 2

Two parent chromosomes P1 = 110101 and P2 = 001110 undergo single-point crossover at a cut point after gene 3. Write the resulting offspring.

Solution: P1 = 110 | 101 and P2 = 001 | 110. Exchanging the segments after the cut point gives Offspring 1 = 110 110 = 110110 and Offspring 2 = 001 101 = 001101, exactly following the single-point crossover mechanism described in Section 4.4.1(a).

Problem 3

A chromosome 101100 undergoes bit-flip mutation with pm = 0.1 applied independently to every gene. If a random-number generator produces the values 0.05, 0.62, 0.08, 0.77, 0.41, 0.03 for the six genes respectively, determine the mutated chromosome (a gene is flipped only if its random number is less than pm).

Solution: Comparing each random number to pm = 0.1: gene 1 (0.05 < 0.1) -> flip 1 to 0; gene 2 (0.62 >= 0.1) -> unchanged (0); gene 3 (0.08 < 0.1) -> flip 1 to 0; gene 4 (0.77 >= 0.1) -> unchanged (1); gene 5 (0.41 >= 0.1) -> unchanged (0); gene 6 (0.03 < 0.1) -> flip 0 to 1. The mutated chromosome is therefore 000101, illustrating the independent, per-gene probabilistic nature of bit-flip mutation described in Section 4.5.1(a).

4.20 Key Formulae at a Glance

Concept	Formula / Expression
Roulette-wheel selection probability	pi = fi / (sum of fj)
Linear fitness scaling	f' = a.f + b
Sigma truncation scaling	f' = f - (f_avg - c.sigma)

Concept	Formula / Expression
Penalty-based fitness (constrained problems)	Fitness(x) = Objective(x) - sum[Li . max(0, gi(x))]
Arithmetic crossover (real-valued)	child = alpha.parent1 + (1-alpha).parent2
Schema theorem (simplified)	m(H,t+1) >= m(H,t).[f(H)/f_avg].[1 - pc.d(H)/(L-1) - o(H).pm]

4.21 Case Study - Using a Genetic Algorithm to Train a Neural Network (Neuroevolution)

A widely cited application of Genetic Algorithms outside classical numerical optimization is neuroevolution - the use of a GA to optimize the weights (and, in more advanced schemes, the architecture) of an artificial neural network, as an alternative or a complement to gradient-based training methods such as back-propagation. This case study illustrates how the general GA cycle of Section 4.6 is instantiated for a machine-learning problem.

4.21.1 Encoding

Each chromosome represents one complete set of weights and biases for a fixed neural-network architecture, using real-valued encoding (Section 4.2.2(b)). For example, a small network with 20 weights and 5 biases would be encoded as a chromosome of 25 real-valued genes, each gene directly storing one weight or bias value, typically constrained to a reasonable range such as [-5, 5].

4.21.2 Fitness Function

The fitness of a chromosome is computed by first decoding it into the corresponding network weights, then evaluating the resulting network's performance on a training (or validation) dataset - typically the negative of the mean-squared error for a regression task, or the classification accuracy for a classification task:

Fitness(chromosome) = 1 / (1 + MeanSquaredError(network_output, target_output))

This formulation ensures that fitness increases monotonically as the network's prediction error decreases, remaining bounded and well-behaved (between 0 and 1) even for very poor initial random weight configurations.

4.21.3 Operators

- Selection: tournament selection is commonly used, since network-evaluation fitness values can vary over a very wide numeric range, which can distort roulette-wheel probabilities.

- Crossover: arithmetic (blend) crossover (Section 4.4.1(d)) is typically preferred over simple single-point crossover for real-valued weight chromosomes, since it produces smoother, more meaningful offspring weight combinations.
- Mutation: Gaussian mutation (Section 4.5.1(d)) is applied to perturb individual weights slightly, allowing fine-tuning of the evolved network without completely disrupting an already good solution.

Neuroevolution is particularly attractive in reinforcement-learning settings, where the objective (the total reward accumulated by an agent) is not directly differentiable with respect to the network's weights, making gradient-based back-propagation difficult to apply directly, whereas a GA - as emphasized throughout Section 4.1.2 - requires only fitness values, not derivatives.

4.22 Practical Guidelines for Choosing Genetic Algorithm Parameters

Selecting appropriate values for population size, crossover probability, and mutation probability (Section 4.9) is largely empirical and problem-dependent, but the following widely cited practical guidelines, distilled from decades of GA research and practice, provide a useful and safe starting point for a new problem:

152. Begin with a moderate population size (roughly 50-100 individuals for small to medium problems), increasing it for higher-dimensional or highly multi-modal search spaces.
153. Set the crossover probability relatively high (pc approximately 0.7-0.9), since crossover is the GA's primary mechanism for combining good partial solutions.
154. Set the mutation probability relatively low (pm approximately 1/L, where L is the chromosome length, or in the range 0.001-0.05), since mutation is intended as a background diversity-preserving operator rather than the primary search driver.
155. Employ elitism (Section 4.3.1(d)) by retaining at least the single best individual unchanged each generation, which reliably improves convergence without materially harming diversity.
156. Monitor population diversity (e.g., the standard deviation of fitness, or of gene values) over generations; if diversity collapses too early (premature convergence), consider increasing pm, increasing population size, or introducing periodic re-initialization (restart) of a fraction of the population.

157. Always run the GA multiple times with different random seeds, since results can vary between runs due to the algorithm's inherent stochasticity, and report best, average, and worst-case performance rather than a single run's result.

4.22.1 Common Pitfalls in Applying Genetic Algorithms

- Choosing an encoding that does not match the natural structure of the problem, forcing the use of inappropriate or ad-hoc crossover/mutation operators (Section 4.16.3).
- Designing a fitness function that is too flat (insufficient discrimination, Section 4.7.1) or too rugged/deceptive, both of which make effective search difficult.
- Using excessively high mutation rates, effectively degrading the GA into an inefficient random search (Section 4.5.2).
- Neglecting to handle constraints properly, allowing the GA to repeatedly generate and waste evaluations on infeasible solutions rather than applying appropriate penalty or repair mechanisms (Section 4.7.2(d)).
- Terminating the run too early, before the population has had sufficient generations to converge toward a high-quality solution, or conversely running for far more generations than necessary once convergence has clearly plateaued.

4.23 Additional Solved Problems

Problem 4

A GA uses tournament selection with tournament size k = 3 on a population with fitness values {12, 45, 30, 8, 50, 22}. If the individuals randomly selected for one tournament are those with fitness 12, 30, and 50, which individual wins the tournament and becomes a parent?

Solution: In tournament selection (Section 4.3.1(b)), the individual with the highest fitness among those randomly drawn into the tournament wins. Among {12, 30, 50}, the individual with fitness 50 has the highest fitness and therefore wins this tournament and is selected as a parent, illustrating how tournament selection applies direct, local competition rather than a global proportionate probability as in roulette-wheel selection.

Problem 5

A permutation-encoded chromosome for a 6-city TSP instance is 2-5-1-4-3-6. Apply swap mutation by exchanging the symbols at positions 2 and 5. Write the mutated chromosome.

Solution: The original chromosome, with 1-based positions, is: position 1 = 2, position 2 = 5, position 3 = 1, position 4 = 4, position 5 = 3, position 6 = 6. Swapping the values at positions 2 and 5 (5 and 3) gives the mutated chromosome 2-3-1-4-5-6, which remains a valid permutation of all six cities, consistent with the permutation-preserving property of swap mutation described in Section 4.5.1(b).

Problem 6

Using sigma truncation scaling with f_avg = 40, sigma = 10, and c = 2, compute the scaled fitness of an individual with raw fitness f = 25.

Solution: Using the sigma truncation formula of Section 4.7.2(b), f' = f - (f_avg - c.sigma) = 25 - (40 - 2x10) = 25 - 20 = 5. Since the result is positive, the scaled fitness remains f' = 5 (had the computation produced a negative value, it would instead be truncated to zero, ensuring that scaled fitness values used for selection are never negative).

4.24 Parallel and Distributed Genetic Algorithms

When the population size or the number of generations required for a problem becomes very large, running a Genetic Algorithm on a single processor can be computationally slow. Because the fitness evaluation of each individual in a generation (Section 4.6.1, Step 2) is typically independent of every other individual, GAs are naturally well suited to parallelization. The most widely used parallel model is the Island (or Coarse-Grained) Model.

4.24.1 The Island Model

In the island model, the total population is divided into several smaller sub-populations (islands), each of which evolves largely independently, running its own complete GA cycle (selection, crossover, mutation) on its own processor or thread. Periodically, a small number of the fittest individuals migrate between islands according to a chosen migration topology (commonly a ring, as illustrated in Fig. 4.9, or a fully connected topology), introducing fresh genetic material into each sub-population and helping to prevent each island from prematurely converging to its own, possibly inferior, local optimum.

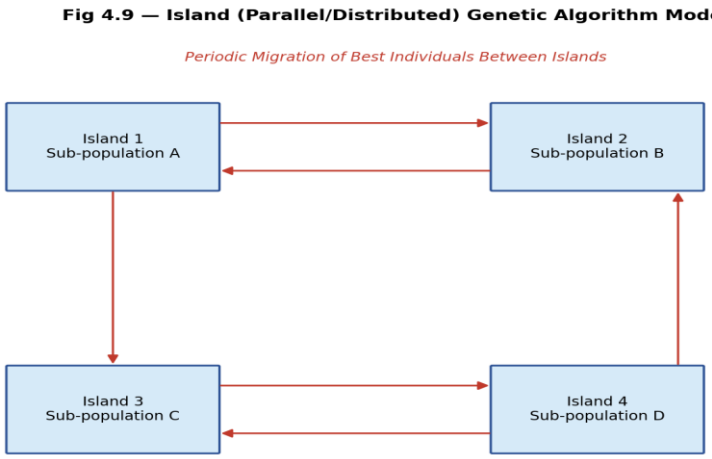

Fig. 4.9 - The island (parallel/distributed) Genetic Algorithm model: sub-populations evolve independently on separate islands, with periodic migration of the fittest individuals between neighbouring islands.

4.24.2 Key Parameters of the Island Model

Parameter	Description
Number of islands	How many independent sub-populations are maintained; more islands increase parallelism but reduce each island's own population size.
Migration interval	How many generations elapse between successive migration events; frequent migration behaves more like a single large population, while infrequent migration allows islands to diverge and specialize.
Migration rate	The number (or fraction) of individuals that migrate between islands at each migration event.
Migration topology	The pattern of connections along which individuals migrate (ring, fully connected, random, or grid/Von Neumann), analogous to the lbest neighbourhood topologies used in PSO (Section 3.9(b)).

Besides reducing wall-clock computation time through parallel execution, the island model has also been observed, in many studies, to improve overall solution quality compared with an equivalent single large population, precisely because the semi-isolated islands maintain greater genetic diversity for longer, reducing the risk of the premature convergence discussed for both GA and PSO in Sections 4.9.1 and 3.10.2 respectively.

4.24.3 Fine-Grained (Cellular) Genetic Algorithms

An alternative parallel model, the Fine-Grained (or Cellular) GA, arranges the entire population on a two-dimensional grid, with each individual occupying one cell and interacting (for selection and crossover) only with its immediate neighbours on the grid, rather than with the population as a whole. This produces a smooth, spatially localized spread of good solutions across the grid over successive generations, offering yet another mechanism for preserving

diversity while still allowing beneficial genetic material to eventually propagate throughout the entire population.

4.25 Handling Constraints in Genetic Algorithms

Many practical optimization problems are constrained - only a subset of the search space represents feasible (valid) solutions. Because the basic GA operators of Sections 4.3-4.5 do not inherently guarantee that offspring remain feasible, several complementary strategies are used to handle constraints effectively:

(a) Penalty Functions

As introduced in Section 4.7.2(d), the fitness function is modified to subtract a penalty proportional to the degree of constraint violation, discouraging (but not strictly forbidding) infeasible solutions, and allowing the GA to potentially pass through infeasible regions of the search space en route to a good feasible solution.

(b) Repair Mechanisms

After crossover or mutation produces an infeasible offspring, a problem-specific repair procedure is applied to modify the offspring minimally until it becomes feasible again - for example, in a knapsack problem, items may be removed one at a time (e.g., in increasing order of value-to-weight ratio) from an over-capacity solution until the weight constraint is satisfied.

(c) Decoders (Indirect Encoding)

Rather than encoding the solution directly, the chromosome encodes an indirect set of instructions or priorities that a deterministic decoding procedure then translates into a guaranteed-feasible solution - for example, in scheduling problems, a chromosome might encode only a priority ordering of jobs, with a separate feasible-schedule-construction algorithm handling the actual, always-feasible, assignment of jobs to time slots.

(d) Feasibility-Preserving Operators

As already seen for permutation encoding in Sections 4.4.1(e) and 4.5.1(b),(e), specially designed crossover and mutation operators can be constructed that are guaranteed, by their very definition, to always produce feasible offspring from feasible parents - the preferred approach whenever such operators can be practically designed for the problem at hand.

4.26 Comparative Summary of Encoding Schemes

Encoding	Best Suited For	Typical Crossover	Typical Mutation
Binary	Discrete/combinatorial problems, classical theoretical analysis	Single-point, two-point, uniform	Bit-flip
Real-valued	Continuous numerical optimization	Arithmetic (blend), simulated binary crossover (SBX)	Gaussian, uniform reset
Permutation	Ordering/sequencing problems (TSP, scheduling)	Order crossover (OX), PMX	Swap, inversion
Tree	Genetic Programming (evolving expressions/programs)	Subtree exchange	Subtree replacement, point mutation

4.27 Final Chapter-End Practice Problems

Problem 7

An island-model GA runs 4 islands of 25 individuals each with a ring migration topology and a migration interval of 10 generations. If the migration rate is 2 individuals per migration event, how many individuals migrate out of the entire system in total during one migration event, and how does the ring topology in Fig. 4.9 route them?

Solution: With 4 islands and a migration rate of 2 individuals per island per migration event, a total of 4 x 2 = 8 individuals migrate during one migration event. Under the ring topology of Fig. 4.9, each island sends its 2 fittest migrating individuals to exactly one neighbouring island (e.g., Island 1 to Island 2, Island 2 to Island 3, Island 3 to Island 4, and Island 4 back to Island 1), so every island simultaneously sends 2 individuals out and receives 2 individuals in, keeping each island's population size constant at 25 after the migration event.

Problem 8

A knapsack-problem chromosome represents item selection as a binary string, but crossover has produced an offspring whose total weight exceeds the knapsack's capacity. Describe, step by step, how a repair mechanism (Section 4.25(b)) could restore feasibility.

Solution: A typical repair procedure first computes the value-to-weight ratio of every item currently selected (gene = 1) in the infeasible offspring. It then repeatedly removes (sets to 0) the selected item with the lowest value-to-weight ratio, recomputing the total weight after each removal, and continues this process until the total weight of the remaining selected items no longer exceeds the knapsack's capacity. This greedy repair strategy restores feasibility while

attempting to preserve as much of the offspring's total value as possible, illustrating the general repair-mechanism approach described in Section 4.25(b).

UNIT – V

ROUGH SETS AND INTEGRATION OF SOFT COMPUTING TECHNIQUES

5.1 Introduction

In Unit II of this course we studied fuzzy set theory, which handles imprecision arising from the vagueness of concepts — the fact that a natural-language term such as 'tall' or 'hot' does not have a sharp boundary. There is, however, a second, quite different source of imprecision that arises very commonly in practical data analysis: imprecision caused not by vague concepts but by insufficient or incomplete information about the objects being studied. For example, if two patients present with identical values for every symptom that a hospital happens to record, they will appear indistinguishable to the hospital's information system even though they may, in reality, have different underlying conditions. Rough Set Theory, introduced by the Polish mathematician and computer scientist Zdzisław Pawlak in his seminal 1982 paper 'Rough Sets', was developed specifically to formalise and reason about this second kind of imprecision — imprecision due to the granularity (coarseness) of available information — using purely crisp (non-fuzzy) mathematical machinery.

This unit introduces the foundational concepts of rough set theory — information systems, the indiscernibility relation, and lower/upper set approximation — and builds on them to study reduct and core computation via the discernibility matrix, the induction of decision rules from data, and finally the broader theme of integrating rough sets with the other soft computing techniques (fuzzy logic, neural networks, and genetic algorithms) studied earlier in this course, to build powerful hybrid intelligent systems.

5.2 Information Systems and Decision Tables

Rough set theory represents knowledge about a domain in the form of an information system (also called an attribute-value system or, informally, a data table). Formally, an information system is a pair S = (U, A), where U = {x1, x2, ..., xn} is a finite, non-empty set of objects called the universe, and A = {a1, a2, ..., am} is a finite, non-empty set of attributes, such that every attribute a ∈ A is a function a: U → Va, where Va is the set of values (the domain) that attribute a can take.

When one of the attributes is distinguished as a special decision attribute d (representing a classification, diagnosis, or outcome that we wish to predict or explain), and the remaining attributes C = A \ {d} are called condition attributes, the information system is called a decision table, written S = (U, C ∪ {d}). Decision tables are the standard starting point for rule induction (Section 5.8) and for most practical applications of rough set theory in data mining and knowledge discovery.

5.2.1 Running Example — A Medical Decision Table

To make the concepts in this unit concrete, we introduce a single running example that will be used throughout the unit: a small medical decision table recording, for eight patients, whether they report a Headache (H), whether they report Muscle-pain (M), their body Temperature (T), and the final diagnosis, Flu (the decision attribute), based on Pawlak's classic illustration of rough sets in medical diagnosis.

Patient	Headache (H)	Muscle-pain (M)	Temperature (T)	Flu (Decision)
x1	Yes	Yes	High	Yes
x2	Yes	Yes	Very High	Yes
x3	Yes	Yes	Normal	Yes
x4	Yes	No	High	Yes
x5	Yes	No	Normal	No
x6	No	Yes	Very High	No
x7	No	Yes	Normal	No
x8	No	No	High	No

Here U = {x1,...,x8}, the condition attribute set is C = {H, M, T}, and the decision attribute is d = Flu ∈ {Yes, No}. We will return to this table repeatedly in Sections 5.3 through 5.8 to illustrate the indiscernibility relation, set approximation, accuracy of approximation, and rule induction.

5.3 The Indiscernibility Relation

The central, defining idea of rough set theory is the indiscernibility relation. Given an information system S = (U, A) and any non-empty subset of attributes B ⊆ A, two objects x, y ∈ U are said to be B-indiscernible (written x IND(B) y) if and only if they have identical values for every attribute in B: a(x) = a(y) for every a ∈ B. The relation IND(B) is easily seen to be an equivalence relation (reflexive, symmetric, and transitive), and it therefore partitions the universe U into a collection of disjoint equivalence classes, denoted U/IND(B), often simply

written U/B. Each equivalence class [x]_B consists of all objects that are indistinguishable from x when only the attributes in B are considered.

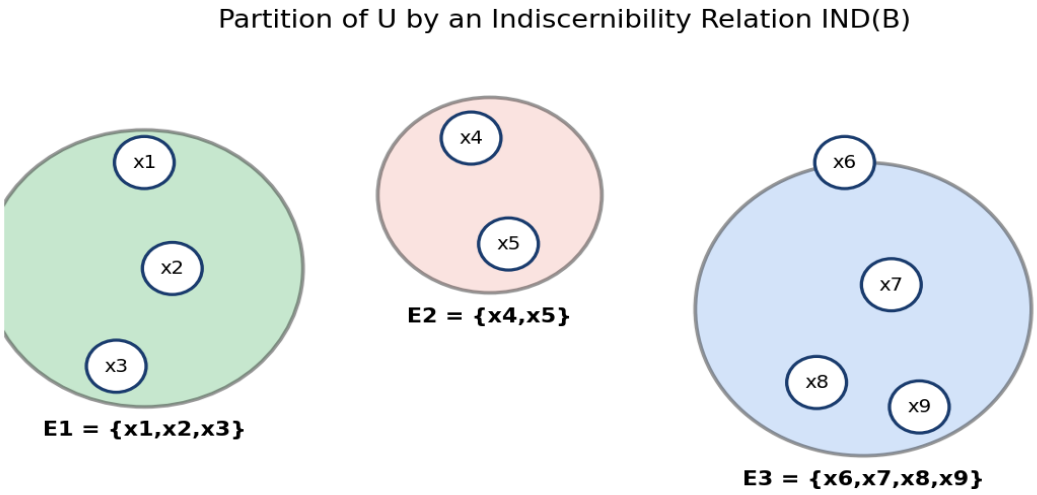

Fig. 5.1 – Partition of the Universe U into Equivalence Classes under IND(B)

Applying this to our running example with B = {H, M} (i.e. considering only Headache and Muscle-pain, and temporarily ignoring Temperature), we obtain the following partition of U:

Equivalence class (H, M)	Members
(Yes, Yes)	{x1, x2, x3}
(Yes, No)	{x4, x5}
(No, Yes)	{x6, x7}
(No, No)	{x8}

Notice that if instead we take B = C = {H, M, T} (the full condition attribute set), every one of the 8 patients has a unique combination of (H, M, T) values, so IND(C) partitions U into 8 singleton classes — with full information, no two distinct patients are indiscernible. This illustrates a general principle: adding more attributes to B can only refine (or leave unchanged) the partition U/IND(B); it can never coarsen it.

5.4 Set Approximation: Lower and Upper Approximations

Suppose we are given an arbitrary (possibly imprecise) target concept, expressed as a subset X ⊆ U — for example, X = the set of patients who actually have the flu. Using only the

information available through a chosen attribute subset B, we generally cannot describe X exactly, because X may 'cut across' some of the equivalence classes of U/IND(B) (i.e. some classes may contain some objects in X and some not in X). Rough set theory resolves this by approximating X from below and from above using two crisp sets built entirely out of B-equivalence classes.

Approximation	Definition
B-Lower approximation, B_(X)	The union of all equivalence classes of U/IND(B) that are entirely contained in X: B_(X) = ∪ { [x]_B ∈ U/B | [x]_B ⊆ X }. Consists of objects that certainly / definitely belong to X.
B-Upper approximation, B̄(X)	The union of all equivalence classes of U/IND(B) that have a non-empty intersection with X: B̄(X) = ∪ { [x]_B ∈ U/B | [x]_B ∩ X ≠ ∅ }. Consists of objects that possibly belong to X.

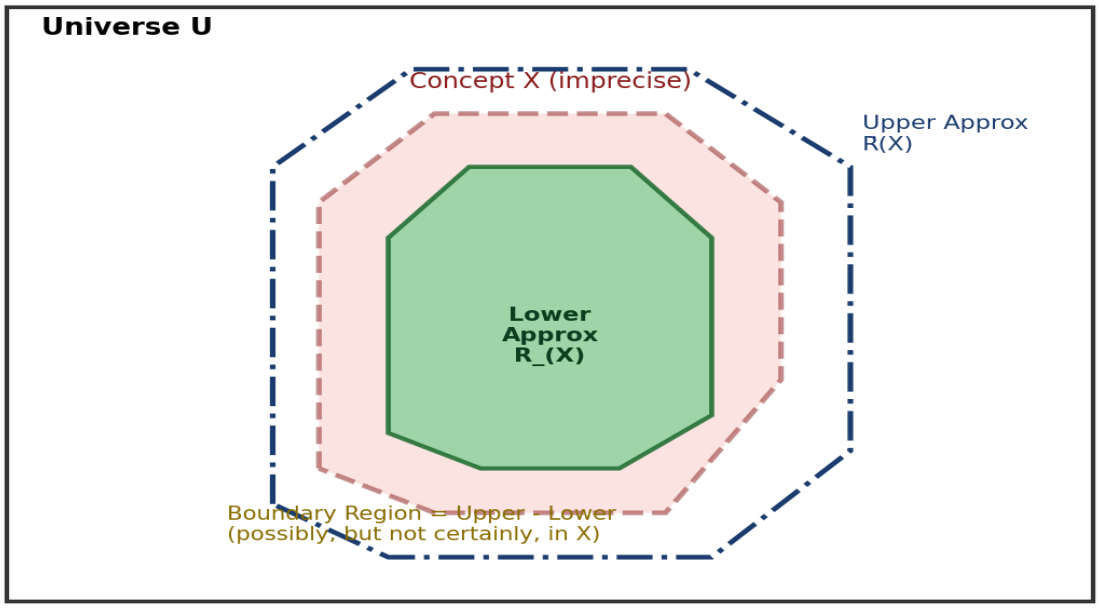

Fig. 5.2 – Lower Approximation, Upper Approximation, and Boundary Region of a Concept X

5.4.1 Worked Example — Approximating 'Flu = Yes' using B = {H, M}

Let X = {x1, x2, x3, x4} be the set of patients whose actual diagnosis is Flu = Yes (read directly from the decision table in Section 5.2.1). Using the partition U/IND(B) computed in Section 5.3 for B = {H, M}, we check each equivalence class against X:

- Class (Yes, Yes) = {x1, x2, x3}: all three members are in X (all have Flu = Yes) → entirely contained in X → contributes to the lower approximation.

- Class (Yes, No) = {x4, x5}: x4 ∈ X (Flu = Yes) but x5 ∉ X (Flu = No) → mixed, partially in X → contributes only to the upper approximation, not the lower.
- Class (No, Yes) = {x6, x7}: neither member is in X (both Flu = No) → entirely outside X → does not contribute to either approximation of X.
- Class (No, No) = {x8}: not in X (Flu = No) → entirely outside X → does not contribute to either approximation of X.

Therefore: B_(X) = {x1, x2, x3} (patients who certainly have the flu, based on Headache and Muscle-pain alone), and $\bar{B}(X)$ = {x1, x2, x3, x4, x5} (patients who possibly have the flu). The boundary region, defined below, is BND_B(X) = $\bar{B}(X)$ − B_(X) = {x4, x5} — these are the patients whose Headache/Muscle-pain profile is shared by at least one flu-positive and at least one flu-negative patient, so we cannot be certain about their diagnosis using only H and M.

5.5 Rough Sets, Boundary Region, and Accuracy of Approximation

5.5.1 Definition of a Rough Set

The boundary region of X with respect to B is defined as BND_B(X) = $\bar{B}(X)$ − B_(X), i.e. the set of objects that cannot be classified with certainty, using the attributes in B, as either belonging to X or not belonging to X. If BND_B(X) = ∅ (i.e. B_(X) = $\bar{B}(X)$), the set X is said to be B-definable (or crisp/exact with respect to B) — it can be expressed exactly as a union of B-equivalence classes, with no ambiguity. If, on the other hand, BND_B(X) ≠ ∅, then X cannot be precisely characterised using the attributes in B, and X is called a rough set (with respect to B). It is precisely this — the impossibility of exact characterisation using the available (granular) attribute information — that rough set theory sets out to formalise and quantify.

In our worked example (Section 5.4.1), since BND_B(X) = {x4, x5} ≠ ∅, the concept X = 'Flu = Yes' is a rough set with respect to B = {H, M}: we cannot say with certainty, using only Headache and Muscle-pain, whether x4 and x5 have the flu.

5.5.2 Positive, Negative, and Boundary Regions

Region	Definition	Interpretation
Positive region, POS_B(X)	POS_B(X) = B_(X)	Objects certainly belonging to X
Negative region, NEG_B(X)	NEG_B(X) = U − $\bar{B}(X)$	Objects certainly NOT belonging to X
Boundary region, BND_B(X)	BND_B(X) = $\bar{B}(X)$ − B_(X)	Objects that cannot be classified with certainty, either way

In the worked example: POS_B(X) = {x1,x2,x3}; NEG_B(X) = U − {x1,x2,x3,x4,x5} = {x6,x7,x8}; BND_B(X) = {x4,x5}. These three regions always partition the universe U completely and without overlap: U = POS_B(X) ∪ BND_B(X) ∪ NEG_B(X).

5.5.3 Accuracy and Quality of Approximation

Rough set theory provides two standard numerical measures of how well a concept X can be approximated using a given attribute subset B.

Measure	Formula	Range / Interpretation
Accuracy of approximation, $\alpha_B(X)$	$\alpha_B(X) = \lvert B_(X)\rvert / \lvert \bar{B}(X)\rvert$ (assuming $\bar{B}(X) \neq \emptyset$)	$0 \le \alpha_B(X) \le 1$. $\alpha_B(X)=1$ means X is B-definable (crisp); smaller values indicate greater vagueness/roughness.
Quality of approximation, $\gamma_B(X)$	$\gamma_B(X) = \lvert B_(X)\rvert / \lvert U\rvert$	$0 \le \gamma_B(X) \le 1$. Represents the fraction of all objects in U that can be correctly, certainly classified as belonging to X using attribute subset B.

For the worked example: $|B_(X)| = |\{x1,x2,x3\}| = 3$, $|\bar{B}(X)| = |\{x1,x2,x3,x4,x5\}| = 5$, and $|U| = 8$. Hence the accuracy of approximation is $\alpha_B(X) = 3/5 = 0.6$, and the quality of approximation is $\gamma_B(X) = 3/8 = 0.375$. An accuracy of 0.6 tells us that using only Headache and Muscle-pain, 60% of the patients who are 'possibly flu-positive' can, in fact, be said with full certainty to be flu-positive; the remaining 40% (i.e. the two boundary patients, x4 and x5) remain ambiguous and would require additional attributes (such as Temperature) to resolve.

5.6 Reduct and Core

A natural and practically important question in rough set analysis is: are all of the available condition attributes actually necessary, or can some of them be removed without losing any classificatory information? Reduct and core are the two central concepts that answer this question.

5.6.1 Reduct

A subset of attributes $R \subseteq C$ is called a reduct of C (with respect to the classification/decision at hand) if (i) R preserves the same classificatory power as the full attribute set C (formally, for the classical/global reduct, IND(R) = IND(C); for the decision-relative reduct discussed further in Section 5.7, R preserves the same positive region as C, i.e. POS_R(d) = POS_C(d)), and (ii) R is minimal — no attribute can be removed from R without destroying this property. In general, an information system may possess several different reducts, each an equally valid,

minimal, non-redundant representation of the original attribute set's classificatory information; a given information system typically does not have a unique reduct.

5.6.2 Core

The core of C is defined as the intersection of all reducts of C: CORE(C) = ∩ (all reducts of C). An attribute belongs to the core if and only if it appears in every reduct — equivalently, if and only if removing that single attribute alone (while keeping all others) actually changes the classification (i.e. that attribute is indispensable and cannot be dropped under any circumstance without loss of information). If an information system has only one reduct, then that reduct is itself equal to the core. If there are multiple reducts with no attribute common to all of them, the core is the empty set.

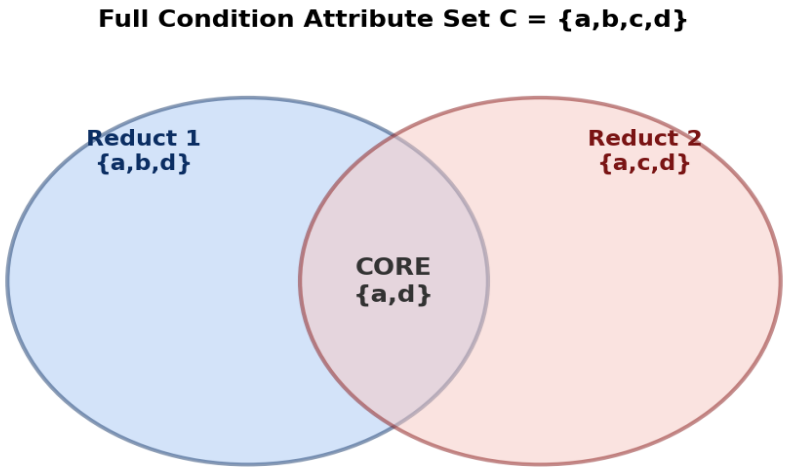

Fig. 5.3 – Reducts as Minimal Attribute Subsets; the Core as Their Intersection

Reducts are of immense practical importance in data mining and machine learning, because they allow a dataset with many attributes (which may be expensive, redundant, or noisy to collect/measure) to be reduced to the smallest possible subset of attributes without any loss of classification accuracy — a form of principled, information-theoretically justified feature selection. The systematic way to compute all reducts of an information system is via the discernibility matrix, introduced next.

5.7 The Discernibility Matrix and Discernibility Function

The discernibility matrix, introduced by Skowron and Rauszer, provides a systematic, purely combinatorial method for computing all the reducts (and hence the core) of an information system, rather than having to test every possible attribute subset by trial and error.

5.7.1 Definition

Given an information system S = (U, A) with U = {x1,...,xn}, the discernibility matrix is an n × n matrix M, where each entry c_ij (for objects x_i and x_j) is the set of all attributes that distinguish (discern) x_i from x_j: c_ij = { a ∈ A | a(x_i) ≠ a(x_j) }. If x_i and x_j are already indiscernible using all attributes in A (i.e. c_ij = ∅), this simply means the two objects are identical with respect to every recorded attribute.

5.7.2 Worked Example — A Fresh, Minimal Decision Table

To illustrate the discernibility matrix and reduct computation clearly, we use a small, dedicated example with 4 objects, 3 binary condition attributes (a, b, c), and a binary decision attribute d.

Object	a	b	c	d (Decision)
x1	0	0	0	1
x2	0	1	1	1
x3	1	0	1	0
x4	1	1	0	0

Comparing every pair of objects and recording which attributes differ, we obtain the (full, classical) discernibility matrix entries: c_12 = {b,c} (a same, b and c differ); c_13 = {a,c}; c_14 = {a,b}; c_23 = {a,b}; c_24 = {a,c}; c_34 = {b,c}.

Pair	Differing Attributes c_ij
(x1,x2)	{b, c}
(x1,x3)	{a, c}
(x1,x4)	{a, b}
(x2,x3)	{a, b}
(x2,x4)	{a, c}
(x3,x4)	{b, c}

5.7.3 The Discernibility Function and Classical Reducts

The discernibility function f(A) is a Boolean expression constructed by taking the conjunction (AND), over all non-empty entries c_ij of the discernibility matrix, of the disjunction (OR) of the attributes in that entry: f(A) = ⋀_{i<j, c_ij≠∅} (⋁_{a ∈ c_ij} a). Reducing this Boolean

expression to its minimal disjunctive normal form (i.e. expressing it as an OR of ANDs, and keeping only the shortest/minimal terms) yields exactly the set of all reducts of the information system: each minimal conjunctive term corresponds to one reduct.

For our example, using all six pairs: f(A) = (b∨c) ∧ (a∨c) ∧ (a∨b) ∧ (a∨b) ∧ (a∨c) ∧ (b∨c) = (a∨b) ∧ (a∨c) ∧ (b∨c) (removing duplicate clauses). Expanding this Boolean expression into its minimal disjunctive normal form gives f(A) = (a∧b) ∨ (a∧c) ∨ (b∧c). Hence the classical reducts of this information system are {a,b}, {a,c}, and {b,c} — that is, any two of the three condition attributes suffice to discern every pair of objects from one another; no single attribute alone is sufficient. Since no attribute appears in all three reducts, CORE(A) = {a,b} ∩ {a,c} ∩ {b,c} = ∅ in this classical (full-discernibility) sense.

5.7.4 Decision-Relative Reducts

In most practical applications (as in our decision table above), we are not actually interested in discerning every pair of objects from each other — we only care about discerning pairs of objects that have different decision values, since objects sharing the same decision may harmlessly remain indiscernible. This leads to the decision-relative discernibility matrix, in which we only include an entry c_ij when d(x_i) ≠ d(x_j) (i.e. the two objects have different decisions), and c_ij continues to record the condition attributes that differ between them (attribute d itself is never included).

Reconsidering our example: the decision-differing pairs are (x1,x3), (x1,x4), (x2,x3), and (x2,x4) (since x1,x2 share d=1 and x3,x4 share d=0). The relevant entries are c_13={a,c}, c_14={a,b}, c_23={a,b}, c_24={a,c}. The decision-relative discernibility function is f_d(A) = (a∨c) ∧ (a∨b) ∧ (a∨b) ∧ (a∨c) = (a∨b) ∧ (a∨c). By the distributive law, (a∨b)∧(a∨c) = a ∨ (b∧c). Hence the decision-relative (D-)reducts are simply {a} and {b,c} — attribute 'a' alone is entirely sufficient to determine the decision d for every object (verify: a=0 gives x1,x2, both d=1; a=1 gives x3,x4, both d=0 — consistent!), and alternatively, the pair {b,c} together also suffice.

This example highlights an important and often under-appreciated distinction: the classical reduct (Section 5.7.3) requires enough attributes to tell every object apart from every other object, whereas the decision-relative reduct (this section) only requires enough attributes to correctly predict the decision, and can be considerably smaller. Since {a} and {b,c} share no

common attribute, the decision-relative core is also empty in this particular example — illustrating that a core can indeed be empty even when useful, non-trivial reducts exist.

5.8 Rule Induction from Rough Sets

One of the most valuable practical outcomes of rough set analysis is the automatic induction of IF-THEN decision rules directly from a decision table, without requiring any assumptions about the underlying probability distribution or functional form of the data (unlike many statistical or neural approaches) — the rules follow directly and transparently from the structure of the equivalence classes and approximations already computed.

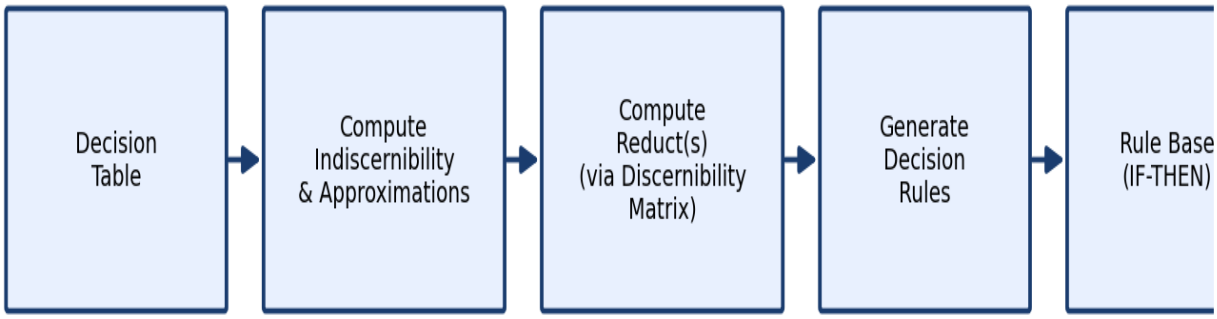

Fig. 5.4 – The Rough-Set-Based Rule Induction Process

5.8.1 Certain Rules and Possible Rules

- A certain (exact) decision rule is derived from an equivalence class that lies entirely within the positive region (lower approximation) of some decision class — that is, every object satisfying the rule's condition part has exactly the same, unambiguous decision value.
- A possible (approximate) decision rule is derived from an equivalence class that lies in the boundary region — such a rule's condition part is satisfied by objects with more than one decision value, and the rule is therefore associated with a certainty factor (also called a rule strength), typically computed as the proportion of objects satisfying the condition that also satisfy the given decision.

5.8.2 Worked Example — Rules Derived from the Medical Decision Table

Returning to the running medical example (Section 5.2.1) with B = {H, M}, recall the four equivalence classes computed in Section 5.3, and their relationship to the decision Flu, established in Section 5.4.1:

- Class {x1,x2,x3} = (H=Yes, M=Yes), entirely Flu=Yes → CERTAIN RULE: IF Headache=Yes AND Muscle-pain=Yes THEN Flu=Yes. (support = 3 objects, certainty factor = 3/3 = 1.0)
- Class {x6,x7} = (H=No, M=Yes), entirely Flu=No → CERTAIN RULE: IF Headache=No AND Muscle-pain=Yes THEN Flu=No. (support = 2, certainty factor = 1.0)
- Class {x8} = (H=No, M=No), entirely Flu=No → CERTAIN RULE: IF Headache=No AND Muscle-pain=No THEN Flu=No. (support = 1, certainty factor = 1.0)
- Class {x4,x5} = (H=Yes, M=No), MIXED (x4 = Flu-Yes, x5 = Flu-No) → two POSSIBLE RULES, each with certainty factor 1/2: IF Headache=Yes AND Muscle-pain=No THEN Flu=Yes (certainty 0.5); and IF Headache=Yes AND Muscle-pain=No THEN Flu=No (certainty 0.5).

As noted in Section 5.4.1, the ambiguity in the last (boundary) class can be resolved by bringing back the previously omitted attribute, Temperature: within the class {x4,x5}, x4 has Temperature=High while x5 has Temperature=Normal — these are distinct attribute values, so adding Temperature as a further condition converts both possible rules into certain rules: IF Headache=Yes AND Muscle-pain=No AND Temperature=High THEN Flu=Yes (certainty 1.0), and IF Headache=Yes AND Muscle-pain=No AND Temperature=Normal THEN Flu=No (certainty 1.0). This demonstrates concretely how the boundary region identifies precisely which objects/situations require additional information (attributes) before a fully certain classification rule can be induced — rough set theory does not merely detect that ambiguity exists, but pinpoints exactly where it lies.

5.8.3 Rule Induction Algorithms — Brief Overview

While the small worked example above was solved by direct inspection, real-world decision tables can have thousands of objects and dozens of attributes, requiring systematic, algorithmic rule-induction procedures. Some well-known rough-set-based rule induction algorithms include:

- LEM2 (Learning from Examples Module, version 2): Induces a minimal set of rules that together cover all objects in the positive region, by greedily selecting, at each step, the attribute-value pair that covers the largest number of currently uncovered objects belonging to the target decision class.
- MODLEM: An extension of LEM2 capable of directly handling continuous (numeric) attributes by dynamically discovering suitable cut-points/discretisation thresholds during rule induction, rather than requiring attributes to be pre-discretised.
- Exhaustive (Boolean-reasoning-based) algorithms: Use the discernibility matrix and function (Section 5.7) directly to enumerate all reducts, and then derive rules from each reduct; guaranteed to find all minimal rules but computationally more expensive for large datasets.

5.9 Integration of Soft Computing Techniques

Having now studied all four principal constituent technologies of soft computing across this course — fuzzy logic (Unit II), evolutionary/genetic computing (Units I and IV), and rough sets (this unit), alongside neural networks introduced in Unit I — we conclude by examining how these techniques are combined into hybrid soft-computing systems that exploit each technology's complementary strengths, as Zadeh originally envisioned when he coined the term 'soft computing' (recall Section 1.2 of Unit I).

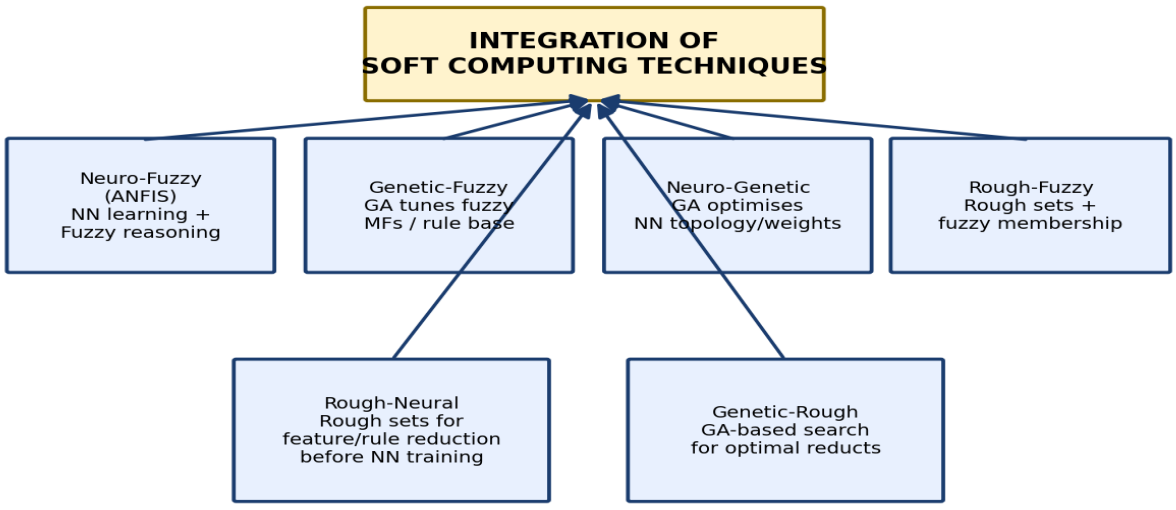

Fig. 5.5 – Principal Hybrid Combinations Formed by Integrating Soft Computing Techniques

5.9.1 Motivation for Hybridisation

Each soft computing technique, used alone, has characteristic strengths and weaknesses: fuzzy logic offers human-interpretable, linguistic reasoning but has limited ability to learn its own rules or membership functions from data; neural networks learn very effectively from data but produce 'black-box' models offering little human interpretability; genetic algorithms search extremely large, complex, poorly-understood spaces effectively but converge relatively slowly for fine numerical tuning; and rough sets excel at attribute reduction and rule induction directly from data but, being based on crisp equivalence classes, are not naturally suited to handling continuous, gradually-varying quantities. Hybridisation aims to combine two (or more) of these technologies so that the strength of one compensates for the weakness of another.

5.9.2 Neuro-Fuzzy Systems (Recap and Extension)

As introduced in Unit I (Section 1.7), neuro-fuzzy systems combine a neural network's ability to learn from data with a fuzzy system's ability to represent that learned knowledge in an interpretable, linguistic, rule-based form. The best-known architecture is the Adaptive Neuro-Fuzzy Inference System (ANFIS), developed by Jang, which represents a Sugeno-type fuzzy inference system (recall Section 2.5.7 of Unit II) as an equivalent multi-layer feed-forward network, in which the membership function parameters and the consequent (linear) coefficients are treated as adjustable network weights and tuned automatically using a hybrid learning rule combining gradient descent (backpropagation) with least-squares estimation. ANFIS thus allows a fuzzy rule base to be learned directly and automatically from training data, rather than requiring an expert to specify it entirely by hand.

5.9.3 Genetic-Fuzzy Systems (Recap and Extension)

Genetic-fuzzy systems use a genetic algorithm (Unit I, Section 1.5, and Unit IV) to automatically optimise the components of a fuzzy inference system: typically either the shape/position of the membership functions (parametric tuning, encoding the MF parameters directly as a real-valued chromosome), or the fuzzy rule base itself (structural learning, encoding the presence/absence and consequent of each candidate rule as part of the chromosome), or both simultaneously in a technique known as the Pittsburgh approach (where an entire rule base is one chromosome) versus the Michigan approach (where each individual chromosome represents just a single rule, and the full rule base emerges from the evolving population as a whole).

5.9.4 Neuro-Genetic Systems (Recap and Extension)

Neuro-genetic systems use a genetic algorithm to design or optimise a neural network: the GA may search over the network's topology (number of layers, number of neurons per layer, connectivity pattern), its initial connection weights (as an alternative or supplement to gradient-based training such as backpropagation), or its learning-rule hyperparameters (learning rate, momentum). This hybrid approach is particularly valuable because neural network architecture design is otherwise a largely trial-and-error process, and a GA's global, population-based search can explore the space of possible architectures far more systematically than manual experimentation.

5.9.5 Rough-Fuzzy Hybrid Systems

Rough-fuzzy hybridisation combines rough sets' principled handling of attribute granularity/reduction with fuzzy logic's principled handling of vagueness in continuous attribute values. A rough-fuzzy hybrid system typically proceeds by first fuzzifying continuous condition attributes into linguistic fuzzy sets (as in Unit II), and then applying rough-set-style indiscernibility, approximation, and reduct/rule-induction machinery (Sections 5.3–5.8 of this unit) directly to these fuzzy-valued attributes, using a suitably generalised, fuzzy version of the indiscernibility relation (in which two objects are considered 'fuzzy-indiscernible' to the degree that their fuzzy attribute values are similar, rather than strictly identical). This allows the attribute-reduction and transparent rule-induction benefits of rough sets to be applied even when the underlying data is continuous and imprecise, rather than being restricted to purely discrete, crisp attribute values.

5.9.6 Rough-Neural and Genetic-Rough Hybrid Systems

Rough sets are also frequently combined with neural networks (rough-neural hybrids) and with genetic algorithms (genetic-rough hybrids). In a typical rough-neural system, rough set attribute reduction (Section 5.6) is first applied as a pre-processing/feature-selection step to remove redundant or irrelevant input attributes, and the resulting reduced attribute set is then used to train a neural network, which benefits from a smaller, more relevant, less noisy input space, typically resulting in faster training and better generalisation. In a genetic-rough hybrid, since the discernibility-matrix-based computation of all possible reducts (Section 5.7) can become computationally very expensive for information systems with a large number of attributes (finding a minimum-size reduct is, in general, an NP-hard combinatorial optimisation problem), a genetic algorithm is instead used as an efficient heuristic search procedure to find

a good, small (though not necessarily provably minimal) reduct directly, without exhaustively constructing and Boolean-reducing the full discernibility function.

5.9.7 Summary Table of Hybrid Soft Computing Architectures

Hybrid Architecture	Combines	Key Benefit
Neuro-Fuzzy (e.g. ANFIS)	Neural network learning + Fuzzy rule-based reasoning	Learns interpretable fuzzy rules automatically from data
Genetic-Fuzzy	Genetic algorithm search + Fuzzy inference system	Automatically tunes membership functions and/or rule base
Neuro-Genetic	Genetic algorithm search + Neural network	Optimises network architecture/weights via global search
Rough-Fuzzy	Rough set approximation + Fuzzy sets	Handles vagueness in continuous attributes with rough-set rule induction
Rough-Neural	Rough set attribute reduction + Neural network	Reduces redundant inputs before training, improving speed/generalisation
Genetic-Rough	Genetic algorithm search + Rough set reduct computation	Efficient heuristic search for near-minimal reducts in large datasets

5.9.8 Case Study — A Rough-Neuro-Fuzzy Hybrid System for Medical Diagnosis

To appreciate how three or more soft computing techniques can be combined within a single application, consider an integrated diagnostic decision-support system built as follows. First, a large historical hospital dataset, containing many candidate symptom and test-result attributes for past patients along with their confirmed diagnoses, is processed using rough set attribute reduction (Section 5.6) to discard redundant or non-informative attributes, retaining only a minimal, high-value reduct of symptoms/tests. Second, since many of the retained attributes (such as body temperature or blood-test values) are continuous and their clinical significance is inherently graded rather than sharply thresholded, these attributes are fuzzified (Unit II, Section 2.2.3) into linguistic fuzzy sets such as 'Normal', 'Elevated', and 'High'. Third, a neuro-fuzzy inference system (ANFIS-style, Section 5.9.2) is then trained on this reduced, fuzzified dataset to learn a compact, interpretable fuzzy rule base linking the retained symptoms to the probability of each candidate diagnosis. The resulting system enjoys the combined benefits of all three constituent technologies: rough sets ensure that only the clinically necessary tests need be performed (reducing cost and patient burden), fuzzy logic ensures that the output is expressed in clinically meaningful, graded linguistic terms rather than an opaque numeric score, and the neural learning component ensures that the fuzzy rule base is continuously improvable as new confirmed cases become available, without requiring a domain expert to manually rewrite the rules from scratch.

5.9.9 Future Scope of Integrated Soft Computing Systems

As data volumes and system complexity continue to grow across virtually every engineering and scientific domain, the integration of soft computing techniques is expected to deepen further. Emerging directions include combining rough-set-based feature/rule reduction with deep neural architectures to make large-scale deep learning models more interpretable and less data-hungry; using multi-objective genetic algorithms (recall Unit I, Section 1.5.20) to jointly optimise the accuracy, interpretability, and computational cost of hybrid fuzzy-rough-neural systems; and deploying lightweight hybrid soft-computing models on edge/IoT devices, where rough-set attribute reduction is particularly valuable for minimising the number of sensors or measurements that must be taken in real time. The foundational ideas studied across this course — fuzzy sets and logic, evolutionary computing, and rough sets — together with their neural-network partner, continue to underpin this rapidly evolving landscape of computational intelligence.

5.10 Applications of Rough Set Theory

Domain	Typical Application
Medical diagnosis	Symptom/test attribute reduction; induction of diagnostic decision rules from patient records
Data mining and knowledge discovery	Feature selection, rule induction, and handling of inconsistent (noisy) data in large databases
Finance	Credit-risk classification, bankruptcy prediction, and stock market trend analysis based on reduced, most-informative financial indicators
Image processing and pattern recognition	Rough-set-based feature selection for reducing high-dimensional image feature vectors prior to classification
Bioinformatics	Gene-expression attribute reduction to identify minimal, most informative sets of genes associated with a disease
Engineering diagnostics	Fault diagnosis in mechanical and electrical systems using rule induction from sensor/attribute decision tables
Text and web mining	Reduction of very high-dimensional term/keyword feature spaces prior to document classification or clustering

As a further illustration, consider the application of rough sets to credit-risk assessment in finance. A bank may record dozens of attributes for each loan applicant — income, employment history, existing debts, repayment history on prior loans, education level, and so on — along with the eventual outcome (defaulted or repaid). Applying rough set reduct computation (Section 5.6) to a historical database of past applicants very often reveals that only a small subset of these attributes (a reduct) is actually necessary to predict the outcome with the same accuracy as the full attribute set, allowing the bank to streamline its loan-application

process by requesting only the genuinely predictive information from future applicants, while the induced decision rules (Section 5.8) provide loan officers with a transparent, auditable justification for each automated credit decision — an increasingly important regulatory requirement in the financial sector.

5.11 Advantages and Limitations of Rough Set Theory

5.11.1 Advantages

- Requires no additional information beyond the data itself — unlike probability theory, it does not require a pre-assumed probability distribution, and unlike fuzzy logic, it does not require an expert-specified membership function.
- Provides an objective, purely data-driven way to identify redundant attributes (via reducts) and to quantify the degree of uncertainty in a classification (via accuracy of approximation).
- Produces transparent, human-interpretable IF-THEN decision rules directly from data, along with an explicit certainty/support measure for each rule.
- Explicitly and precisely identifies the boundary region — the specific objects/cases for which the available attributes are insufficient for a certain classification — rather than merely reporting an aggregate accuracy figure.
- Combines naturally and synergistically with fuzzy logic, neural networks, and genetic algorithms in hybrid soft-computing systems (Section 5.9).

5.11.2 Limitations

- Being based on crisp equivalence classes, classical rough set theory does not directly handle continuous, real-valued attributes without prior discretisation, which can itself introduce information loss or arbitrary threshold choices.
- Finding a reduct of minimum size is, in general, an NP-hard combinatorial problem; exhaustive discernibility-matrix-based computation of all reducts can become computationally expensive for information systems with a large number of attributes.
- The choice of which reduct to use (when multiple reducts exist) is not always unique or obvious, and different reducts may lead to different induced rule sets.
- Purely crisp rough sets do not, by themselves, model the vagueness of natural-language concepts (that is the domain of fuzzy logic) — hybridisation (Section 5.9.5) is required to address both forms of imprecision simultaneously.

5.12 Summary

Rough set theory, introduced by Pawlak, provides a mathematically rigorous framework for reasoning about imprecision arising from insufficient or granular information, complementing fuzzy set theory's treatment of vagueness. An information system (or decision table) represents objects via condition attributes and, optionally, a decision attribute; the indiscernibility relation IND(B), for any attribute subset B, partitions the universe into equivalence classes of objects that are indistinguishable using B. Any target concept X can then be approximated from below (the lower approximation, containing objects certainly in X) and from above (the upper approximation, containing objects possibly in X); when these two approximations differ, X is a rough (rather than crisp/exact) set with respect to B, and the difference constitutes the boundary region, whose size relative to the upper approximation determines the accuracy of approximation. A reduct is a minimal attribute subset preserving the same classificatory power as the full attribute set, and the core (the intersection of all reducts) identifies attributes that are strictly indispensable; both can be computed systematically via the discernibility matrix and discernibility function. Rough sets also support the direct induction of certain and possible IF-THEN decision rules from data, without requiring assumptions about probability distributions or membership functions. Finally, rough sets combine synergistically with fuzzy logic, neural networks, and genetic algorithms — the other soft computing technologies studied throughout this course — to produce hybrid systems (neuro-fuzzy, genetic-fuzzy, neuro-genetic, rough-fuzzy, rough-neural, and genetic-rough) that exploit the complementary strengths of each constituent technology, fully realising Zadeh's original vision of soft computing as a synergistic consortium of computational methodologies.

5.13 Additional Solved Numerical Problems

Problem 1

Given U = {x1,...,x6} and attribute subset B, suppose IND(B) yields the partition U/B = {{x1,x3}, {x2,x5,x6}, {x4}}. Let X = {x1,x3,x4,x5} be a target concept. Compute B_(X), $\bar{B}$(X), BND_B(X), and α_B(X).

Solution: Check each class against X: {x1,x3} ⊆ X (both members in X) → contributes to lower approximation. {x2,x5,x6}: x5 ∈ X but x2, x6 ∉ X → mixed, contributes only to upper approximation. {x4} ⊆ X (its only member is in X) → contributes to lower approximation. Hence B_(X) = {x1,x3,x4}; $\bar{B}$(X) = {x1,x3,x4} ∪ {x2,x5,x6} = {x1,x2,x3,x4,x5,x6} = U; BND_B(X) = $\bar{B}$(X) − B_(X) = {x2,x5,x6}. Accuracy α_B(X) = |B_(X)|/|$\bar{B}$(X)| = 3/6 = 0.5.

Problem 2

For a decision table with condition attributes {p,q,r,s}, the decision-relative discernibility function is found to be f_d = (p∨q) ∧ (p∨r) ∧ (q∨r∨s). Determine the decision-relative reduct(s).

Solution: We must find the minimal disjunctive normal form of f_d. First simplify (p∨q)∧(p∨r) using distribution: (p∨q)∧(p∨r) = p ∨ (q∧r). Substituting: f_d = [p ∨ (q∧r)] ∧ (q∨r∨s). Distributing over the AND: f_d = [p ∧ (q∨r∨s)] ∨ [(q∧r) ∧ (q∨r∨s)]. Since (q∧r) already implies (q∨r∨s) is true, the second term simplifies to just (q∧r). The first term, p∧(q∨r∨s), expands to (p∧q) ∨ (p∧r) ∨ (p∧s). So f_d = (p∧q) ∨ (p∧r) ∨ (p∧s) ∨ (q∧r). Since none of these four terms can be removed without changing the function (none absorbs another, as can be checked), the reducts are {p,q}, {p,r}, {p,s}, and {q,r}. Since 'p' does not appear in the reduct {q,r}, and no single attribute appears in all four reducts, the core is empty: CORE = ∅.

Problem 3

In a decision table, an equivalence class E = {x1,x2,x3,x4,x5} under a condition attribute subset B has decision values {Yes, Yes, No, Yes, No}. Compute the certainty factor of the possible rule 'IF [condition defining E] THEN Decision = Yes', and identify what this class contributes to the positive, negative, and boundary regions.

Solution: Of the 5 objects in E, 3 have decision Yes (x1,x2,x4) and 2 have decision No (x3,x5). The certainty factor of the rule '... THEN Decision = Yes' is therefore 3/5 = 0.6 (and correspondingly, the rule '... THEN Decision = No' has certainty factor 2/5 = 0.4). Since E is not entirely contained in either the Yes class or the No class, it does not contribute to the positive region of either decision value; instead, all 5 objects of E belong to the boundary region BND_B(Decision=Yes) (and, equivalently, BND_B(Decision=No), since for a binary decision the boundary region of X and of its complement U-X coincide).

Problem 4

A decision table has |U| = 20 objects. For a given attribute subset B and target concept X, the lower approximation contains 8 objects and the upper approximation contains 14 objects. Compute the accuracy of approximation α_B(X) and the quality of approximation γ_B(X), and state how many objects lie in the boundary region.

Solution: Accuracy α_B(X) = |B_(X)|/|B̄(X)| = 8/14 ≈ 0.571. Quality γ_B(X) = |B_(X)|/|U| = 8/20 = 0.4. The number of objects in the boundary region is |B̄(X)| − |B_(X)| = 14 − 8 = 6 objects.

5.14 Glossary of Key Terms

Term	Meaning
Information system	A pair (U,A) of a universe of objects and a set of attributes describing them
Decision table	An information system with one attribute distinguished as the decision (class) attribute
Indiscernibility relation IND(B)	An equivalence relation grouping objects that have identical values for every attribute in B
Equivalence class	A maximal set of objects that are pairwise indiscernible under a given attribute subset
Lower approximation	The set of objects certainly belonging to a target concept X, given attribute subset B
Upper approximation	The set of objects possibly belonging to a target concept X, given attribute subset B
Boundary region	The set of objects that cannot be classified with certainty as in or out of X
Rough set	A set with a non-empty boundary region — i.e. one that cannot be exactly defined using the given attributes
Positive region	Objects certainly belonging to the target concept (equal to the lower approximation)
Negative region	Objects certainly not belonging to the target concept
Accuracy of approximation	The ratio of the size of the lower approximation to the size of the upper approximation
Quality of approximation	The fraction of all objects in the universe correctly, certainly classified using a given attribute subset
Reduct	A minimal attribute subset preserving the classificatory power of the full attribute set
Core	The intersection of all reducts; attributes that are indispensable
Discernibility matrix	A matrix recording, for every pair of objects, the attributes that distinguish them
Discernibility function	A Boolean function, derived from the discernibility matrix, whose minimal terms correspond to reducts
Certain rule	A decision rule derived from the positive region, holding with full (100%) certainty
Possible rule	A decision rule derived from the boundary region, holding with a certainty factor less than 1
Neuro-fuzzy system	A hybrid combining neural-network learning with fuzzy rule-based reasoning (e.g. ANFIS)
Rough-fuzzy system	A hybrid combining rough-set attribute reduction/rule induction with fuzzy handling of continuous vagueness

5.15 Review Questions

158. Define an information system and a decision table, with a suitable example.

159. Define the indiscernibility relation IND(B). Prove that it is an equivalence relation.

160. Explain, with a worked example, how the lower and upper approximations of a target concept X are computed with respect to an attribute subset B.

161. Define the boundary region of a set X. When is X said to be a rough set with respect to B?

162. Define the positive, negative, and boundary regions of a decision table. Show that they partition the universe U.

163. Define accuracy of approximation and quality of approximation, and compute both for a given worked example.

164. Define reduct and core. Explain why an information system may have more than one reduct.

165. Explain the construction of a discernibility matrix and discernibility function, and describe how reducts are derived from it.

166. Differentiate between a classical (full) reduct and a decision-relative reduct, with a suitable example showing that they can differ.

167. Explain, with an example, how certain and possible decision rules are induced from a rough-set analysis of a decision table.

168. Describe the LEM2 algorithm for rule induction.

169. Explain any three hybrid soft-computing architectures that integrate rough sets with fuzzy logic, neural networks, or genetic algorithms.

170. Describe a real-world case study illustrating the integration of rough sets, fuzzy logic, and neural networks in a single application.

171. Discuss the applications, advantages, and limitations of rough set theory.

172. Given a decision table with 6 objects and a partition U/B = {{x1,x2},{x3},{x4,x5,x6}}, and target concept X = {x1,x2,x4,x5}, compute the lower approximation, upper approximation, boundary region, and accuracy of approximation.

5.16 Multiple Choice Questions

173. Rough set theory was introduced by: (a) Lotfi Zadeh (b) Zdzisław Pawlak (c) John Holland (d) Ebrahim Mamdani — Ans: (b)

174. Rough set theory primarily addresses imprecision arising from: (a) Vagueness of natural-language concepts (b) Insufficient/granular information (c) Randomness (d) Neural network weight initialisation — Ans: (b)

175. The indiscernibility relation IND(B) is: (a) Reflexive only (b) An equivalence relation (c) A partial order (d) A fuzzy relation — Ans: (b)

176. The lower approximation of X consists of objects that: (a) Possibly belong to X (b) Certainly belong to X (c) Certainly do not belong to X (d) Cannot be classified — Ans: (b)

177. If $B\_(X) = \bar{B}(X)$, then X is: (a) A rough set (b) Undefined (c) A crisp/B-definable set (d) Empty — Ans: (c)

178. The accuracy of approximation $\alpha\_B(X)$ is computed as: (a) $|\bar{B}(X)|/|B\_(X)|$ (b) $|B\_(X)|/|\bar{B}(X)|$ (c) $|B\_(X)|/|U|$ (d) $|BND\_B(X)|/|U|$ — Ans: (b)

179. A reduct is: (a) Any subset of attributes (b) A minimal attribute subset preserving classificatory power (c) The full attribute set (d) The decision attribute — Ans: (b)

180. The core of an information system is defined as: (a) The union of all reducts (b) The intersection of all reducts (c) The largest reduct (d) The smallest attribute value — Ans: (b)

181. The discernibility matrix entry c_ij contains: (a) Attributes where x_i and x_j agree (b) Attributes where x_i and x_j differ (c) The decision value of x_i (d) The equivalence class of x_j — Ans: (b)

182. A certain decision rule is derived from: (a) The boundary region (b) The negative region only (c) The positive region (lower approximation) (d) The core — Ans: (c)

183. ANFIS is an example of which hybrid architecture? (a) Rough-Fuzzy (b) Neuro-Fuzzy (c) Genetic-Rough (d) Rough-Neural — Ans: (b)

184. Rough-set-based attribute reduction used as pre-processing before neural network training is an example of: (a) Genetic-Fuzzy hybridisation (b) Rough-Neural hybridisation (c) Neuro-Genetic hybridisation (d) Fuzzy-Genetic hybridisation — Ans: (b)

185. Finding a minimum-size reduct is, in general: (a) A polynomial-time problem (b) An NP-hard problem (c) Always trivial (d) Impossible — Ans: (b)

186. The Michigan approach in genetic-fuzzy systems encodes: (a) An entire rule base as one chromosome (b) A single rule per chromosome (c) Only membership function parameters (d) Only the decision attribute — Ans: (b)