Soft computing

3rd class\ 2nd course\ Computer Science Department\ College of Science for Women

Dr. Noor A. Ibraheem

Soft computing

3rd class\ 2nd course\ Computer Science Department\ College of Science for Women

Dr. Noor A. Ibraheem

Introduction to Soft Computing

What is intelligence?

Real intelligence is what determines the normal thought process of a human. Artificial intelligence is a property of machines which gives it ability to mimic the human thought process. The intelligent machines are developed based on the intelligence of a subject, of a designer, of a person, of a human being.

What is AI?

Artificial Intelligence is concerned with the design of intelligence in an artificial device.

There are two ideas in the definition.

1. Intelligence
2. Artificial device

Typical AI problems

While studying the typical range of tasks that we might expect an "intelligent entity" to perform, we need to consider both "common-place" tasks as well as expert tasks. Examples of common-place tasks include

– Recognizing people, objects.

– Communicating (through natural language).

– Navigating around obstacles on the streets

Intelligent behavior

This discussion brings us back to the question of what constitutes intelligent behaviour. Some of these tasks and applications are:

1. Perception involving image recognition and computer vision
2. Reasoning
3. Learning

4. Understanding language involving natural language processing, speech processing

5. Solving problems

6. Robotics

What does Soft Computing mean?

1.1 Definition of Soft Computing

Prior to 1994 when Zadeh (Zadeh 1994) first defined "soft computing", the currently-handled concepts used to be referred to in an isolated way, whereby each was spoken of individually with an indication of the use of fuzzy methodologies. Although the idea of establishing the area of soft computing dates back to 1990 (Zadeh 2001), it was in (Zadeh 1994) that Zadeh established the definition of soft computing in the following terms:

"Basically, soft computing is not a homogeneous body of concepts and techniques. Rather, it is a partnership of distinct methods that in one way or another conform to its guiding principle. At this juncture, the dominant aim of soft computing is to exploit the tolerance for imprecision and uncertainty to achieve tractability, robustness and low solutions cost. The principal constituents of soft computing are fuzzy logic, neurocomputing, and probabilistic reasoning, with the latter subsuming genetic algorithms, belief networks, chaotic systems, and parts of learning theory. In the partnership of fuzzy logic, neurocomputing, and probabilistic reasoning, fuzzy logic is mainly concerned with imprecision and approximate reasoning; neurocomputing with learning and curve-fitting; and probabilistic reasoning with uncertainty and belief propagation".

Soft computing could therefore be seen as a series of techniques and methods so that real practical situations could be dealt with in the same way as humans deal with them, i.e. on the basis of intelligence, common sense, consideration of analogies, approaches, etc. In this sense, soft computing is a family of problem-resolution methods headed by approximate reasoning and functional and optimisation approximation methods, including search methods. Soft computing is therefore the theoretical basis for the area of intelligent systems and it is evident that the difference between the area of artificial intelligence and that of intelligent systems is that the first is based on hard computing and the second on soft computing. Soft Computing is still growing and developing.

From this other viewpoint on a second level, soft computing can be then expanded into other components which contribute to a definition by extension, such as the one first given. From the beginning (Bonissone 2002), the components considered to be the most important in this second level are probabilistic reasoning, fuzzy logic and fuzzy sets, neural networks, and genetic algorithms, which because of their interdisciplinary, applications and results immediately stood out over other methodologies such as the previously mentioned chaos theory, evidence theory, etc. The popularity of genetic algorithms, together with their proven efficiency in a wide variety of areas and applications, their attempt to imitate natural creatures (e.g. plants, animals, humans) which are clearly soft (i.e. flexible, adaptable, creative, intelligent, etc.), and especially the extensions and different versions, transform this fourth second-level ingredient into the well-known evolutionary algorithms which consequently comprise the fourth fundamental component of soft computing, as shown in the following diagram, see Figure.

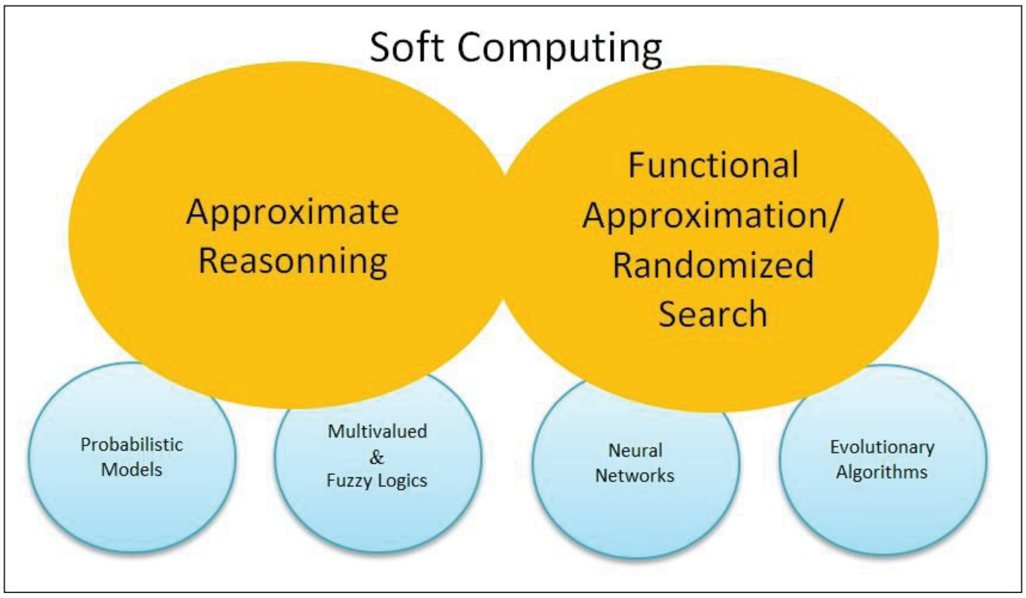


Figure : What does Soft Computing mean?

1.2 Soft Computing Goals

Soft Computing is a new multidisciplinary field, to construct new generation of Artificial Intelligence, known as Computational Intelligence. The main goal of Soft Computing is to develop intelligent machines to provide solutions to real world problems, which are not modeled, or too difficult to model mathematically. Its aim is to exploit the tolerance for

- Approximation: The model features are similar to the real ones, but not the same.
- Uncertainty: here we are not sure that the features of the model are the same as that of the entity (belief).
- Imprecision and Partial Truth: in order to achieve close resemblance with human like decision making. Here the model features are not the same as that of the real ones, but close to them.

1.3 Importance of Soft Computing

The aim of Soft Computing is to exploit tolerance for imprecision, uncertainty, approximate reasoning, and partial truth in order to achieve close resemblance with human-like decision making. Soft Computing is a new multidisciplinary field, to construct a new generation of Artificial Intelligence, known as *Computational Intelligence.*

As stated in (Verdegay 2003), since the fuzzy boom of the 1990s, methodologies based on fuzzy sets (i.e. soft computing) have become a permanent part of all areas of research, development and innovation, and their application has been extended to all areas of our daily life: health, banking, home, and are also the object of study on different educational levels. Similarly, there is no doubt that thanks to the technological potential that we currently have, computers can handle problems of tremendous complexity (both in comprehension and dimension) in a wide variety of new fields.

As we mentioned above, since the 1990s, evolutionary algorithms have proved to be extremely valuable for finding good solutions to specific problems in these fields, and thanks to their scientific attractiveness, the diversity of their applications and the considerable efficiency of their solutions in intelligent systems, they have been incorporated into the second level of soft computing components.

Evolutionary algorithms, however, are merely another class of heuristics, or meta-heuristics, in the same way as Tabu Search, Simulated Annealing, Hill Climbing, Variable Neighborhood Search, Estimation Distribution Algorithms, Scatter Search, Reactive Search and very many others are. Generally speaking, all these heuristic algorithms (meta-heuristics) usually provide solutions which are not ideal, but which largely satisfy the decision-maker or the user. When these act on the basis that satisfaction is better than optimization, they perfectly illustrate Zadeh's famous sentence (Zadeh 1994):

"…in contrast to traditional hard computing, soft computing exploits the tolerance for imprecision, uncertainty, and partial truth to achieve tractability, robustness, low solution-cost, and better rapport with reality".

- Linear and non- linear terms

- In simple terms, a **nonlinear system** is one in which the output of the system is not proportional to the input. This is, of course, in contrast to linear systems.
- **A linear process** is one in which something changes or progresses straight from one stage to another, and has a starting point and an ending point.

1.4 Properties of Soft Computing methods

These methods have in common: They

1. Are nonlinear.
2. Have the ability to deal with non-linearity.
3. Follow more human like reasoning paths than classical methods.
4. Utilize self-learning.
5. Utilize yet-to-be proven theorems.
6. Are robust in the presence of noise or errors.

1.5 Why using Soft Computing approach?

Mathematical model & analysis can be done for relatively simple systems. More complex systems arising in biology, medicine and management systems remain intractable to conventional mathematical and analytical methods. Soft computing deals with imprecision, uncertainty, partial truth and approximation to achieve tractability, robustness and low solution cost. It extends its application to various disciplines of Engineering and science. Typically human can:

1. Take decisions
2. Inference from previous situations experienced

3. Expertise in an area
4. Adapt to changing environment
5. Learn to do better
6. Social behavior of collective intelligence

Intelligent control strategies have emerged from the above mentioned characteristics of human/ animals.

1.6 Characteristics Soft Computing:

1. Human Expertise
2. Biologically inspired computing models
3. New Optimization Techniques
4. Numerical Computation
5. New Application domains
6. Model-free learning
7. Intensive computation
8. Fault tolerance
9. Goal driven characteristics
10. Real world applications

1.7 Soft computing techniques / tools / methods

Intelligent Control Strategies (Components of Soft Computing): The popular soft computing components in designing intelligent control theory are:

1. Fuzzy Logic
2. Neural Networks
3. Evolutionary Algorithms

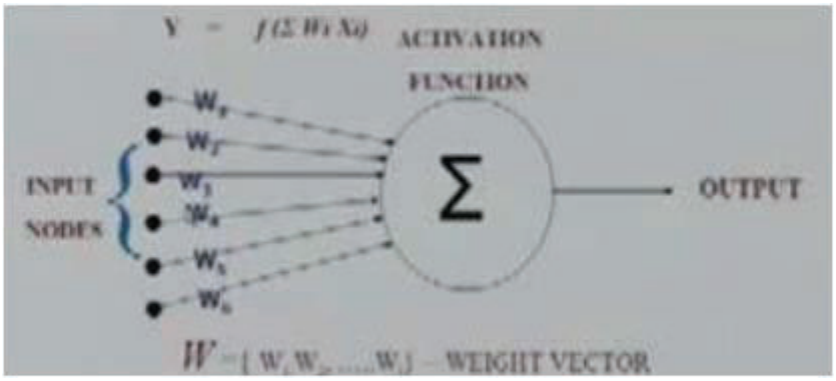
1. Fuzzy logic:

Most of the time, people are fascinated about fuzzy logic controller. At some point of time in Japan, the scientists designed fuzzy logic controller even for household appliances like a room heater or a washing machine. Its popularity is such that it has been applied to various engineering products.

2. Neural networks:

Neural networks are basically inspired by various way of observing the biological organism. Most of the time, it is motivated from human way of learning. It is a learning theory. This is an artificial network that learns from example and because it is distributed in nature, fault tolerant, parallel processing of data and distributed structure.

The basic elements of artificial Neural Network are: input nodes, weights, activation function and output node. Inputs are associated with synaptic weights. They are all summed and passed through an activation function giving output y. In a way, output is summation of the signal multiplied with synaptic weight over many input channels.



Basic elements of an artificial neuron

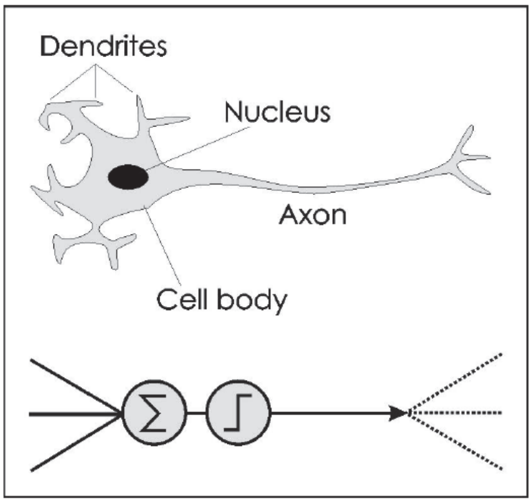


Figure: Analogy of biological neuron and artificial neuron

Above figure shows a biological neuron on top. Through axon this neuron actuates the signal and this signal is sent out through synapses to various neurons. Similarly shown a classical artificial neuron (bottom).This is a computational unit. There are many inputs reaching this. The input excites this neuron. Similarly, there are many inputs that excite this computational unit and the output again excites many other units like here. Like that taking certain concepts in actual neural network, we develop these artificial computing models having similar structure.

Neural networks are analogous to adaptive control concepts that we have in control theory and one of the most important aspects of intelligent control is to learn the control parameters, to learn the system model. Some of the learning methodologies we will be learning here is the error-back propagation algorithm, real-time learning algorithm for recurrent network, Kohonen‟s self-organizing feature map & Hopfield network.

Features of Artificial Neural Network (ANN) models:

1. Parallel Distributed information processing
2. High degree of connectivity between basic units
3. Connections are modifiable based on experience
4. Learning is a continuous unsupervised process
5. Learns based on local information
6. Performance degrades with less units

3. Evolutionary algorithms:

These are mostly derivative free optimization algorithms that perform random search in a systematic manner to optimize the solution to a hard problem. In this course Genetic Algorithm being the first such algorithm developed in 1970‟s will be discussed in detail. The other algorithms are swarm based that mimic behavior of organisms, or any systematic process.

Hard Computing and Soft Computing

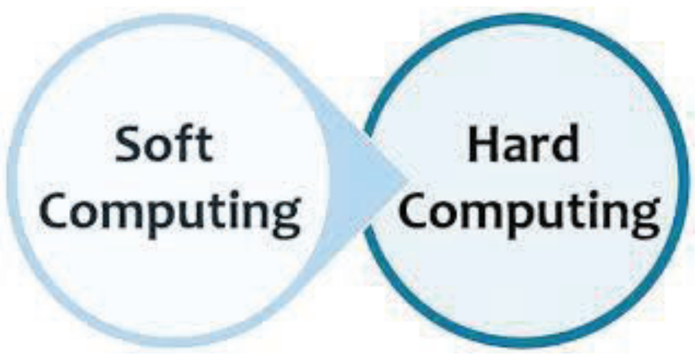


Figure: Hard computing

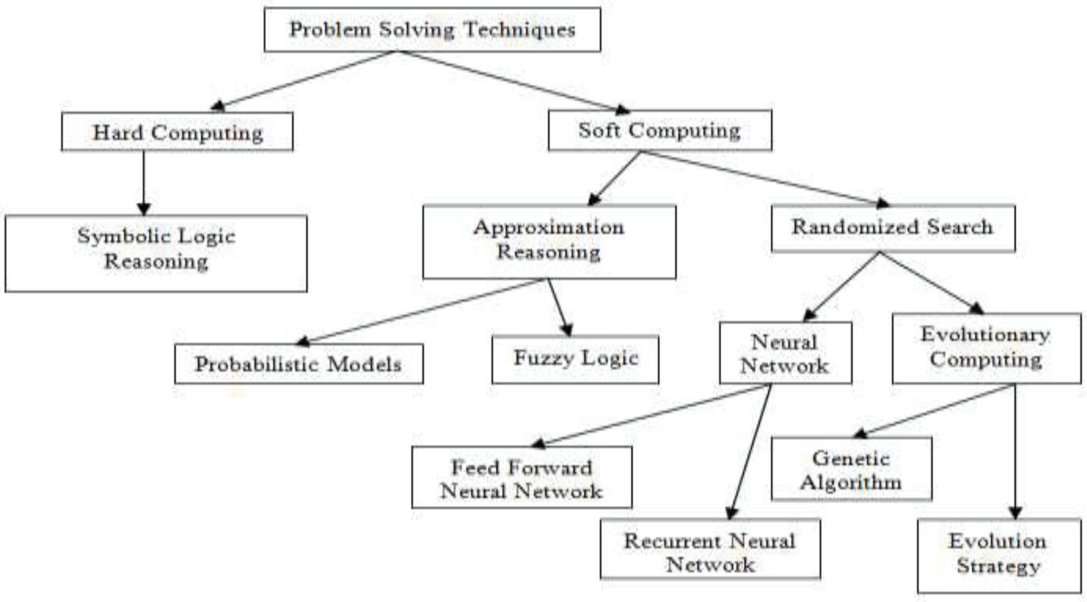
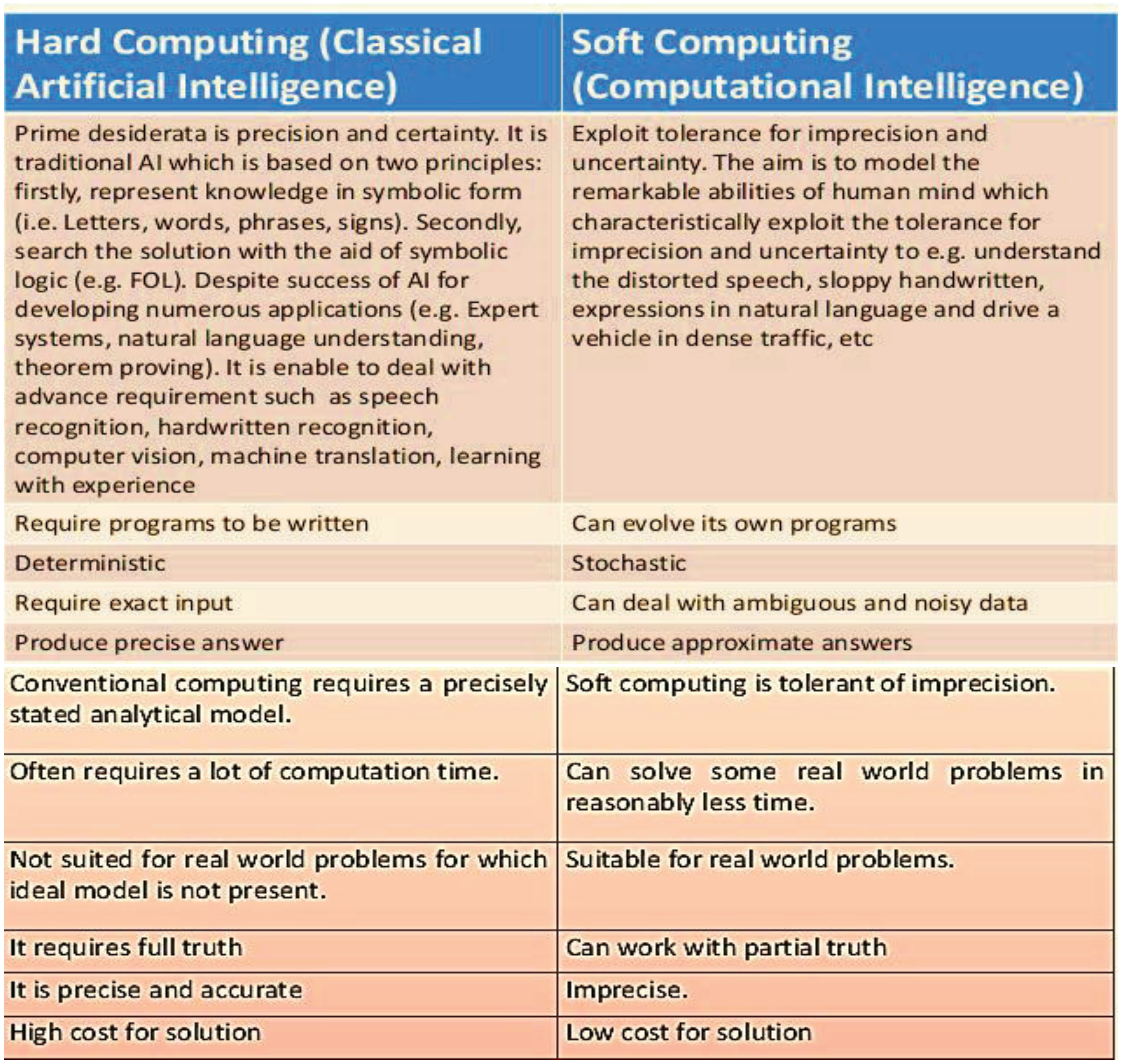
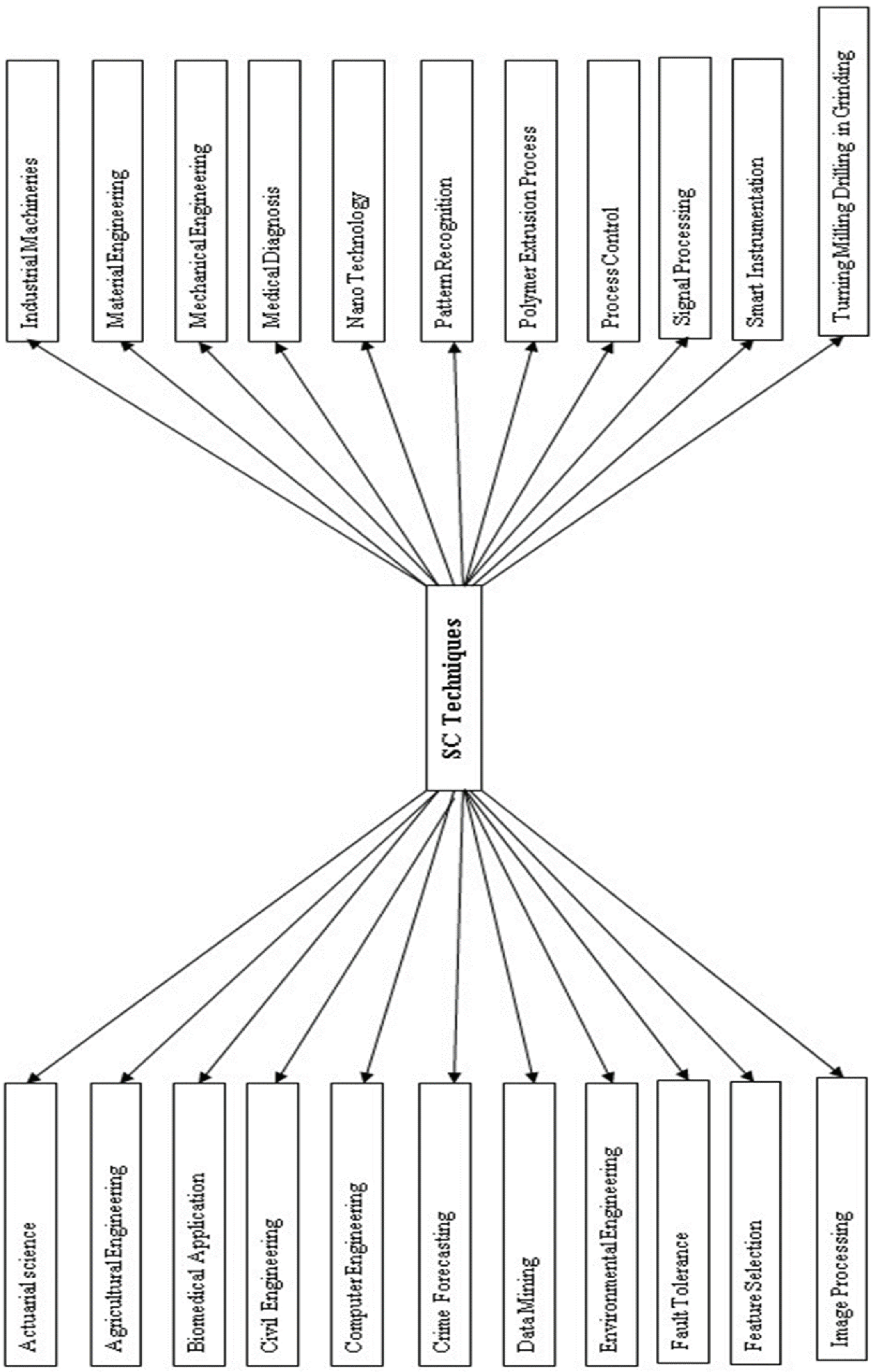


Figure: Overview of Problem Solving Techniques.



Hard Computing (Classical Artificial Intelligence)	Soft Computing (Computational Intelligence)
Prime desiderata is precision and certainty. It is traditional AI which is based on two principles: firstly, represent knowledge in symbolic form (i.e. Letters, words, phrases, signs). Secondly, search the solution with the aid of symbolic logic (e.g. FOL). Despite success of AI for developing numerous applications (e.g. Expert systems, natural language understanding, theorem proving). It is enable to deal with advance requirement such as speech recognition, hardwritten recognition, computer vision, machine translation, learning with experience	Exploit tolerance for imprecision and uncertainty. The aim is to model the remarkable abilities of human mind which characteristically exploit the tolerance for imprecision and uncertainty to e.g. understand the distorted speech, sloppy handwritten, expressions in natural language and drive a vehicle in dense traffic, etc
Require programs to be written	Can evolve its own programs
Deterministic	Stochastic
Require exact input	Can deal with ambiguous and noisy data
Produce precise answer	Produce approximate answers
Conventional computing requires a precisely stated analytical model.	Soft computing is tolerant of imprecision.
Often requires a lot of computation time.	Can solve some real world problems in reasonably less time.
Not suited for real world problems for which ideal model is not present.	Suitable for real world problems.
It requires full truth	Can work with partial truth
It is precise and accurate	Imprecise.
High cost for solution	Low cost for solution

1.8 Application areas of soft computing


SC Techniques
Industrial Machineries
Material Engineering
Mechanical Engineering
Medical Diagnosis
Nano Technology
Pattern Recognition
Polymer Extrusion Process
Process Control
Signal Processing
Smart Instrumentation
Turning Milling Drilling in Grinding
Actuarial science
Agricultural Engineering
Biomedical Application
Civil Engineering
Computer Engineering
Crime Forecasting
Data Mining
Environmental Engineering
Fault Tolerance
Feature Selection
Image Processing

Soft computing

3rd class\ 2nd course\ 2nd lecture
Computer Science Department\ College of Science for Women

Dr. Noor A. Ibraheem

Forward Chaining and backward chaining in AI

What is an Inference Engine?

An Inference Engine is a tool of Artificial Intelligence that is used as a component of the system to deduce new information from a knowledge base using logical rules and reasoning. The first-ever Inference Engines were a part of expert systems in AI. As previously stated, Inference Engines predict outcomes with the already existing pool of data, comprehensively analyzing it and using logical reasoning to predict the outcomes.

The inference engine is applies logical rules to the knowledge base to infer new information from known facts. The first inference engine was part of the expert system.

This same process would be repeated as new facts would be discovered and this would make the inference engine trigger additional rules for its findings. After some runs of the inference engine, it was noticed that Inference Engines works in one of the two ways, either based on goals or based on facts, which later came to be known as **forwarding chaining** and **backward chaining**.

Examples regarding Inference Rules

Let’s take a look at some simple examples to help you differentiate between both sets of inference rules.

Inference Rules

- **Deductive inference rule:**

Forward Chaining: **Conclude from "A" and "A implies B"** to "**B**".

A

A -> B

B

Example:

It is raining.

If it is raining, the street is wet.

The street is wet.

- **Abductive inference rule:**

Backward Chaining: **Conclude from "B" and "A implies B**" to "**A**".

B

A -> B

A

Example:

The street is wet.

If it is raining, the street is wet.

It is raining.

Horn Clause and Definite clause:

Horn clause and definite clause are the forms of sentences, which enables knowledge base to use a more restricted and efficient inference algorithm. Logical inference algorithms use forward and backward chaining approaches, which require knowledge base in the form of the **first-order definite clause**.

Definite clause: A clause which is a disjunction of literals with **exactly one positive literal** is known as a definite clause or strict horn clause.

Horn clause: A clause which is a disjunction of literals with **at most one positive literal** is known as horn clause. Hence all the definite clauses are horn clauses.

Example: (¬ p V ¬ q V k). It has only one positive literal **k.**

A. Forward Chaining

Forward chaining is also known as a forward deduction or forward reasoning method when using an inference engine. Forward chaining is a form of reasoning which start with atomic sentences in the knowledge base and applies inference rules (Modus Ponens) in the forward direction to extract more data until a goal is reached.

Forward Chaining is one of the two main methods of inference engine which uses the logical process of inferring unknown truths to find a solution from the known set of data by using determined conditions and rules.

The Forward-chaining algorithm starts from known facts, triggers all rules whose premises are satisfied, and add their conclusion to the known facts. This process repeats until the problem is solved.

As a data-driven as well as bottom-up logic approach, forward chaining starts from known facts and conditions, then progresses towards logical conclusion using if-then statements. Then these conditions and rules are applied to the problem until no further applicable situations are left or the limit has been reached. Forward Chaining searches for any solutions and can come up with an infinite number of possible conclusions.

Forward Chaining in AI

The Forward-thinking approach is used in AI to help an AI agent solve logical problems by inspecting the data from the previous learnings and then coming to a conclusion full of solutions. That's not all, Forward Chaining might as well be used to explore the available information or answer a question or solve a problem. Forward chaining is extensively used to break down a long and complex logical approach by attaching each step once the previous one is completed. This way, it goes from beginning to the end with relative ease.

Steps for working of Forwarding Chaining

1. Step 1: We start from the already stated facts, and then, we'll subsequently choose the facts that do not have any implications at all.
2. Step 2: Now, we will state those facts that can be inferred from available facts with satisfied premises.
3. Step 3: In step 3 we can check the given statement that needs to be checked and check whether it is satisfied with the substitution which infers all the previously stated facts. Thus we reach our goal.

Properties of Forward-Chaining:

- It is a down-up approach, as it moves from bottom to top.
- It is a process of making a conclusion based on known facts or data, by starting from the initial state and reaches the goal state.
- Forward-chaining approach is also called as data-driven as we reach to the goal using available data.
- Forward -chaining approach is commonly used in the expert system, such as CLIPS, business, and production rule systems.

B. Backward Chaining:

Backward-chaining is also known as a backward deduction or backward reasoning method when using an inference engine. A backward chaining algorithm is a form of reasoning, which starts with the goal and works backward, chaining through rules to find known facts that support the goal.

Backward Chaining in AI

The Backward Chaining approach is used in AI to find the conditions and rules because of which a particular logical result or conclusion was reached. Real-life applications of Backward Chaining include use to find information regarding conclusions and solutions in reverse engineering practices as well as game theory applications.

Backward Chaining is a logical process of determining unknown facts from known solutions by moving backwards from known solutions to determine the initial conditions and rules.

This means that Backward Chaining is a top-down reasoning approach that starts from conclusions and then goes back towards the conditions it was inferred from using the depth-first approach. In short, this means that Backward Chaining traces back through the code and applies logic to determine which of the following actions would have caused the result.

Steps of working for Backward Chaining

1. Step 1. In the first step, we'll take the Goal Fact and from the goal fact, we'll derive other facts that we'll prove true.
2. Step 2: We'll derive other facts from goal facts that satisfy the rules
3. Step 3: At step-3, we will extract further fact which infers from facts inferred in step 2.
4. Step 4: We'll repeat the same until we get to a certain fact that satisfies the conditions.

Properties of backward chaining:

- It is known as a top-down approach.
- Backward-chaining is based on modus ponens inference rule.

- In backward chaining, the goal is broken into sub-goal or sub-goals to prove the facts true.
- It is called a goal-driven approach, as a list of goals decides which rules are selected and used.
- Backward -chaining algorithm is used in game theory, automated theorem proving tools, inference engines, proof assistants, and various AI applications.
- The backward-chaining method mostly used a **depth-first search** strategy for proof.

Difference between Forward Chaining and Backward Chaining

S No	Forward Chaining	Backward Chaining
1.	It starts from known facts extract more data unit it reaches to the goal using inference rule	It starts from the goal and works backward through inference rules to find the required facts that support the goal.
2.	Bottom-up Approach	Top-Down Approach
3.	Known as Data-driven approach as we use given data to reach the goals	Known as goal-driven approach because we use the goal given to reach the facts that support the goals
4	Applies a breadth-first search strategy	Applies a depth-first search strategy
5	Tests for all the available rules	Only tests for certain given and selected rules
6	Suitable for planning, monitoring, control, and interpretation application.	Suitable for diagnostic, prescription, and debugging application.

7.	Can generate infinite number of possible conclusions	Can generate a finite number of possible concluding facts and conditions
8.	Operates in Forward Direction	Operates in Backward Direction
9	Forward Chaining is aimed for any conclusion.	Backward chaining is aimed for only the required data.

Example of a Declarative Knowledge Base

Father(peter,mary)
Father(peter,john)
Mother(mary,mark)
Mother(jane,mary)

Father(X,Y) AND Father(Y,Z) $\rightarrow$ Grandfather(X,Z)
Father(X,Y) AND Mother(Y,Z) $\rightarrow$ Grandfather(X,Z)
Mother(X,Y) AND Father(Y,Z) $\rightarrow$ Grandmother(X,Z)
Mother(X,Y) AND Mother(Y,Z) $\rightarrow$ Grandmother(X,Z)
Father(X,Y) AND Father(X,Z) $\rightarrow$ Sibling(Y,Z)
Mother(X,Y) AND Mother(X,Z) $\rightarrow$ Sibling(Y,Z)

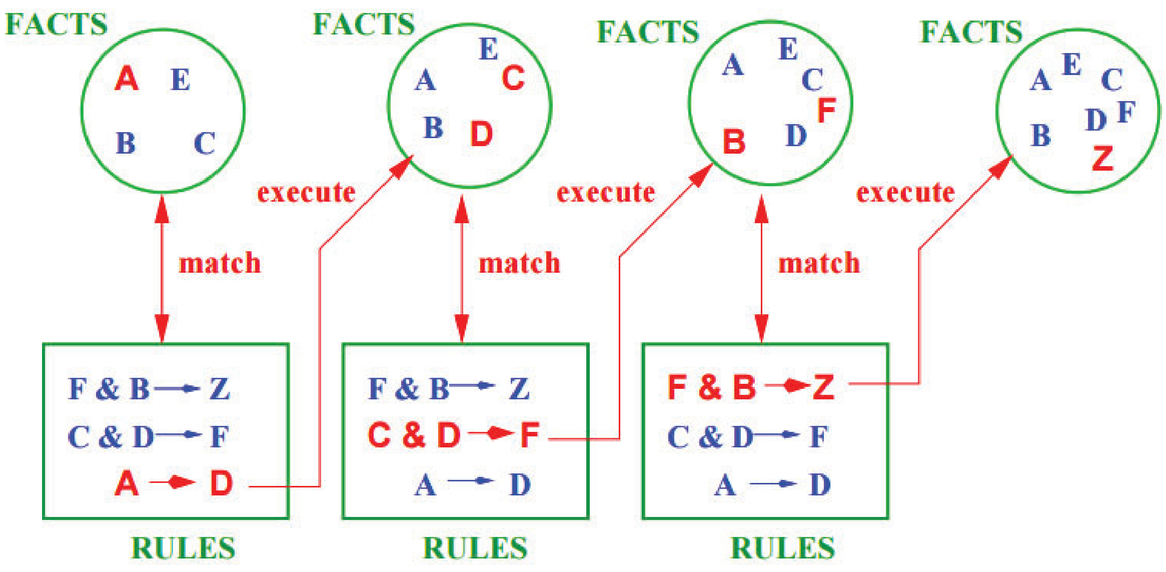
The rules can be used to derive all grandparent and sibling relationships so it is (forward chaining)

Question: what about (backward chaining)?

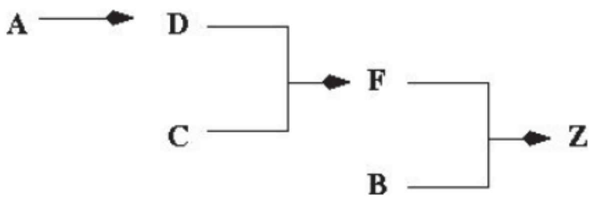
Example Forward Chaining

Goal state: Z

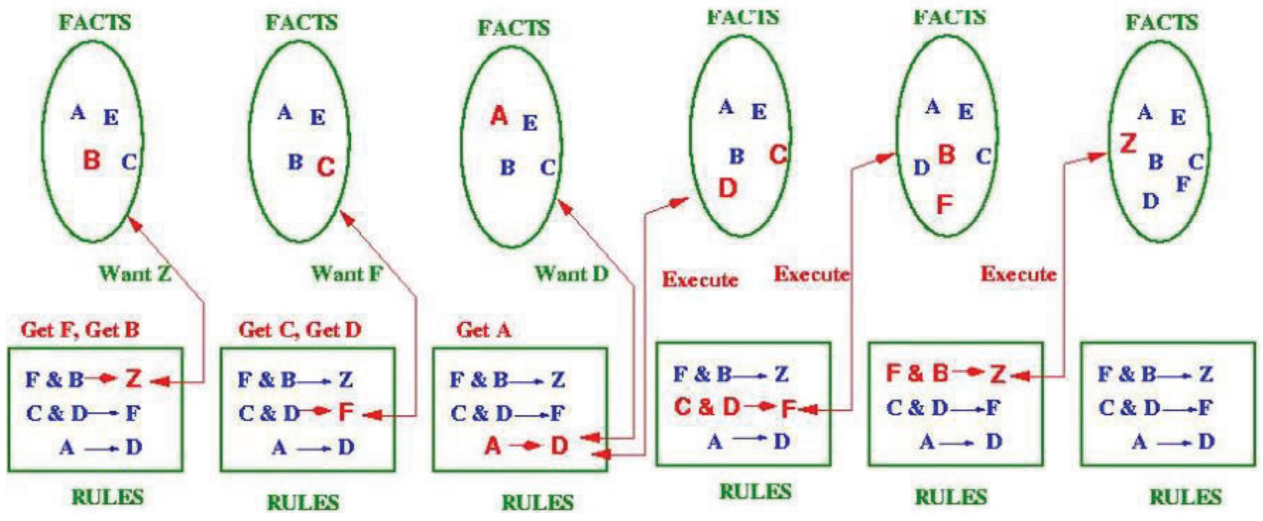
Termination condition: stop if Z is derived or no further rule can be applied



The Forward Inference Chain In this example there are no more rules, so we can draw the inference chain:



Example Backward Chaining



The Backward Inference Chain The first part of the chain works back from the goal until only the initial facts are required, at which point we know how to traverse the chain to achieve the goal state.

$$\begin{tikzcd}
 & B & & \\
Z \arrow[ur] \arrow[dr] & & D \arrow[r] & A \\
 & F \arrow[ur] \arrow[dr] & & \\
 & & C &
\end{tikzcd}$$

Soft computing

3rd class\ 2nd course\ 3rd lecture
Computer Science Department\ College of Science for Women

Dr. Noor A. Ibraheem

Evolutionary Computation

Evolutionary Computation

In computer science, evolutionary computation is a family of algorithms for global optimization inspired by biological evolution, and the subfield of artificial intelligence and soft computing studying these algorithms. In evolutionary computation, the process of natural evolution is used as a role model for a strategy for finding optimal or near optimal solutions for a given problem.

Benefits using EC techniques mostly come from flexibility gains and their fitness to the objective target in combination with a robust behavior. Now days, EC is consider as an adaptable concept for problems solution, especially complex optimization problems. This vision is the alternative to some old descriptions that shows EC as a collection of similar algorithms ready to be used in any problem.

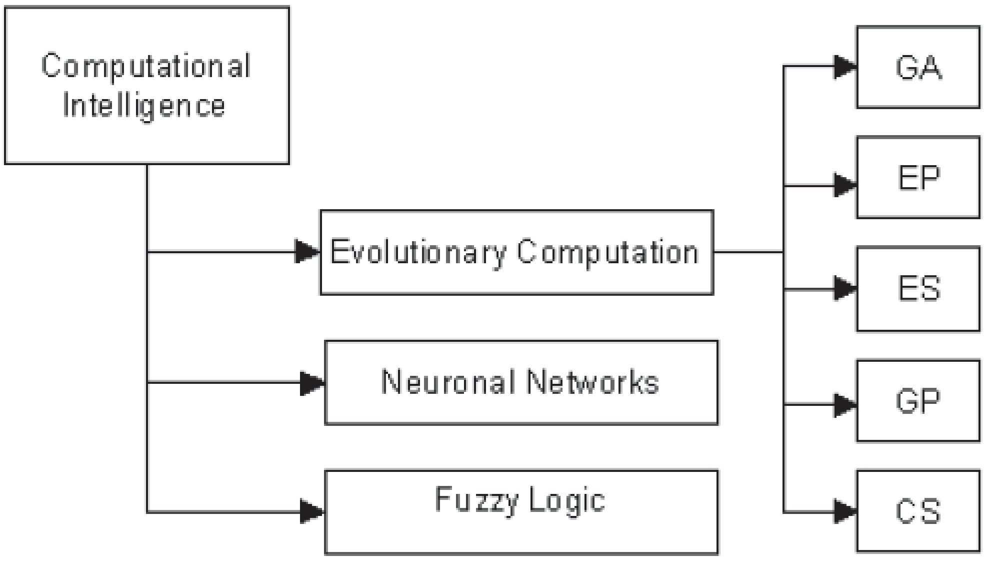


Figure 1. Different families of evolutionary algorithms.

Optimization problem

Optimization is a process that finds a best, or optimal, solution for a problem.

The Optimization problems are centered around three factors :

A. An objective function : which is to be minimized or maximized;

Examples:

1. In manufacturing, we want to maximize the profit or minimize the cost.

2. In designing an automobile panel, we want to maximize the strength.

B. A set of unknowns or variables: that affect the objective function,

Examples:

1. In manufacturing, the variables are amount of resources used or the time spent.

2. In panel design problem, the variables are shape and dimensions of the panel.

C. A set of constraints: that allow the unknowns to take on certain values but exclude others;

Examples:

1. In manufacturing, one constrain is, that all "time" variables to be non-negative.

2 In the panel design, we want to limit the weight and put constrain on its shape.

An optimization problem is defined as : Finding values of the **variables** that minimize or maximize the **objective function** while satisfying the **constraints**.

Optimization Methods

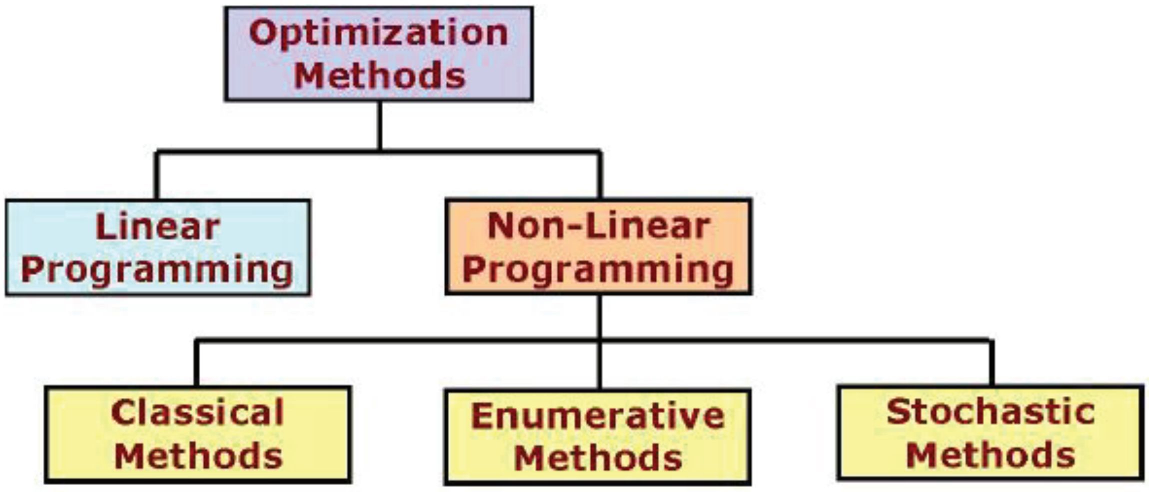
Many optimization methods exist and categorized as shown below. The suitability of a method depends on one or more problem a characteristics to be optimized to meet one or more objectives like:

— low cost,

— high performance,

— low loss

These characteristics are not necessarily obtainable, and requires knowledge about the problem.



Optimization Methods

Each of these methods are briefly discussed indicating the nature of the problem they are more applicable.

Linear Programming

Intends to obtain the optimal solution to problems that are perfectly represented by a set of linear equations; thus require a priori knowledge of the problem. Here the

— The functions to be minimized or maximized, is called **objective functions**,

— The set of linear equations are called **restrictions**.

— The **optimal solution**, is the one that minimizes (or maximizes) the objective function.

Example: "Traveling salesman", seeking a minimal traveling distance.

Non- Linear Programming

Intended for problems described by non-linear equations. The methods are divided in three large groups:

Classical, Enumerative and Stochastic.

Classical search uses deterministic approach to find best solution. These methods requires knowledge of gradients or higher order derivatives. In many practical problems, some desired information are not available, means deterministic algorithms are inappropriate.

The techniques are subdivide into:

— **Direct methods**, e.g. Newton or Fibonacci

— **Indirect methods**.

Enumerative search goes through every point (one point at a time) related to the function's domain space. At each point, all possible solutions are generated and tested to find optimum solution. It is easy to implement but usually require significant computation. In the field of artificial intelligence, the enumerative methods are subdivided into two categories:

— **Uninformed methods**, e.g. Mini-Max algorithm

— Informed methods, e.g. Alpha-Beta and A* ,

Stochastic search deliberately introduces randomness into the search process. The injected randomness may provide the necessary impetus to move away from a local solution when searching for a global optimum. e.g., a gradient vector criterion for “smoothing” problems. Stochastic methods offer robustness quality to optimization process. Among the stochastic techniques, the most widely used are:

— Evolutionary Strategies (ES),

— Genetic Algorithms (GA), and

— Simulated Annealing (SA).

The ES and GA emulate nature's evolutionary behavior, while SA is based on the physical process of annealing a material.

Search Optimization

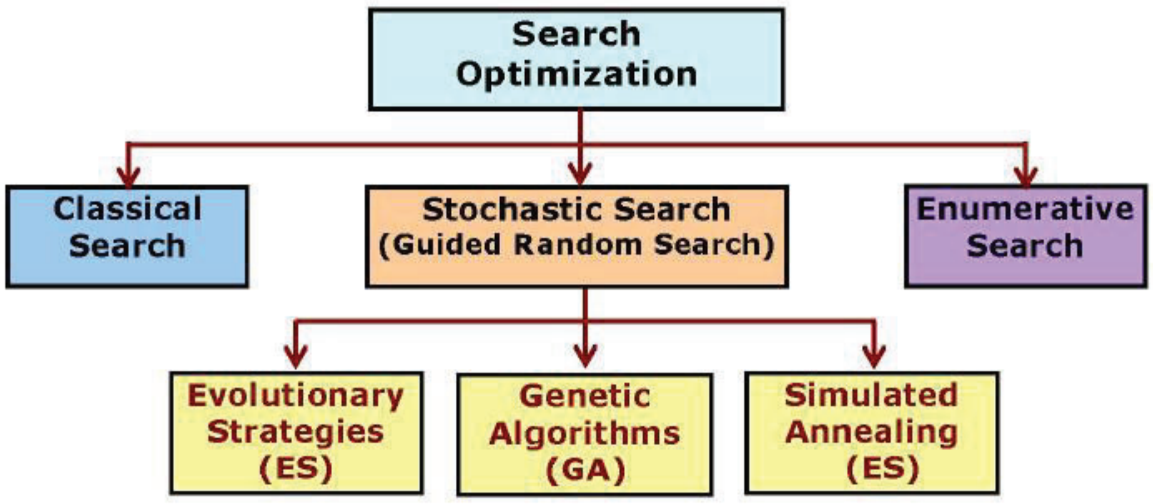
Among the three Non-Linear search methodologies, just mentioned the immediate concern is ***Stochastic search*** which means

— Evolutionary Strategies (ES),

-— Genetic Algorithms (GA), and

— Simulated Annealing (SA).

The two other search methodologies, shown below, the Classical and the Enumerative methods, are first briefly explained. Later the Stochastic methods are discussed in detail. All these methods belong to Non-Linear search.



Non- Linear search methods

Classical or Calculus based search

Uses deterministic approach to find best solutions of an optimization problem.

— The solutions satisfy a set of necessary and sufficient conditions of the optimization problem.

— The techniques are subdivide into *direct* and *indirect methods*.

Enumerative Search

Here the search goes through every point (one point at a time) related to the function's domain space.

— At each point, all possible solutions are generated and tested to find optimum solution.

—It is easy to implement but usually require significant computation.

Thus these techniques are not suitable for applications with large domain spaces. In the field of artificial intelligence, the enumerative methods are subdivided into two categories: *Uninformed and Informed methods.*

- ✓ ***Uninformed or blind methods:***

— Example: Mini-Max algorithm,

— Search all points in the space in a predefined order,

— Used in game playing.

- ✓ ***Informed methods :***

— Example: Alpha-Beta and A*,

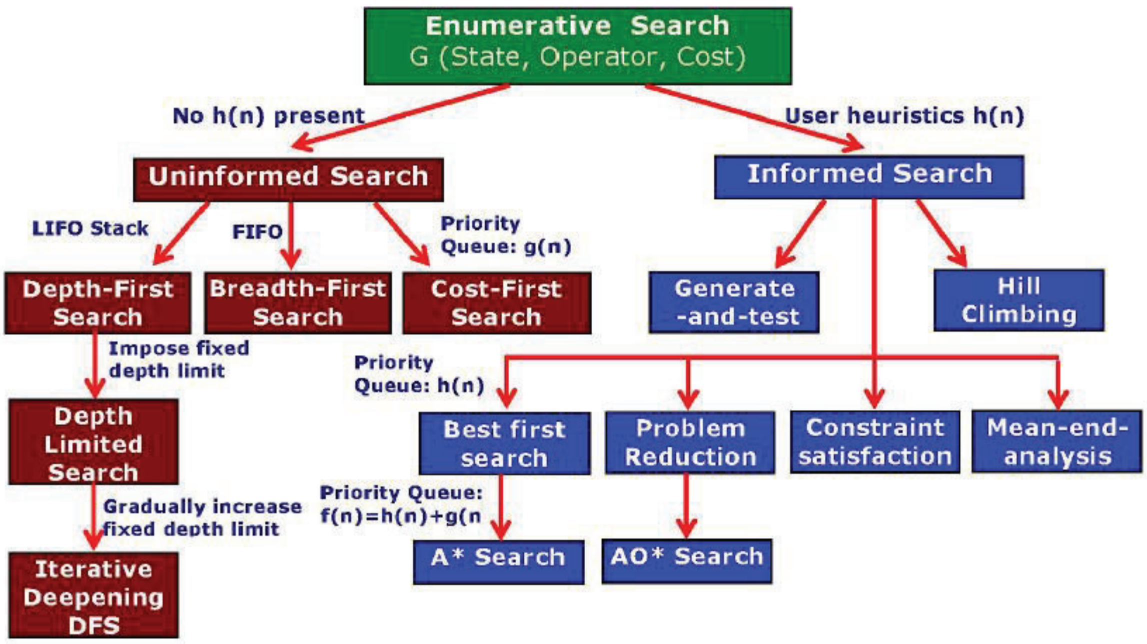
— does more sophisticated search

— uses domain specific knowledge in the form of a cost function or heuristic to reduce cost for search.

The Enumerative search techniques follows, the traditional search and control strategies, in the domain of Artificial Intelligence.

- o The **search methods** explore the search space "intelligently"; means evaluating possibilities without investigating every single possibility.

- There are many **control structures** for search; the depth-first search and breadth-first search are two common search strategies.
- The taxonomy of search algorithms in AI domain is given below.



Enumerative Search Algorithms in AI Domain

Soft computing

3rd class\ 2nd course\ 4th lecture
Computer Science Department\ College of Science for Women

Dr. Noor A. Ibraheem

Genetic Algorithm (GA)

Introduction

What are Genetic Algorithms and why Genetic Algorithm? Algorithms (GAs). Biological background, working principles, Basic. Genetic Algorithm, Flow chart for Genetic Algorithm.

Encoding

Binary Encoding, Value Encoding, Permutation Encoding, Tree Encoding.

Operators of Genetic Algorithm

Random population, Reproduction or Selection Roulette wheel selection, Boltzmann selection; Fitness function; Crossover: One-point crossover, Two-point crossover, Uniform crossover, Arithmetic, Heuristic; Mutation: Flip bit, Boundary, Gaussian, Non-uniform, and Uniform;

What are Genetic Algorithms (GAs)?

Genetic Algorithms (GAs) are *adaptive heuristic search algorithm* **based on** the *evolutionary ideas* of *natural selection and genetics*. Genetic algorithms (GAs) are a ***part of Evolutionary computing***, a rapidly growing area of artificial intelligence. GAs are inspired by **Darwin's theory** about evolution - "survival of the fittest". GAs represent an intelligent exploitation of a random search used to **solve optimization problems**.

GAs, although randomized, exploit historical information to direct the search into the region of better performance within the search space. In nature, competition among individuals for scanty resources results in **the fittest individuals dominating over the weaker ones.**

Solving problems mean looking for solutions, which is best among others. Finding the solution to a problem is often thought:

- In computer science and AI, as a process of **search** through the space of possible solutions. The set of possible solutions defines the search space (also called state space) for a given problem. Solutions or partial solutions are viewed as points in the search space.
- In engineering and mathematics, as a process of **optimization**. The problems are first formulated as mathematical models expressed in terms of functions and then to find a solution, discover the parameters that optimize the model or the function components that provide optimal system performance.

Why Genetic Algorithms

It is better than conventional AI; it is more robust.

- Unlike older AI systems, the GA's do not break easily even if the inputs changed slightly, or in the presence of reasonable noise.
- While performing search in large state-space, multi-modal state-space, or n-dimensional surface, a genetic algorithms offer significant benefits over many other typical search optimization techniques like – linear programming, heuristic, depth-first, breath-first.

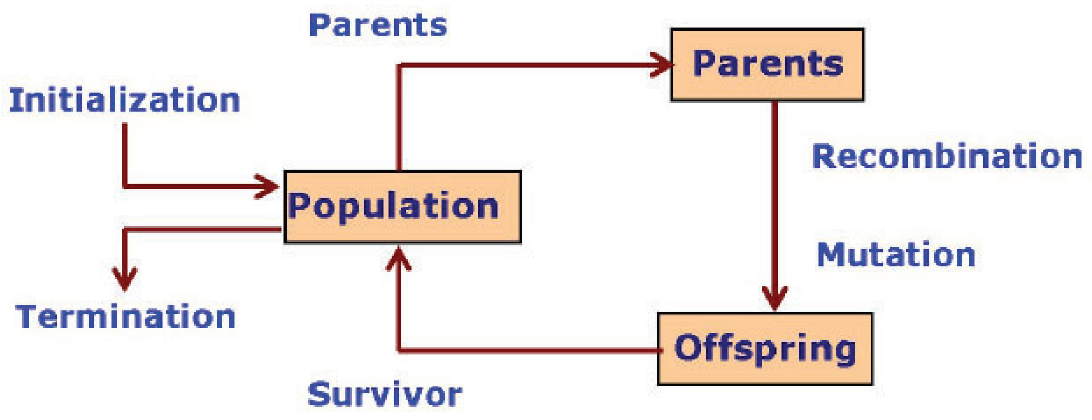
"Genetic Algorithms are good at taking large, potentially huge search spaces and navigating them, looking for optimal combinations of things, the solutions one might not otherwise find in a lifetime." Salvatore Mangano Computer Design, May 1995.

Biological Background - Basic Genetics

- ♣ Every **organism** has a set of rules, describing how that organism is built. All living organisms consist of **cells**.
- ♣ In each cell there is same set of **chromosomes**. Chromosomes are strings of DNA and serve as a model for the whole organism.
- ♣ A chromosome consists of **genes**, blocks of DNA.

- ♣ Each gene encodes a particular protein that represents a **trait** (feature), e.g., color of eyes.
- ♣ Possible settings for a trait (e.g. blue, brown) are called **alleles**.
- ♣ Each gene has its own position in the chromosome called its **locus**.
- ♣ Complete set of genetic material (all chromosomes) is called a **genome**.
- ♣ Particular set of genes in a genome is called **genotype**.
- ♣ The physical expression of the genotype (the organism itself after birth) is called the **phenotype**, its physical and mental characteristics, such as eye color, intelligence etc.
- ♣ When two organisms mate they share their genes; the resultant offspring may end up having half the genes from one parent and half from the other. This process is called **recombination** (cross over).
- ♣ The new created offspring can then be mutated. **Mutation** means, that the elements of DNA are a bit changed. This changes are mainly caused by errors in copying genes from parents.
- ♣ The **fitness** of an organism is measured by success of the organism in its life (survival).

Below shown, the general scheme of evolutionary process in genetic along with pseudo-code.



General Scheme of Evolutionary process

Pseudo-Code

```
Begin
Initialize population with random candidate solution.
EVALUATE each candidate;
Repeat until (termination condition) is satisfied Do
1. Select parents;
2. Recombine pairs of parents;
3. Mutate the resulting offspring;
4. Select individuals or the next generation;
End.
```

Search Space

In solving problems, some solution will be the best among others. The space of all feasible solutions (among which the desired solution resides) is called **search space** (also called state space).

— Each point in the search space represents one **possible solution**.

— Each possible solution can be "marked" by its value (or **fitness**) for the problem.

- The GA looks for the **best solution** among a number of possible solutions represented by one point in the search space.

— Looking for a solution is then equal to looking for some extreme value (minimum or maximum) in the search space.

— At times the search space may be well defined, but usually only a few points in the search space are known.

In using GA, the process of finding solutions generates other points (possible solutions) as evolution proceeds.

Working Principles in General

Before getting into GAs, it is necessary to explain few terms.

- ♣ Chromosome: a set of genes; a chromosome contains the solution in form of genes.
- ♣ Gene: a part of chromosome; a gene contains a part of solution. It determines the solution. e.g. 16743 is a chromosome and 1, 6, 7, 4 and 3 are its genes.
- ♣ Individual: same as chromosome.
- ♣ Population: number of individuals present with same length of chromosome.
- ♣ Fitness: the value assigned to an individual based on how far or close a individual is from the solution; greater the fitness value better the solution it contains.
- ♣ Fitness function: a function that assigns fitness value to the individual. It is problem specific.
- ♣ Breeding: taking two fit individuals and then intermingling there chromosome to create new two individuals.
- ♣ Mutation: changing a random gene in an individual.
- ♣ Selection: selecting individuals for creating the next generation.

Working principles in GA:

Genetic algorithm begins with a set of solutions (represented by chromosomes) called the population.

- Solutions from one population are taken and used to form a new population. This is motivated by the possibility that the new population will be better than the old one.

- Solutions are selected according to their fitness to form new solutions (offspring); more suitable they are, more chances they have to reproduce.

- This is repeated until some condition (e.g. number of populations or improvement of the best solution) is satisfied.

Outline of the Basic Genetic Algorithm

1. **[Start]** Generate random population of ***n*** chromosomes (i.e. suitable solutions for the problem).

2. **[Fitness]** Evaluate the fitness ***f(x)*** of each chromosome x in the population.

3. **[New population]** Create a new population by repeating following steps until the new population is complete.

   (a) **[Selection]** Select two parent chromosomes from a_ population according to their fitness (better the fitness, bigger the chance to be selected)

   (b) **[Crossover]** with a crossover probability, cross over the parents to form new. Offspring (children). If no crossover was performed, offspring is the exact copy of parents.

   (c) **[Mutation]** with a mutation probability, mutate new offspring at each locus (position in chromosome).

4. **[Replace]** Use new generated population for a further run of the algorithm

5. **[Test]** If the end condition is satisfied, stop, and return the best solution in current population

6. **[Loop]** Go to step2

Note: *The genetic algorithm's performance is largely influenced by two operators called crossover and mutation. These two operators are the most important parts of GA.*

Soft computing

3rd class\ 2nd course\ 5th lecture
Computer Science Department\ College of Science for Women

Dr. Noor A. Ibraheem

Genetic Algorithm (GA)

Encoding

Before a genetic algorithm can be put to work on any problem, a method is needed to encode potential solutions to that problem in a form so that computer can process.

- One common approach is to encode solutions as binary strings: sequences of **1's** and **0's**, where the digit at each position represents the value of some aspect of the solution.

Example:

A Gene represents some data (eye color, hair color, sight, etc.).

A chromosome is an array of genes. In binary form

A **Gene** looks like: **(11100010)**

A **Chromosome** looks like: **Gene1 Gene2 Gene3 Gene4**

(11000010, 00001110, 001111010, 10100011)

A chromosome should in some way contain information about solution which it represents; it thus requires encoding. The most popular way of encoding is a **binary string** like:

Chromosome 1: 1101100100110110

Chromosome 2: 1101111000011110

Each bit in the string represent some characteristics of the solution.

- There are many other ways of encoding, e.g., encoding values as integer or real numbers or some permutations and so on.

♣ The virtue of these encoding method depends on the problem to work on.

Binary Encoding

a Binary encoding is the most common to represent information contained. In genetic algorithms, it was first used because of its relative simplicity.

— In binary encoding, every chromosome is a string of bits: **0** or **1**, like

Chromosome1: 101100101100101011100101

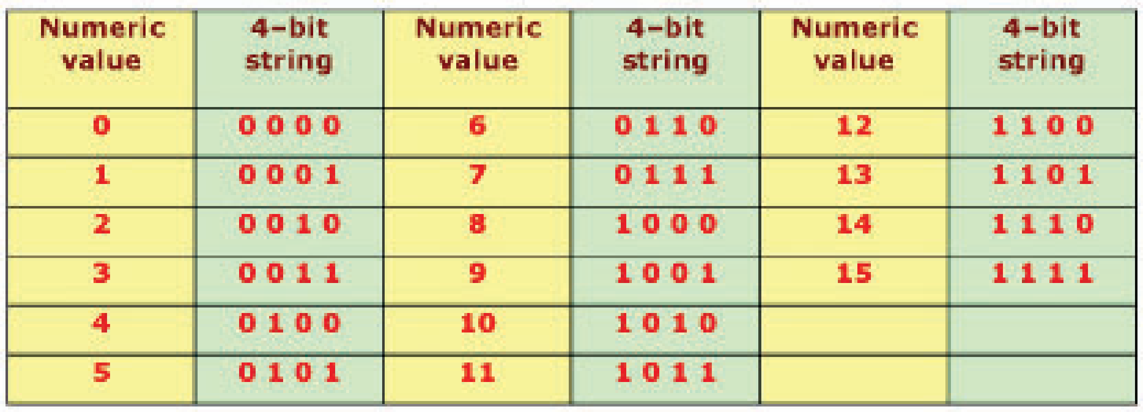
Chromosome2: 111111100000110000011111

— Binary encoding gives many possible chromosomes even with a small number of alleles le possible settings for a trait (features).

— This encoding is often not natural for many problems and sometimes corrections must be made after crossover and/or mutation.

Example 1:

One variable function, say **0** to **15** numbers, numeric values, represented by 4 bit binary string.



Numeric value	4-bit string	Numeric value	4-bit string	Numeric value	4-bit string
0	0 0 0 0	6	0 1 1 0	12	1 1 0 0
1	0 0 0 1	7	0 1 1 1	13	1 1 0 1
2	0 0 1 0	8	1 0 0 0	14	1 1 1 0
3	0 0 1 1	9	1 0 0 1	15	1 1 1 1
4	0 1 0 0	10	1 0 1 0		
5	0 1 0 1	11	1 0 1 1		

Example 2:

Two variable function represented by 4 bit string for each variable.

Let two variables **X_1, X_2** as **(1011 0110).**

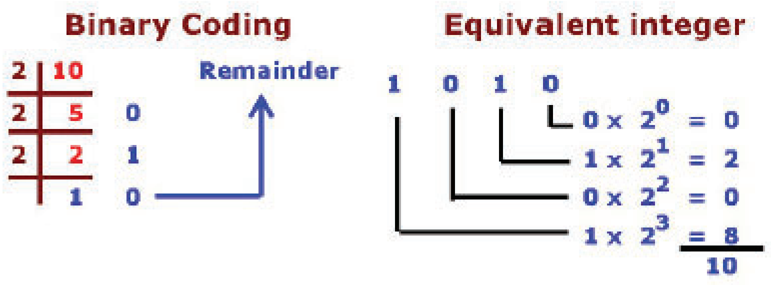
Every variable will have both upper and lower limits as

$$x_i^L \leq x_i \leq x_i^U$$

Because 4-bit string can represent integers from **0** to **15**,

so (**0000** **0000**) and (**1111** **1111**) represent the points for X_1, X_2 as (x_1^L, x_2^L) and (x_1^U, x_2^U) respectively.

Thus, an ***n-bit*** string can represent integers from **0** to **2^n -1**, i.e. **2^n** integers.



Value Encoding

The Value encoding can be used in problems where values such as real numbers are used. Use of binary encoding for this type of problems a would be difficult.

1. In value encoding, every chromosome is a sequence of some values.

2. The Values can be anything connected to the problem, such as: real numbers, characters or objects.

Examples:

ChromosomeA: 1.2324 5.3243 0.4556 2.3293 2.4545

Chromosome B: ABDJEIFIDHDIERJFDLDFLFEGT

Chromosome C: (back), (back), (right), (forward), (left)

3. Value encoding is often necessary to develop some new types of crossovers and mutations specific for the problem.

Permutation Encoding

Permutation encoding can be used in ordering problems, such as traveling salesman problem or task ordering problem.

1. In permutation encoding, every chromosome is a string of numbers that represent a position in a sequence.

Chromosome A: 153264798

Chromosome B: 856723149

2. Permutation encoding is useful for ordering problems. For some problems, crossover and mutation corrections must be made to leave the chromosome consistent.

Examples:

1. The Traveling Salesman problem:

There are cities and given distances between them. Traveling salesman has to visit all of them, but he does not want to travel more than necessary. Find a sequence of cities with a minimal traveled distance. Here, encoded chromosomes describe the order of cities the salesman visits.

2. The Eight Queens problem:

There are eight queens. Find a way to place them onachess board so that no two queens attack each other. Here, encoding describes the position of a queen on each row.

Operators of Genetic Algorithm

Genetic operators used in genetic algorithms maintain genetic diversity. Genetic diversity or variation is a necessity for the process of evolution. Genetic operators are analogous to those which occur in the natural world:

A. **Reproduction** (or Selection);

B. **Crossover** (or Recombination); and

C. **Mutation**.

In addition to these operators, there are some parameters of GA.

One important parameter is **Population size.**

— Population size says how many chromosomes are in population (in one generation).

— If there are only few chromosomes, then GA would have a few possibilities to perform crossover and only a small part of search space is explored.

— If there are many chromosomes, then GA slows down.

— Research shows that after some limit, it is not useful to increase population size, because it does not help in solving the problem faster. The population size depends on the type of encoding and the problem.

A. Reproduction, or Selection

Reproduction is usually the first operator applied on population. From the population, the chromosomes are selected to be parents to crossover and produce offspring.

The problem is how to select these chromosomes?

According to Darwin's evolution theory "survival of the fittest" - the best ones should survive and create new offspring.

— The Reproduction operators are also called Selection operators.

— Selection means extract a subset of genes from an existing population, according to any definition of quality. Every gene has a meaning, so one can derive from the gene a kind of quality measurement called fitness function. Following this quality (fitness value), selection can be performed.

— Fitness function quantifies the optimality of a solution (chromosome) so that a particular solution may be ranked against all the other solutions. The function depicts the closeness of a given 'solution' to the desired result.

Many reproduction operators exists and they all essentially do same thing.

They pick from current population the strings of above average and insert their multiple copies in the mating pool in a probabilistic manner.

The most commonly used methods of selecting chromosomes for parents to crossover are:

- Roulette wheel selection,
- Rank selection
- Boltzmann selection,
- Steady state selection.
- Tournament selection,

Some methods are illustrated next.

Example of Selection

Evolutionary Algorithms is to maximize the function ***f(x)*** $= x^2$ with ***x*** in the integer interval **[0, 31],** i.e., **x= 0, 1 ...30, 31.**

1. The first step is encoding of chromosomes; use binary representation for integers; 5-bits are used to represent integers up to **31**.

2. Assume that the population size is **4**.

3. Generate initial population at random. They are chromosomes or genotypes; e.g., **01101, 11000, 01000, 10011.**

4. Calculate fitness value for each individual.

(a) Decode the individual into an integer (called phenotypes),

$01101 \rightarrow 13;\ 11000 \rightarrow 24;\ 01000 \rightarrow 8;\ 10011 \rightarrow 19;$

(b) Evaluate the fitness according to $f(x) = x^2$

$13 \rightarrow 169;\quad 24 \rightarrow 57;\quad 8 \rightarrow 64;\quad 19 \rightarrow 361.$

5. Select parents (two individuals) for crossover based on their fitness in p_i. Out of many methods for selecting the best chromosomes, if **roulette-wheel selection** is used, then the probability of the ith string in the population is

$$p_i = F_i \;/\; \left(\sum_{j=1}^{n} F_j\right)$$

Where:

F_i is fitness for the string i in the population, expressed as *f(x)*

p_i is probability of the string i being selected,

n is no of individuals in the population, is population size, *n=4*

$n * p_i$ is expected count.

String No	Initial Population	X value	Fitness Fi $f(x) = x^2$	p i	Expected count N * Prob *i*
1	0 1 1 0 1	13	169	0.14	0.58
2	1 1 0 0 0	24	576	0.49	1.97
3	0 1 0 0 0	8	64	0.06	0.22
4	1 0 0 1 1	19	361	0.31	1.23
Sum			1170	1.00	4.00
Average			293	0.25	1.00
Max			576	0.49	1.97

The string no 2 has maximum chance of selection.

Roulette wheel selection (Fitness-Proportionate Selection)

Roulette-wheel selection, also known as Fitness Proportionate Selection, is genetic operator, used for selecting potentially useful solutions for recombination.

In fitness-proportionate selection:

— The chance of an individual's being selected is proportional to its fitness, greater or less than its competitors' fitness.

— Conceptually, this can be thought as a game of Roulette.

The Roulette-wheel simulates 8 individuals with fitness values' F_i, marked at its circumference; e.g.

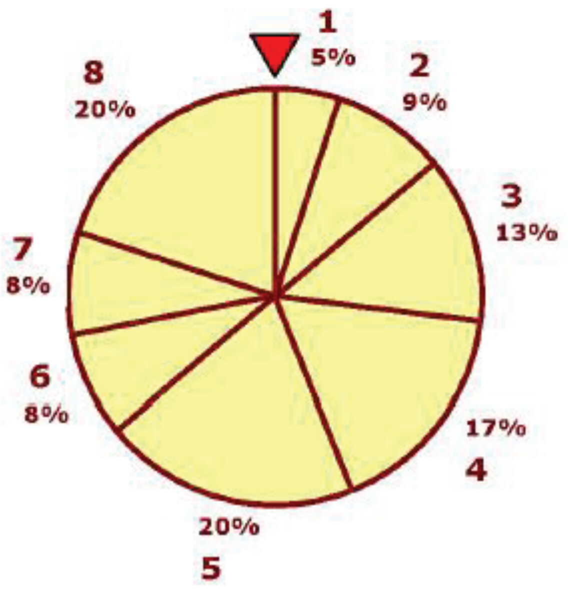
— the 5th individual has a higher fitness than others, so the wheel would choose the 5°" individual more than other individuals.

—the fitness of the individuals is calculated as the wheel is spun n= 8 times, each time selecting an instance, of the string, chosen by the wheel pointer.

Probability of ith string is:

$$p_i = F_i \, / \, (\sum_{j=1}^{n} F_j)$$

Where ***n =no of individuals***, called population size; ***p_i = probability*** of ***ith*** string being selected; ***Fi = fitness*** for ***ith***



Roulette-wheel Shows 8 individual with fitness.

Soft computing

3rd class\ 2nd course\ 6th lecture
Computer Science Department\ College of Science for Women

Dr. Noor A. Ibraheem

Genetic Algorithm (GA)

B. Crossover

Crossover is a genetic operator that combines (mates) two chromosomes (parents) to produce a new chromosome (offspring). The idea behind crossover is that the new chromosome may be better than both of the parents if it takes the best characteristics from each of the parents.

Crossover occurs during evolution according to a user-definable crossover probability. Crossover selects genes from parent chromosomes and creates a new offspring.

The Crossover operators are of many types.

— One simple way is, **One-Point crossover**.

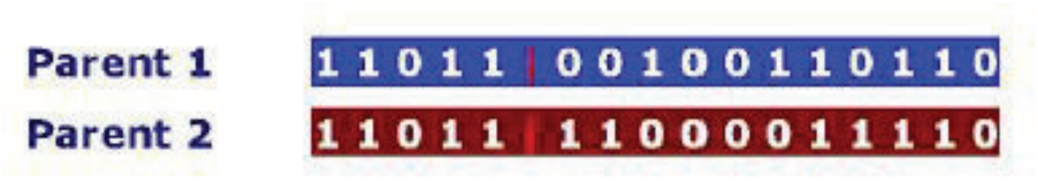
— The others are **Two Point, Uniform, Arithmetic**, and **Heuristic crossovers**.

The operators are selected based on the way chromosomes are encoded.

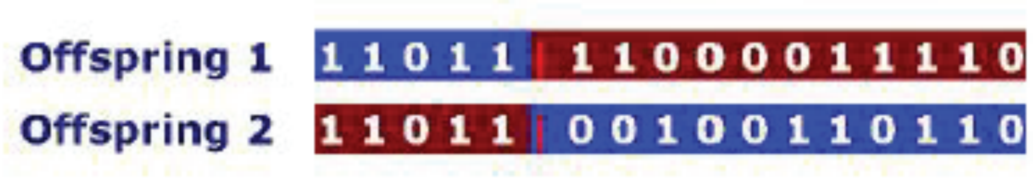
♣ One-Point Crossover

One-Point crossover operator randomly selects one crossover point and then copy everything before this point from the first parent and then everything after the crossover point copy from the second parent. The Crossover would then look as shown below.

Consider the two parents selected for crossover.



Interchanging the parents chromosomes after the crossover points, The Offspring produced are:

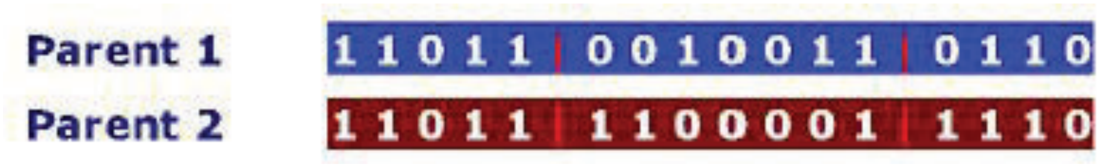


Note: The symbol, a vertical line, | is the chosen crossover point.

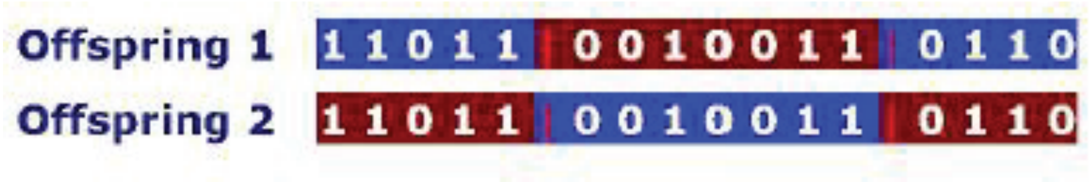
♣ Two-Point Crossover

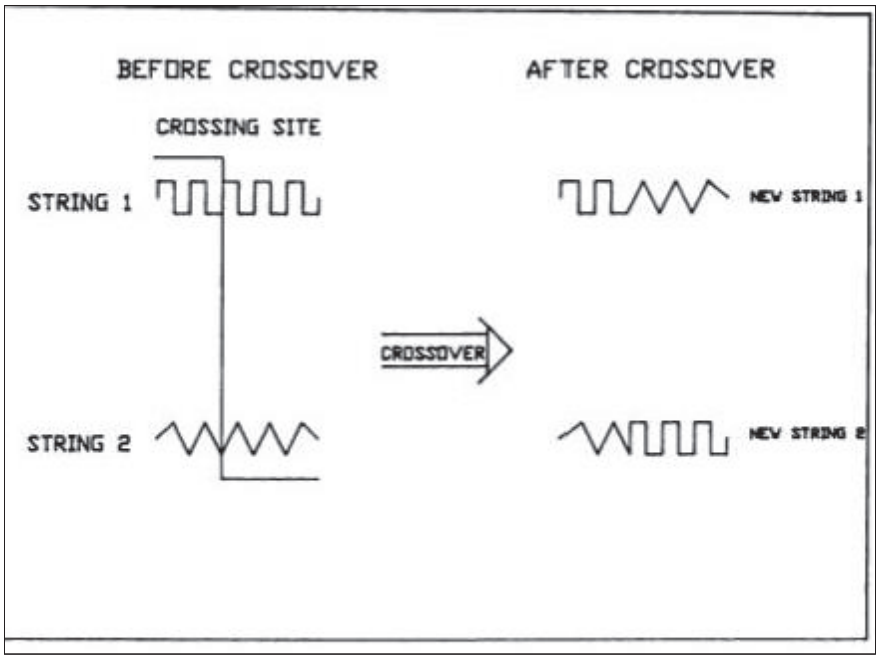
Two-Point crossover operator randomly selects two crossover points within a chromosome then interchanges the two parent chromosomes between these points to produce two new offspring.

Consider the two parents selected for crossover :



Interchanging the parents chromosomes between the crossover points, The Offspring produced are :





One-point Crossover example

C. Mutation

After a crossover is performed, mutation takes place. Mutation is a genetic operator used to maintain genetic diversity from one generation of a population of chromosomes to the next.

Mutation occurs during evolution according to a user-definable mutation probability, usually set to fairly low value, say **0.01** a good first choice. Mutation alters one or more gene values in a chromosome from its initial state. This can result in entirely new gene values being added to the gene pool. With the new gene values, the genetic algorithm may be able to arrive at better solution than was previously possible.

Mutation is an important part of the genetic search, helps to prevent the population from stagnating at any local optima. Mutation is intended to prevent the search falling into a local optimum of the state space. The Mutation operators are of many type.

— one simple way is, **Flip Bit**.

— the others are **Boundary**, **Non-Uniform, Uniform,** and **Gaussian**.

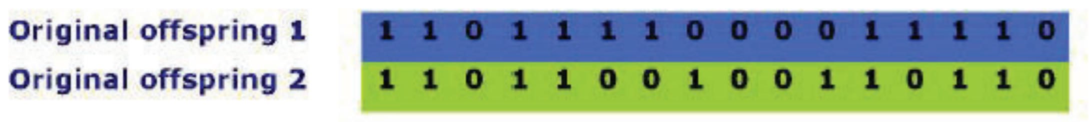
The operators are selected based on the way chromosomes are encoded .

♣ Flip Bit

The mutation operator simply inverts the value of the chosen gene.

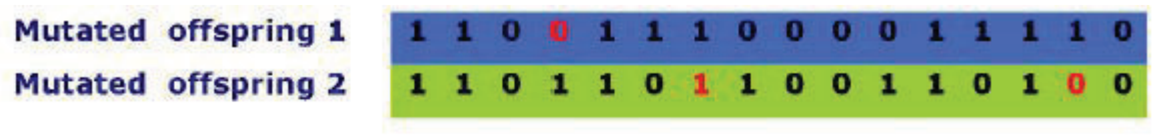
i.e. 0 goes to 1 and 1 goes to 0.

This mutation operator can only be used for binary genes. Consider the two original off-springs selected for mutation.



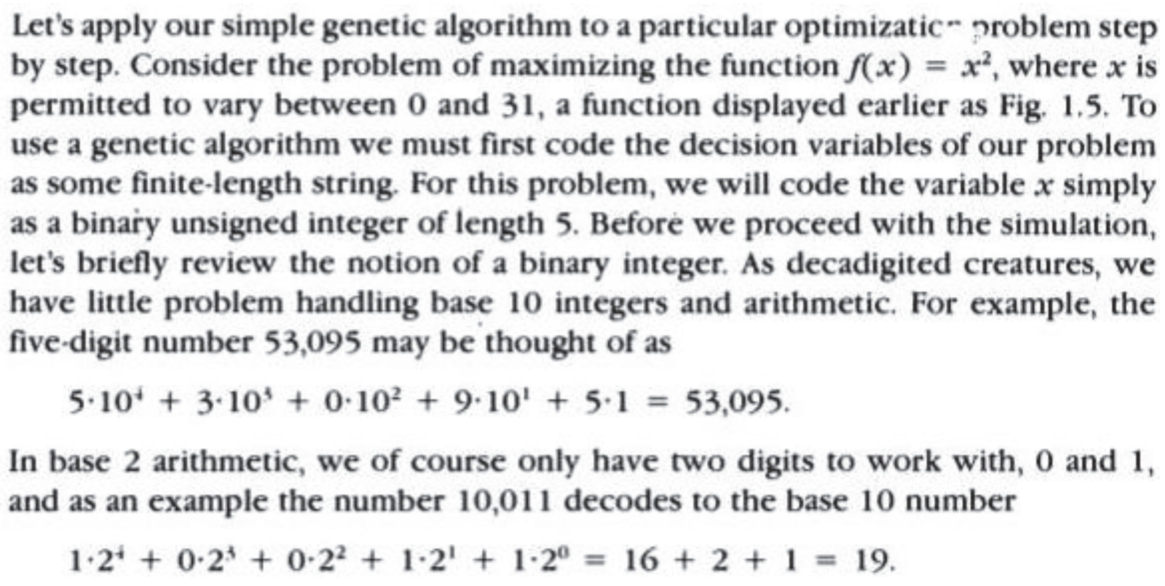
Original offspring 1	1	1	0	1	1	1	1	0	0	0	0	1	1	1	1	0
Original offspring 2	1	1	0	1	1	0	0	1	0	0	1	1	0	1	1	0

Invert the value of the chosen gene as 0 to1 and 1 to 0 The Mutated Off-spring produced are:



Mutated offspring 1	1	1	0	0	1	1	1	0	0	0	0	1	1	1	1	0
Mutated offspring 2	1	1	0	1	1	0	1	1	0	0	1	1	0	1	0	0

Example: Genetic algorithm at work - simulate example by hand



Let's apply our simple genetic algorithm to a particular optimization problem step by step. Consider the problem of maximizing the function $f(x) = x^2$, where x is permitted to vary between 0 and 31, a function displayed earlier as Fig. 1.5. To use a genetic algorithm we must first code the decision variables of our problem as some finite-length string. For this problem, we will code the variable x simply as a binary unsigned integer of length 5. Before we proceed with the simulation, let's briefly review the notion of a binary integer. As decadigited creatures, we have little problem handling base 10 integers and arithmetic. For example, the five-digit number 53,095 may be thought of as

$$5 \cdot 10^4 + 3 \cdot 10^3 + 0 \cdot 10^2 + 9 \cdot 10^1 + 5 \cdot 1 = 53{,}095.$$

In base 2 arithmetic, we of course only have two digits to work with, 0 and 1, and as an example the number 10,011 decodes to the base 10 number

$$1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 16 + 2 + 1 = 19.$$

TABLE 1.2 A Genetic Algorithm by Hand

String No.	Initial Population (Randomly Generated)	x Value (Unsigned Integer)	$f(x)$ x^2	pselect, $\frac{f_i}{\Sigma f}$	Expected count $\frac{f_i}{\bar{f}}$	Actual Count from Roulette Wheel
1	0 1 1 0 1	13	169	0.14	0.58	1
2	1 1 0 0 0	24	576	0.49	1.97	2
3	0 1 0 0 0	8	64	0.06	0.22	0
4	1 0 0 1 1	19	361	0.31	1.23	1
Sum			1170	1.00	4.00	4.0
Average			293	0.25	1.00	1.0
Max			576	0.49	1.97	2.0

TABLE 1.2 *(Continued)*

Mating Pool after Reproduction (Cross Site Shown)	Mate (Randomly Selected)	Crossover Site (Randomly Selected)	New Population	x Value	$f(x)$ x^2
0 1 1 0 | 1	2	4	0 1 1 0 0	12	144
1 1 0 0 | 0	1	4	1 1 0 0 1	25	625
1 1 | 0 0 0	4	2	1 1 0 1 1	27	729
1 0 | 0 1 1	3	2	1 0 0 0 0	16	256
					1754
					439
					729

NOTES:

1) Initial population chosen by four repetitions of five coin tosses where heads = 1, tails = 0.
2) Reproduction performed through 1 part in 8 simulation of roulette wheel selection (three coin tosses).
3) Crossover performed through binary decoding of 2 coin tosses (TT = 00_2 = 0 = cross site 1, HH = 11_2 = 3 = cross site 4).
4) Crossover probability assumed to be unity $p_c = 1.0$.
5) Mutation probability assumed to be 0.001, $p_m = 0.001$, Expected mutations = $5 \cdot 4 \cdot 0.001$ = 0.02. No mutations expected during a single generation. None simulated.

The General steps of Genetic Algorithm

In the genetic algorithm process is as follows:

Step 1. Determine the number of chromosomes, generation, and mutation rate and crossover rate value

Step 2. Generate chromosome-chromosome number of the population, and the initialization value of the genes chromosome-chromosome with a random value

Step 3. Process steps 4-7 until the number of generations is met

Step 4. Evaluation of fitness value of chromosomes by calculating objective function

Step 5. Chromosomes selection

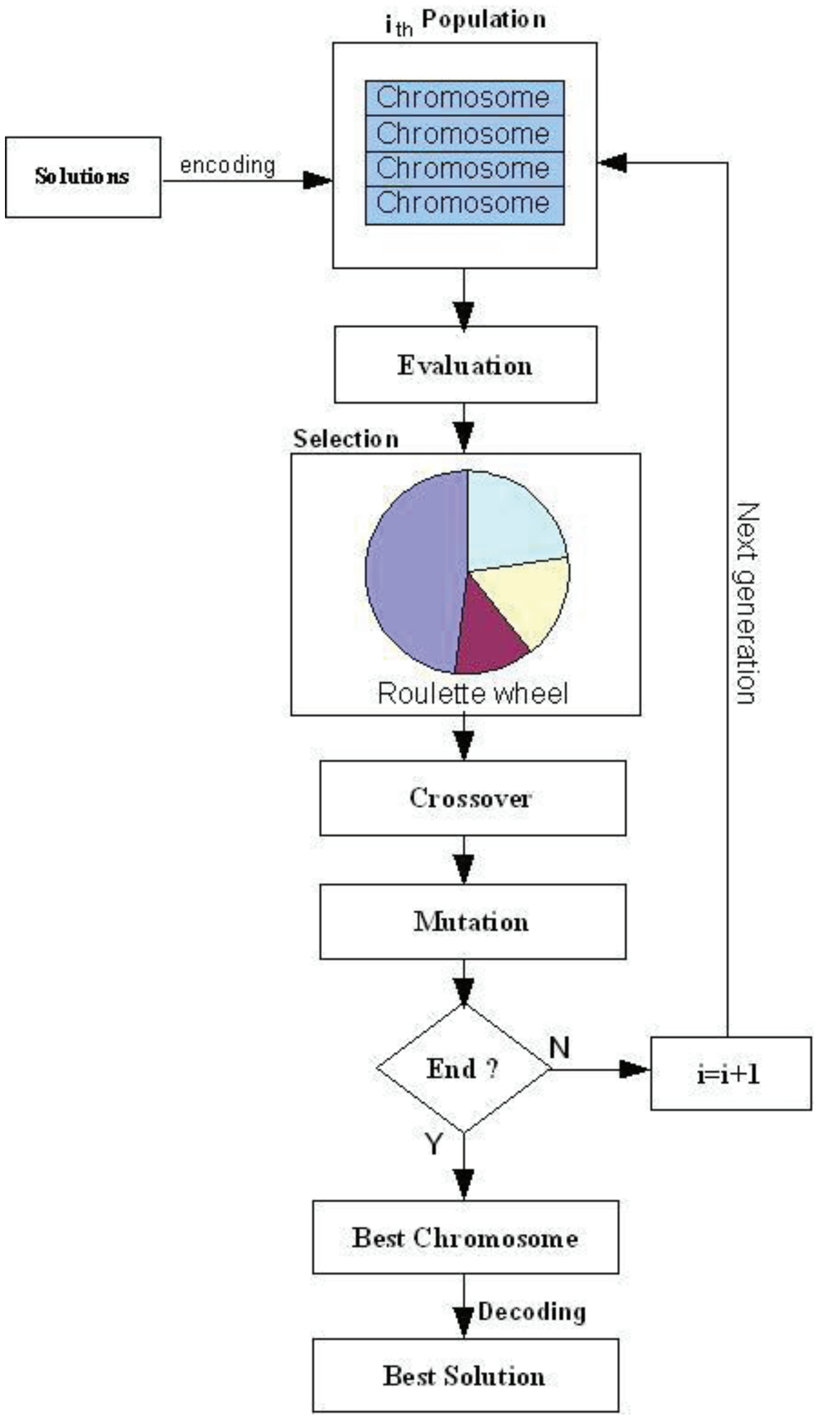
Step 5. Crossover

Step 6. Mutation

Step 7. New Chromosomes (Offspring)

Step 8. Solution (Best Chromosomes)

The flowchart of algorithm can be seen in the following Figure.



Genetic algorithm flowchart

Numerical Example

Here are examples of applications that use genetic algorithms to solve the problem of combination. Suppose there is equality a +2 b +3 c +4 d = 30, genetic algorithm will be used to find the value of a, b, c, and d that satisfy the above equation. First we should formulate the objective function, for this problem

the objective is minimizing the value of function f(x) where f (x) = ((a + 2b + 3c + 4d) - 30). Since there are four variables in the equation, namely a, b, c, and d, we can compose the chromosome as follow:

a	b	c	d

To speed up the computation, we can restrict that the values of variables a, b, c, and d are integers between 0 and 30.

Step 1. Initialization

For example we define the number of chromosomes in population are 6, then we generate random value of gene a, b, c, d for 6 chromosomes

Chromosome[1] = [a;b;c;d] = [12;05;23;08]

Chromosome[2] = [a;b;c;d] = [02;21;18;03]

Chromosome[3] = [a;b;c;d] = [10;04;13;14]

Chromosome[4] = [a;b;c;d] = [20;01;10;06]

Chromosome[5] = [a;b;c;d] = [01;04;13;19]

Chromosome[6] = [a;b;c;d] = [20;05;17;01] Step 2. Evaluation

We compute the objective function value for each chromosome produced in initialization step:

F_obj[1] = Abs((12 + 2*05 + 3*23 + 4*08) - 30)

= Abs((12 + 10 + 69 + 32) - 30)

= Abs(123 - 30)

= 93

F_obj[2] = Abs((02 + 2*21 + 3*18 + 4*03) - 30)

= Abs((02 + 42 + 54 + 12) - 30)

= Abs(110 - 30)

= 80

F_obj[3] = Abs((10 + 2*04 + 3*13 + 4*14) - 30)

= Abs((10 + 08 + 39 + 56) - 30)

= Abs(113 - 30)

= 83

F_obj[4] = Abs((20 + 2*01 + 3*10 + 4*06) - 30)

= Abs((20 + 02 + 30 + 24) - 30)

= Abs(76 - 30)

= 46

F_obj[5] = Abs((01 + 2*04 + 3*13 + 4*19) - 30)

= Abs((01 + 08 + 39 + 76) - 30)

= Abs(124 - 30)

= 94

F_obj[6] = Abs((20 + 2*05 + 3*17 + 4*01) - 30)

= Abs((20 + 10 + 51 + 04) - 30)

= Abs(85 - 30)

= 55

Step 3. Selection

1. The fittest chromosomes have higher probability to be selected for the next generation. To compute fitness probability we must compute the fitness of each chromosome. To avoid divide by zero problem, the value of F_obj is added by 1.

Fitness[1] = 1 / (1+F_obj[1])

= 1 / 94

= 0.0106

Fitness[2] = 1 / (1+F_obj[2])

= 1 / 81

= 0.0123

Fitness[3] = 1 / (1+F_obj[3])

= 1 / 84

= 0.0119

Fitness[4] = 1 / (1+F_obj[4])

= 1 / 47

= 0.0213

Fitness[5] = 1 / (1+F_obj[5])

= 1 / 95

= 0.0105

Fitness[6] = 1 / (1+F_obj[6])

= 1 / 56

= 0.0179

Total = 0.0106 + 0.0123 + 0.0119 + 0.0213 + 0.0105 + 0.0179 3.

= 0.0845

The probability for each chromosomes is formulated by: P[i] = Fitness[i] / Total

P[1] = 0.0106 / 0.0845

= 0.1254

P[2] = 0.0123 / 0.0845

= 0.1456

P[3] = 0.0119 / 0.0845

= 0.1408

P[4] = 0.0213 / 0.0845

= 0.2521

P[5] = 0.0105 / 0.0845

= 0.1243

P[6] = 0.0179 / 0.0845

= 0.2118

From the probabilities above we can see that Chromosome 4 that has the highest fitness, this chromosome has highest probability to be selected for next generation chromosomes. For the selection process we use roulette wheel, for that we should compute the cumulative probability values:

C[1] = 0.1254

C[2] = 0.1254 + 0.1456

= 0.2710

C[3] = 0.1254 + 0.1456 + 0.1408

= 0.4118

C[4] = 0.1254 + 0.1456 + 0.1408 + 0.2521

= 0.6639

C[5] = 0.1254 + 0.1456 + 0.1408 + 0.2521 + 0.1243

= 0.7882

C[6] = 0.1254 + 0.1456 + 0.1408 + 0.2521 + 0.1243 + 0.2118

= 1.0

Having calculated the cumulative probability of selection process using roulette-wheel can be done. The process is to generate random number R in the range 0-1 as follows.

R[1] = 0.201

R[2] = 0.284

R[3] = 0.099

R[4] = 0.822

R[5] = 0.398

R[6] = 0.501

If random number R [1] is greater than P [1] and smaller than P [2] then select Chromosome [2] as a chromosome in the new population for next generation:

NewChromosome[1] = Chromosome[2]

NewChromosome[2] = Chromosome[3]

NewChromosome[3] = Chromosome[1]

NewChromosome[4] = Chromosome[6]

NewChromosome[5] = Chromosome[3]

NewChromosome[6] = Chromosome[4]

Chromosome in the population thus became:

Chromosome[1] = [02;21;18;03]

Chromosome[2] = [10;04;13;14]

Chromosome[3] = [12;05;23;08]

Chromosome[4] = [20;05;17;01]

Chromosome[5] = [10;04;13;14]

Chromosome[6] = [20;01;10;06]

Step 4. Crossover

In this example, we use one-cut point, i.e. randomly select a position in the parent chromosome then exchanging sub-chromosome. Parent chromosome which will mate is randomly selected and the number of mate Chromosomes is controlled using crossover_rate (ρc) parameters.

Chromosome k will be selected as a parent if R [k] <ρc. Suppose we set that the crossover rate is 25%, then Chromosome number k will be selected for crossover if random generated value for Chromosome k below 0.25. The process is as follows: First we generate a random number R as the number of population.

R[1] = 0.191

R[2] = 0.259

R[3] = 0.760

R[4] = 0.006

R[5] = 0.159

R[6] = 0.340

For random number R above, parents are Chromosome [1], Chromosome [4] and Chromosome [5] will be selected for crossover.

Chromosome[1] >< Chromosome[4]

Chromosome[4] >< Chromosome[5]

Chromosome[5] >< Chromosome[1]

After chromosome selection, the next process is determining the position of the crossover point. This is done by generating random numbers between 1 to (length of Chromosome – 1). In this case, generated random numbers should be between 1 and 3. After we get the crossover point, parents Chromosome will be cut at crossover point and its gens will be interchanged. For example we generated 3 random number and we get:

C[1] = 1

C[2] = 1

C[3] = 2

Then for first crossover, second crossover and third crossover, parent's gens will be cut at gen number 1, gen number 1 and gen number 3 respectively, e.g.

Chromosome[1] = Chromosome[1] >< Chromosome[4]

= [02;21;18;03] >< [20;05;17;01]

= [02;05;17;01]

Chromosome[4] = Chromosome[4] >< Chromosome[5]

= [20;05;17;01] >< [10;04;13;14]

= [20;04;13;14]

Chromosome[5] = Chromosome[5] >< Chromosome[1]

= [10;04;13;14] >< [02;21;18;03]

= [10;04;18;03]

Thus Chromosome population after experiencing a crossover process:

Chromosome[1] = [02;05;17;01]

Chromosome[2] = [10;04;13;14]

Chromosome[3] = [12;05;23;08]

Chromosome[4] = [20;04;13;14]

Chromosome[5] = [10;04;18;03]

Chromosome[6] = [20;01;10;06]

Chromosome k will be selected as a parent if R [k] <ρc. Suppose we set that the crossover rate is 25%, then Chromosome number k will be selected for crossover if random generated value for Chromosome k below 0.25. The process is as follows: First we generate a random number R as the number of population.

R[1] = 0.191

R[2] = 0.259

R[3] = 0.760

R[4] = 0.006

R[5] = 0.159

R[6] = 0.340

For random number R above, parents are Chromosome [1], Chromosome [4] and Chromosome [5] will be selected for crossover.

Chromosome[1] >< Chromosome[4]

Chromosome[4] >< Chromosome[5]

Chromosome[5] >< Chromosome[1]

After chromosome selection, the next process is determining the position of the crossover point. This is done by generating random numbers between 1 to (length of Chromosome – 1). In this case, generated random numbers should be between 1 and 3. After we get the crossover point, parents Chromosome will be cut at crossover point and its gens will be interchanged. For example we generated 3 random number and we get:

C[1] = 1

C[2] = 1

C[3] = 2

Then for first crossover, second crossover and third crossover, parent's gens will be cut at gen number 1, gen number 1 and gen number 3 respectively, e.g.

Chromosome[1] = Chromosome[1] >< Chromosome[4]

= [02;21;18;03] >< [20;05;17;01]

= [02;05;17;01]

Chromosome[4] = Chromosome[4] >< Chromosome[5]

= [20;05;17;01] >< [10;04;13;14]

= [20;04;13;14]

Chromosome[5] = Chromosome[5] >< Chromosome[1]

= [10;04;13;14] >< [02;21;18;03]

= [10;04;18;03]

Thus Chromosome population after experiencing a crossover process:

Chromosome[1] = [02;05;17;01]

Chromosome[2] = [10;04;13;14]

Chromosome[3] = [12;05;23;08]

Chromosome[4] = [20;04;13;14]

Chromosome[5] = [10;04;18;03]

Chromosome[6] = [20;01;10;06]

Step 5. Mutation

Number of chromosomes that have mutations in a population is determined by the mutation rate parameter. Mutation process is done by replacing the gen at random position with a new value. The process is as follows. First we must calculate the total length of gen in the population. In this case the total length of gen is total_gen = number_of_gen_in_Chromosome * number of population

= 4 * 6

= 24

Mutation process is done by generating a random integer between 1 and total_gen (1 to 24). If generated random number is smaller than mutation_rate(ρm) variable then marked the position of gen in chromosomes. Suppose we define ρm 10%, it is expected that 10% (0.1) of total_gen in the population that will be mutated:

number of mutations = 0.1 * 24

= 2.4

≈ 2

Suppose generation of random number yield 12 and 18 then the chromosome which have mutation are Chromosome number 3 gen number 4 and Chromosome 5 gen number 2. The value of mutated gens at mutation point is replaced by random number between 0-30. Suppose generated random number are 2 and 5 then Chromosome composition after mutation are:

Chromosome[1] = [02;05;17;01]

Chromosome[2] = [10;04;13;14]

Chromosome[3] = [12;05;23;02]

Chromosome[4] = [20;04;13;14]

Chromosome[5] = [10;05;18;03]

Chromosome[6] = [20;01;10;06]

Finishing mutation process then we have one iteration or one generation of the genetic algorithm. We can now evaluate the objective function after one generation:

Chromosome[1] = [02;05;17;01]

F_obj[1] = Abs((02 + 2*05 + 3*17 + 4*01) - 30)

= Abs((2 + 10 + 51 + 4) - 30)

= Abs(67 - 30)

= 37

Chromosome[2] = [10;04;13;14]

F_obj[2] = Abs((10 + 2*04 + 3*13 + 4*14) - 30)

= Abs((10 + 8 + 33 + 56) - 30)

= Abs(107 - 30)

= 77

Chromosome[3] = [12;05;23;02]

F_obj[3] = Abs((12 + 2*05 + 3*23 + 4*02) - 30)

= Abs((12 + 10 + 69 + 8) - 30)

= Abs(87 - 30)

= 47

Chromosome[4] = [20;04;13;14]

F_obj[4] = Abs((20 + 2*04 + 3*13 + 4*14) - 30)

= Abs((20 + 8 + 39 + 56) - 30)

= Abs(123 - 30)

= 93

Chromosome[5] = [10;05;18;03]

F_obj[5] = Abs((10 + 2*05 + 3*18 + 4*03) - 30)

= Abs((10 + 10 + 54 + 12) - 30)

= Abs(86 - 30)

= 56

Chromosome[6] = [20;01;10;06]

F_obj[6] = Abs((20 + 2*01 + 3*10 + 4*06) - 30)

= Abs((20 + 2 + 30 + 24) - 30)

= Abs(76 - 30)

= 46

From the evaluation of new Chromosome we can see that the objective function is decreasing, this means that we have better Chromosome or solution compared with previous chromosome generation. New Chromosomes for next iteration are:

Chromosome[1] = [02;05;17;01]

Chromosome[2] = [10;04;13;14]

Chromosome[3] = [12;05;23;02]

Chromosome[4] = [20;04;13;14]

Chromosome[5] = [10;05;18;03]

Chromosome[6] = [20;01;10;06]

These new Chromosomes will undergo the same process as the previous generation of Chromosomes such as evaluation, selection, crossover and mutation and at the end it produce new generation of Chromosome for the next iteration. This process will be repeated until a predetermined number of generations. For this example, after running 50 generations, best chromosome is obtained:

Chromosome = [07; 05; 03; 01]

This means that:

a = 7, b = 5, c = 3, d = 1

If we use the number in the problem equation

a +2 b +3 c +4 d = 30

7 + (2 * 5) + (3 * 3) + (4 * 1) = 30

We can see that the value of variable a, b, c and d generated by genetic algorithm can satisfy

Soft computing

3rd class\ 2nd course\ 7th lecture
Computer Science Department\ College of Science for Women

Dr. Noor A. Ibraheem

Neural Networks (NNs)

Why Neural Networks And Why Now?

As modern computers become ever more powerful, scientists continue to be challenged to use machines effectively for tasks that are relatively simple for humans.

Based on examples, together with some feedback from a “teacher,” we learn

easily to recognize the letter A or distinguish a cat from a bird. More experience allows us to refine our responses and improve our performance. Although eventually, we may be able to describe rules by which we can make such decisions, these do not necessarily reflect the actual process we use. Even without a teacher, we can group similar patterns together. Yet another common human activity is trying to achieve a goal that involves maximizing a resource (time with one’s family, for example) while satisfying certain constraints (such as the need to earn a living). Each of these types of problems illustrates tasks for which computer solutions may be sought.

Traditional, sequential, logic-based digital computing excels in many areas, but has been less successful for other types of problems. The development of artificial neural networks began approximately 50 years ago, motivated by a desire to try both to understand the brain and to emulate some of its strengths.

WHAT IS A NEURAL NET?

1.2.1 Artificial Neural Networks

An *artificial neural network* is an information-processing system that has certain performance characteristics in common with biological neural networks. Artificial neural networks have been developed as generalizations of mathematical models of human cognition or neural biology, based on the assumptions that:

1. Information processing occurs at many simple elements called neurons.
2. Signals are passed between neurons over connection links.
3. Each connection link has an associated weight, which, in a typical neural net, multiplies the signal transmitted.
4. Each neuron applies an activation function (usually nonlinear) to its net input (sum of weighted input signals) to determine its output signal.

A neural network is characterized by (1) its pattern of connections between the neurons (called its *architecture*), (2) its method of determining the weights on the connections (called its *training*, or *learning, algorithm*), and (3) its *activation function*.

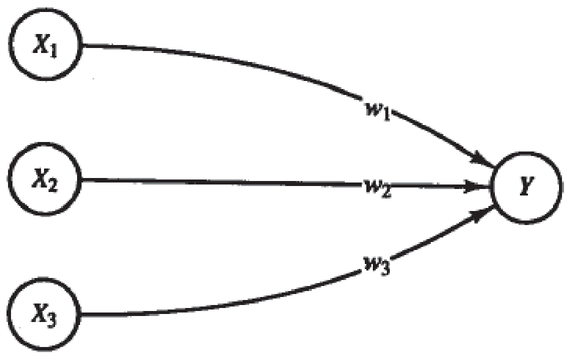
Since *what* distinguishes (artificial) neural networks from other approaches to information processing provides an introduction to both *how* and *when* to use neural networks, let us consider the defining characteristics of neural networks further.

A neural net consists of a large number of simple processing elements called *neurons, units, cells,* or *nodes*. Each neuron is connected to other neurons by means of directed communication links, each with an associated weight. The weights represent information being used by the net to solve a problem. Neural nets can be applied to a wide variety of problems, such as storing and recalling data or patterns, classifying patterns, performing general mappings from input patterns to output patterns, grouping similar patterns, or finding solutions to constrained optimization problems.

Each neuron has an internal state, called its *activation* or *activity level,* which is a function of the inputs it has received. Typically, a neuron sends its activation as a signal to several other neurons. It is important to note that a neuron can send only one signal at a time, although that signal is broadcast to several other neurons.

For example, consider a neuron Y, illustrated in Figure 1.1, that receives inputs from neurons X_1, X_2, and X_3. The activations (output signals) of these neurons are x_1, x_2, and x_3, respectively. The weights on the connections from X_1, X_2, and X_3 to neuron Y are w_1, w_2, and w_3, respectively. The net input, *y_in*, to neuron Y is the sum of the weighted signals from neurons X_1, X_2, and X_3, i.e.,

$$y\_in = w_1x_1 + w_2x_2 + w_3x_3.$$



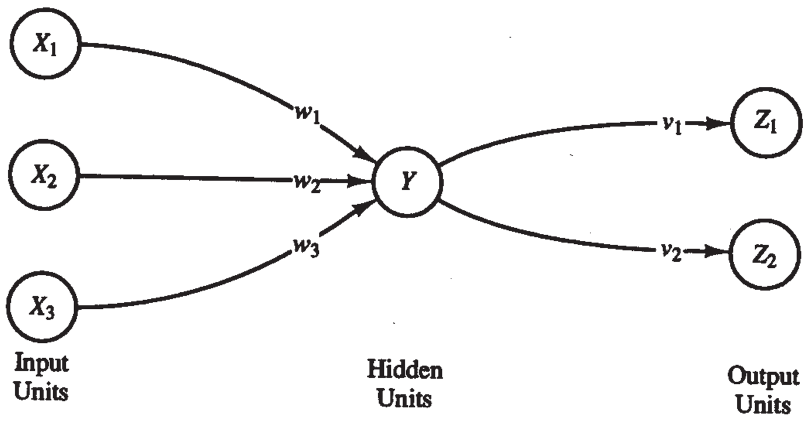
A simple (artificial) neuron.

The activation y of neuron Y is given by some function of its net input, $y = f(y\_in)$, e.g., the logistic sigmoid function (an S-shaped curve)

$$f(x) = \frac{1}{1 + \exp(-x)},$$

or any of a number of other activation functions. Several common activation functions are illustrated in Section 1.4.3.

Although the neural network in Figure 1.2 is very simple, the presence of a hidden unit, together with a nonlinear activation function, gives it the ability to solve many more problems than can be solved by a net with only input and output units. On the other hand, it is more difficult to train (i.e., find optimal values for the weights) a net with hidden units. The arrangement of the units (the architecture



A very simple neural network.

of the net) and the method of training the net are discussed further in Section 1.4. A detailed consideration of these ideas for specific nets, together with simple examples of an application of each net, is the focus of the following chapters.

Biological Neural Networks

The extent to which a neural network models a particular biological neural system varies. For some researchers, this is a primary concern; for others, the ability of the net to perform useful tasks (such as approximate or represent a function) is more important than the biological plausibility of the net. Although our interest lies almost exclusively in the computational capabilities of neural networks, we shall present a brief discussion of some features of biological neurons that may help to clarify the most important characteristics of artificial neural networks. In addition to being the original inspiration for artificial nets, biological neural systems suggest features that have distinct computational advantages.

There is a close analogy between the structure of a biological neuron (i.e., a brain or nerve cell) and the processing element (or artificial neuron) presented in the rest of this book. In fact, the structure of an individual neuron varies much less from species to species than does the organization of the system of which the neuron is an element.

A biological neuron has three types of components that are of particular interest in understanding an artificial neuron: its *dendrites, soma,* and *axon.* The many dendrites receive signals from other neurons. The signals are electric impulses that are transmitted across a synaptic gap by means of a chemical process. The action of the chemical transmitter modifies the incoming signal (typically, by scaling the frequency of the signals that are received) in a manner similar to the action of the weights in an artificial neural network.

The soma, or cell body, sums the incoming signals. When sufficient input is received, the cell fires; that is, it transmits a signal over its axon to other cells. It is often supposed that a cell either fires or doesn't at any instant of time, so that transmitted signals can be treated as binary. However, the frequency of firing varies and can be viewed as a signal of either greater or lesser magnitude. This corresponds to looking at discrete time steps and summing all activity (signals received or signals sent) at a particular point in time.

The transmission of the signal from a particular neuron is accomplished by an action potential resulting from differential concentrations of ions on either side of the neuron's axon sheath (the brain's ''white matter''). The ions most directly involved are potassium, sodium, and chloride.

A generic biological neuron is illustrated in Figure 1.3, together with axons from two other neurons (from which the illustrated neuron could receive signals) and dendrites for two other neurons (to which the original neuron would send signals). Several key features of the processing elements of artificial neural networks are suggested by the properties of biological neurons, viz., that:

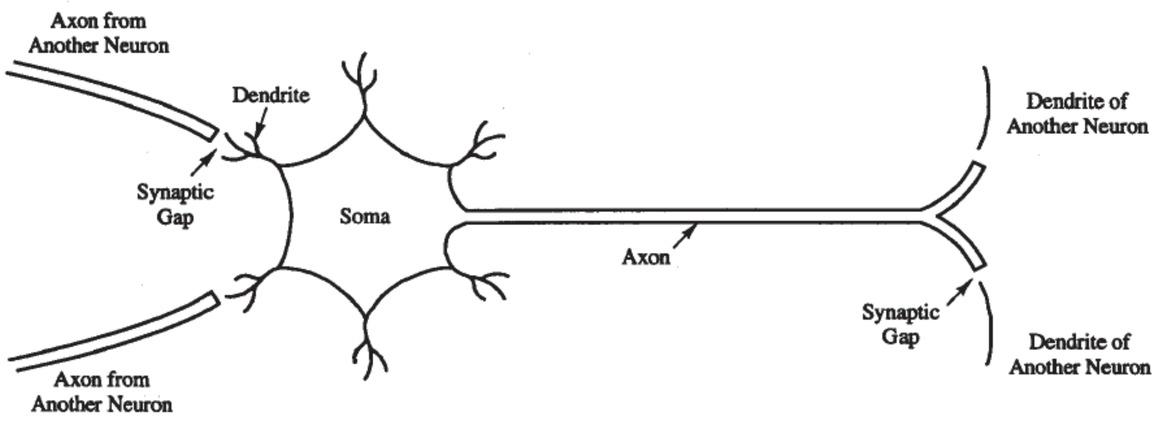
1. The processing element receives many signals.
2. Signals may be modified by a weight at the receiving synapse.
3. The processing element sums the weighted inputs.
4. Under appropriate circumstances (sufficient input), the neuron transmits a single output.
5. The output from a particular neuron may go to many other neurons (the axon branches).

Other features of artificial neural networks that are suggested by biological neurons are:

6. Information processing is local (although other means of transmission, such as the action of hormones, may suggest means of overall process control).
7. Memory is distributed:
   a. Long-term memory resides in the neurons' synapses or weights.
   b. Short-term memory corresponds to the signals sent by the neurons.
8. A synapse's strength may be modified by experience.
9. Neurotransmitters for synapses may be excitatory or inhibitory.

Yet another important characteristic that artificial neural networks share with biological neural systems is *fault tolerance*. Biological neural systems are fault tolerant in two respects. First, we are able to recognize many input signals that are somewhat different from any signal we have seen before. An example of this is our ability to recognize a person in a picture we have not seen before or to recognize a person after a long period of time.

Second, we are able to tolerate damage to the neural system itself. Humans are born with as many as 100 billion neurons. Most of these are in the brain, and most are not replaced when they die [Johnson & Brown, 1988]. In spite of our continuous loss of neurons, we continue to learn. Even in cases of traumatic neural



Biological neuron.

loss, other neurons can sometimes be trained to take over the functions of the damaged cells. In a similar manner, artificial neural networks can be designed to be insensitive to small damage to the network, and the network can be retrained in cases of significant damage (e.g., loss of data and some connections).

WHERE ARE NEURAL NETS BEING USED?

1- Signal processing

2- Control

3- Pattern recognition

4- Medicine

5- Speech recognition

6- Business

Soft computing

3rd class\ 2nd course\ 8th lecture
Computer Science Department\ College of Science for Women

Dr. Noor A. Ibraheem

FUZZY LOGIC FUNDAMENTALS

Introduction

The past few years have witnessed a rapid growth in the number and variety of applications of fuzzy logic (FL). FL techniques have been used in image-understanding applications such as detection of edges, feature extraction, classification, and clustering. Fuzzy logic poses the ability to mimic the human mind to effectively employ modes of reasoning that are approximate rather than exact. In traditional hard computing, decisions or actions are based on precision, certainty, and vigor. Precision and certainty carry a cost. In soft computing, tolerance and impression are explored in decision making. The exploration of the tolerance for imprecision and uncertainty underlies the remarkable human ability to understand distorted speech, decipher sloppy handwriting, comprehend nuances of natural language, summarize text, and recognize and classify images. With FL, we can specify mapping rules in terms of words rather than numbers.

Another basic concept in FL is the fuzzy if–then rule. Although rule-based systems have a long history of use in artificial intelligence, what is missing in such systems is machinery for dealing with fuzzy consequents or fuzzy antecedents. In most applications, an FL solution is a translation of a human solution.

Recently, many intelligent systems called neuro fuzzy systems have been used. There are many ways to combine neural networks and FL techniques. we will introduce FL concepts such as fuzzy sets and their properties, FL operators, hedges, fuzzy proposition and rule-based systems, fuzzy maps and inference engine, defuzzification methods, and the design of an FL decision system.

Fuzzy Sets and Membership Functions

Zadeh introduced the term fuzzy logic in his seminal work “Fuzzy sets,” which described the mathematics of fuzzy set theory (1965). Plato laid the foundation for what would become fuzzy logic, indicating that there was a third region beyond True and False. Fuzzy techniques in the form of approximate reasoning provide decision support and expert systems with powerful reasoning capabilities.

The permissiveness of fuzziness in the human thought process suggests that much of the logic behind thought processing is not traditional two valued logic or even multivalued logic, but logic with fuzzy truths, fuzzy connectiveness, and fuzzy rules of inference. A fuzzy set is an extension of a crisp set. Crisp sets allow only full membership or no membership at all, whereas fuzzy sets allow partial membership. In a crisp set, membership or nonmembership of element x in set A is described by a characteristic function

$\mu_A(x)$, where $\mu_A(x) = 1$ if $x \in A$ and $\mu_A(x) = 0$ if $x \notin A$.

Fuzzy set theory extends this concept by defining partial membership. A fuzzy set A on a universe of discourse U is characterized by a membership function

$\mu_A(x)$

that takes values in the interval . Fuzzy sets represent commonsense linguistic labels like slow, fast, small, large, heavy, low, medium, high, tall, etc. A given element can be a member of more than one fuzzy set at a time.

A fuzzy set A in U may be represented as a set of ordered pairs. Each pair consists of a generic element x and its grade of membership function;

that is, i $A = \{(x, \mu_A(x)) \mid x \in U\}$, x is called a support value if $\mu_A(x) > 0$.

A linguistic variable x in the universe of discourse U is characterized by

$T(x) = \{T_x^1, T_x^2, \ldots, T_x^k\}$ and $u(x) = \{\mu_x^1, \mu_x^2, \ldots, \mu_x^k\}$

where is the term set of x — that is, the set of names of linguistic values of x, with each T_x^i being a fuzzy number with membership function μ_x i defined on U. For example, if x indicates height, then may refer to sets such as short, medium, or tall.

A membership function is essentially a curve that defines how each point in the input space is mapped to a membership value (or degree of membership) between 0 and 1. As an example, consider a fuzzy set tall. Let the universe of discourse be heights from 40 inches to 90 inches. With a crisp set, all people with height 72 or more inches are considered tall, and all people with height of less than 72 inches are considered not tall. The crisp set membership function for set tall is shown in Figure 1. The corresponding fuzzy set with a smooth membership function is shown in Figure 2. The curve defines the transition from not tall and shows the degree of membership for a given height.

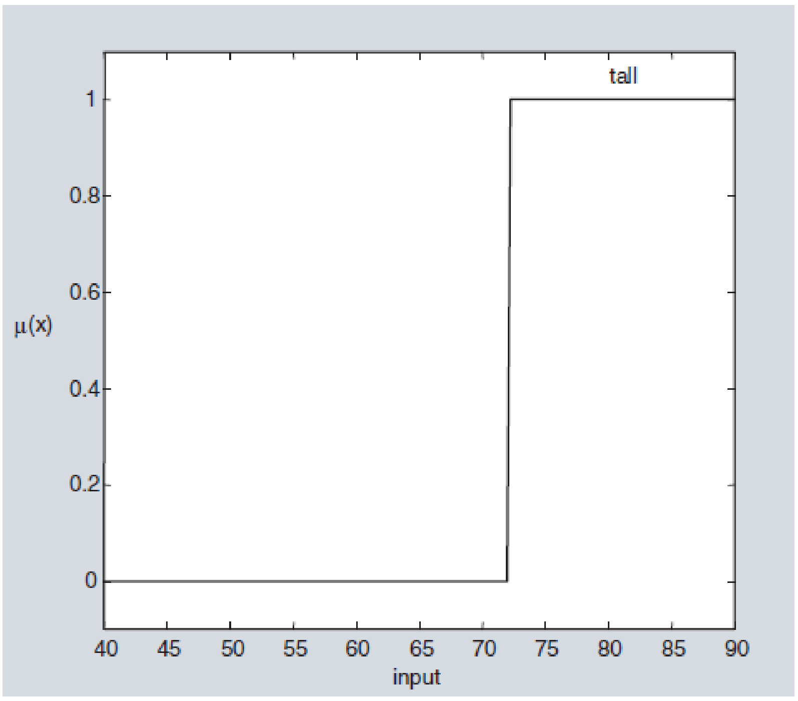


Figure 1 Crisp membership function.

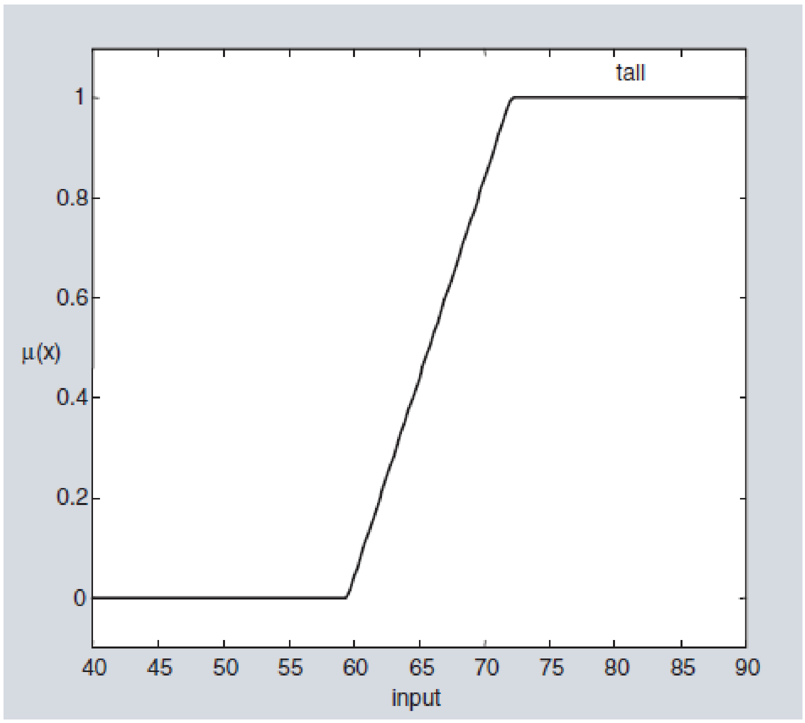


Figure 2 An example of a fuzzy membership function.

Logical Operations and If–Then Rules

Fuzzy set operations are analogous to crisp set operations. The important thing in defining fuzzy set logical operators is that if we keep fuzzy values to the extremes 1 (True) or 0 (False), the standard logical operations should hold. In order to define fuzzy set logical operators, let us first consider crisp set operators. The most elementary crisp set operations are union, intersection, and complement, which essentially correspond to OR, AND, and NOT operators, respectively.

Let A and B be two subsets of U. The union of A and B, denoted $A \cup B$, contains all elements in either A or B; that is, $\mu_{A\cup B}(x) = 1$ if $x \in A$ or $x \in B$.

intersection of A and B, denoted $A \cap B$, contains all the elements that are simultaneously in A and B $\mu_{A\cap B}(x) = 1$ if $x \in A$ and $x \in B$.

The complement of A is denoted by , and it contains all elements that are not in A; that is $\mu_{\bar{A}}(x) = 1$ if $x \notin A$, and $\mu_{\bar{A}}(x) = 0$ if $x \in A$.

The truth tables for these operators are shown in Figure 3.

AND			OR			NOT	
A	B	A∪B	A	B	A∪B	A	$\bar{A}$
0	0	0	0	0	0	0	1
0	1	0	0	1	1	1	0
1	0	0	1	0	1		
1	1	1	1	1	1		

Figure 3 Truth tables for AND, OR, and NOT operators.

Fuzzy Inference System

A fuzzy inference system (FIS) essentially defines a nonlinear mapping of the input data vector into a scalar output, using fuzzy rules. The mapping process involves input/output membership functions, FL operators, fuzzy if–then rules, aggregation of output sets, and defuzzification. An FIS with multiple outputs can be considered as a collection of independent multi-input, single-output systems. A general model of a fuzzy inference system (FIS) is shown in Figure 4.

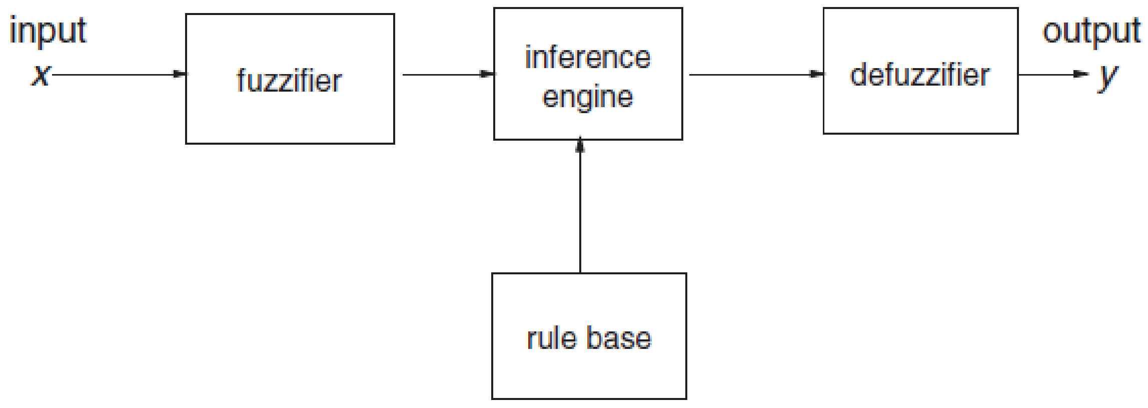


Figure 4. Block diagram of a fuzzy inference system.

The FLS maps crisp inputs into crisp outputs. It can be seen from the figure that the FIS contains four components: the fuzzifier, inference engine, rule base, and defuzzifier. The rule base contains linguistic rules that are provided by experts. It is also possible to extract rules from numeric data. Once the rules have been established, the FIS can be viewed as a system that maps an input vector to an output vector. The fuzzifier maps input numbers into corresponding fuzzy memberships. This is required in order to activate rules that are in terms of linguistic variables. The fuzzifier takes input values and determines the degree to which they belong to each of the fuzzy sets via membership functions. The inference engine defines mapping from input fuzzy sets into output fuzzy sets. It determines the degree to which the antecedent is satisfied for each rule. If the antecedent of a given rule has more than one clause, fuzzy operators are applied to obtain one number that represents the result of the antecedent for that rule.

It is possible that one or more rules may fire at the same time. Outputs for all rules are then aggregated. During aggregation, fuzzy sets that represent the output of each rule are combined into a single fuzzy set. Fuzzy rules are fired in parallel, which is one of the important aspects of an FIS. In an FIS, the order in which rules are fired does not affect the output. The defuzzifier maps output fuzzy sets into a crisp number. Given a fuzzy set that encompasses a range of output values, the defuzzifier returns one number, thereby moving from a fuzzy set to a crisp number. Several methods for defuzzification are used in practice, including the centroid, maximum, mean of maxima, height, and modified height defuzzifier. The most popular defuzzification method is the centroid, which calculates and returns the center of gravity of the aggregated fuzzy set. FISs employ rules. However, unlike rules in conventional expert systems, a fuzzy rule localizes a region of space along the function surface instead of isolating a point on the surface. For a given input, more than one rule may fire. Also, in an FIS, multiple regions are combined in the output space to produce a composite region. A general schematic of an FIS is shown in Figure 5.

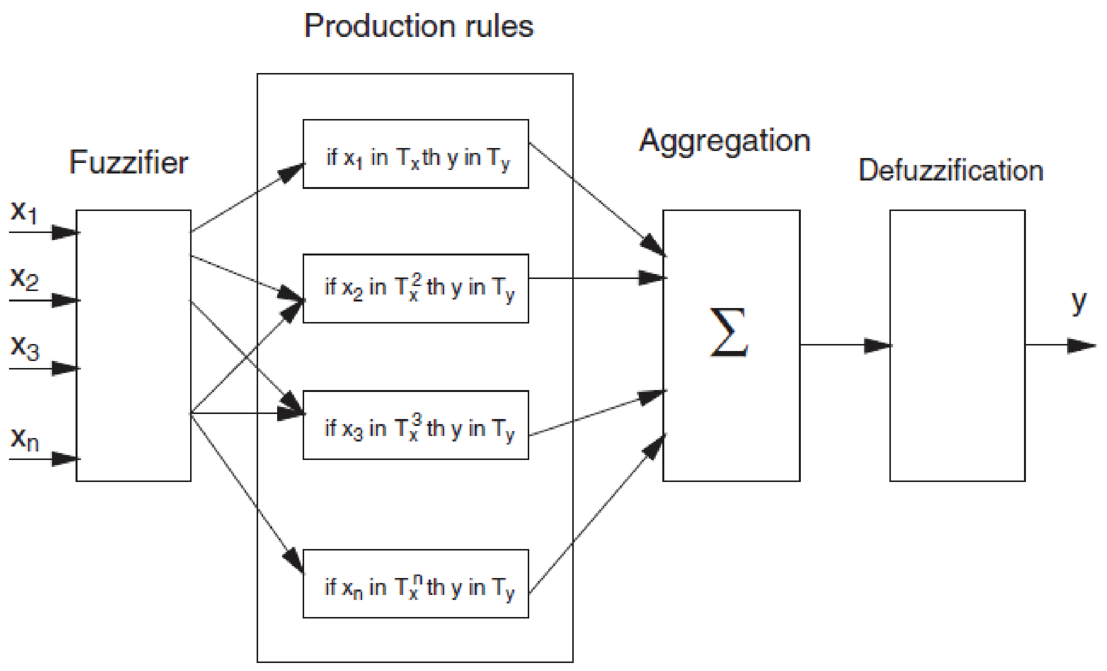

Production rules
Fuzzifier
x_1
x_2
x_3
x_n
if x_1 in T_x th y in T_y
if x_2 in T_x^2 th y in T_y
if x_3 in T_x^3 th y in T_y
if x_n in T_x^n th y in T_y
Aggregation
Σ
Defuzzification
y

Figure 5. Schematic diagram of a fuzzy inference system.

The inference process can be described completely in the five steps shown in Figure 6.

Step 1: Fuzzy Inputs

The first step is to take inputs and determine the degree to which they belong to each of the appropriate fuzzy sets via membership functions.

Step 2: Apply Fuzzy Operators

Once the inputs have been fuzzified, we know the degree to which each part of the antecedent has been satisfied for each rule. If a given rule has more than one part, the fuzzy logical operators are applied to evaluate the composite firing strength of the rule.

Step 3: Apply the Implication Method

The implication method is defined as the shaping of the output membership functions on the basis of the firing strength of the rule. The input for the implication process is a single number given by the antecedent, and the output is a fuzzy set. Two commonly used methods of implication are the minimum and the product.

Step 4: Aggregate all Outputs

Aggregation is a process whereby the outputs of each rule are unified. Aggregation occurs only once for each output variable. The input to the aggregation process is the truncated output fuzzy sets returned by the implication process for each rule. The output of the aggregation process is the combined output fuzzy set.

Step 5: Defuzzify

The input for the defuzzification process is a fuzzy set (the aggregated output fuzzy set), and the output of the defuzzification process is a crisp value obtained by using some defuzzification method such as the centroid, height, or maximum.

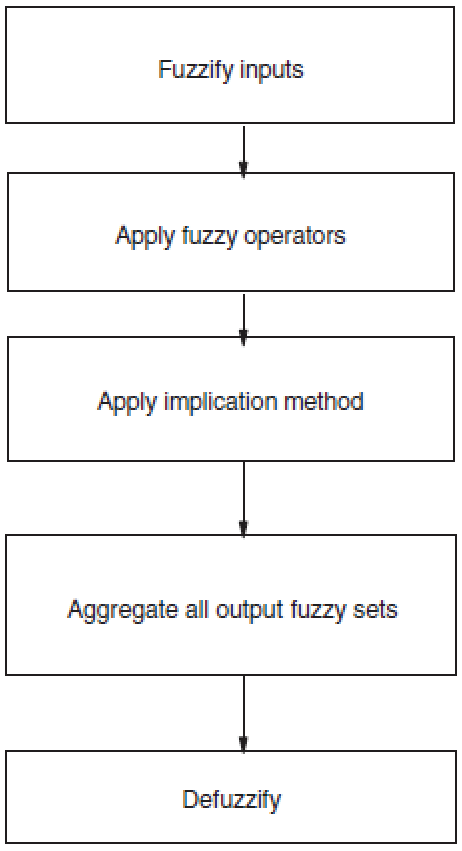


Figure 6. Fuzzy inference process.

Defuzzification

A fuzzy inference system maps an input vector to a crisp output value. In order to obtain a crisp output, we need a defuzzification process. The input to the defuzzification process is a fuzzy set (the aggregated output fuzzy set), and the output of the defuzzification process is a single number.The main methods may be described as follows:

(a) Centroid defuzzification method

(b) Maximum-decomposition method

(c) Center of maxima

(d) Height defuzzification

Summary

Fuzzy set allows partial memberships. Fuzzy sets represent linguistic labels or term sets such as slow, fast, low, medium, high, etc. Fuzzy membership functions represent term sets. Commonly used membership functions are triangular, trapezoidal, bell shaped, and Gaussian curves. In fuzzy logic, the truth of any statement is a matter of degree. In fuzzy logic, operators such as AND, OR, and NOT are implemented by intersection, union, and complement operators. There are various ways to define these operators.

Commonly, AND, OR, and NOT operators are implemented by the min, max, and complement operators. A fuzzy inference system (FIS) maps crisp inputs to crisp outputs. An FIS consists of four components: the fuzzifier, inference engine, rule base, and defuzzifier. The fuzzifier maps input numbers into corresponding fuzzy membership values. The inference engine defines mapping from input fuzzy sets to output fuzzy sets. The defuzzifier maps the output fuzzy sets into a crisp number. The commonly used defuzzification method is the centriod method.