Module - 4 (Fuzzy Inference System & Genetic Algorithm)

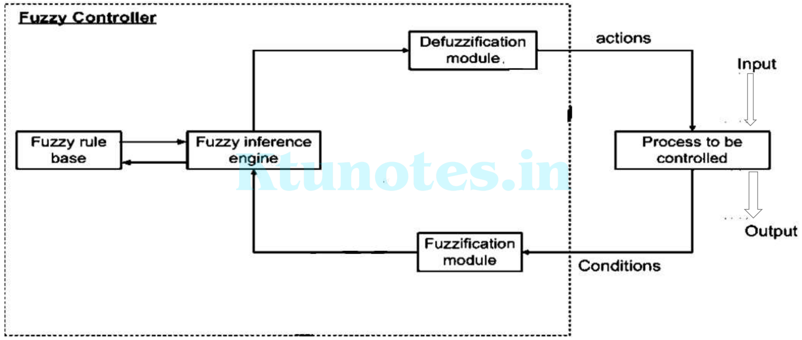
- Fuzzy Inference Systems –
  - Mamdani and Sugeno types.
- Fuzzy Logic Controller.
- Concepts of genetic algorithm.
- Operators in genetic algorithm
  - coding,
  - selection,
  - cross over,
  - mutation.
- Stopping condition for genetic algorithm.

Fuzzy Logic Controller

Fuzzy Logic Controller

- Fuzzy logic controllers (FLC) are the most active research area in the application of fuzzy set theory, fuzzy reasoning, and fuzzy logic
- FLC are special expert systems, which employ a knowledge base expressed in terms of fuzzy inference rules, and a fuzzy inference engine to solve a problem.
- Fuzzy logic control (FLC) techniques usually decompose a complex system into several subsystems according to the human experts' knowledge about the system.
- It utilizes the prior experience of the functionary about the system to be controlled. The main role of the functionary is to set up decision-based rules by analyzing the system behavior and the linguistic input variables within the framework of the system.

Fuzzy Logic Controller



Fuzzy Logic Controller

- The basic architecture of a fuzzy logic controller is shown in Figure .
- The principal components of an FLC system are :
  - Fuzzifier,
  - Fuzzy rule base and a fuzzy knowledge base,
  - Fuzzy inference engine, and
  - Defuzzifier.
- The fuzzifier present converts crisp quantities into fuzzy quantities.
- In the fuzzification stage, the input variable is transformed into a linguistic variable with the help of predefined membership functions (MFs).
- The output of the fuzzification stage is then used to generate the fuzzified output according to the rules set defined

Fuzzy Logic Controller

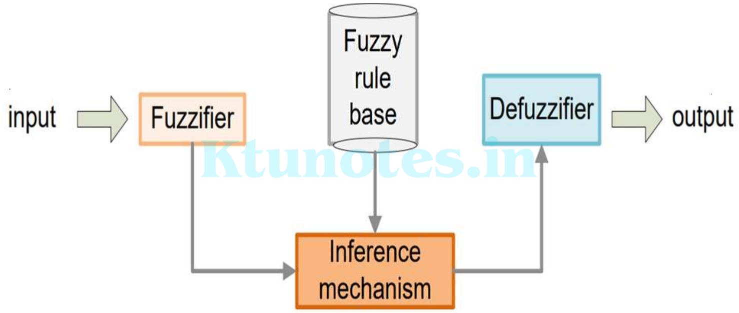
- The fuzzy rule base stores knowledge about the operation of the process of domain expertise.
- The fuzzy knowledge base stores the knowledge about all the input-output fuzzy relationships. It includes the membership functions defining the input variables to the fuzzy rule base and the out variables to the system under control.
- The inference engine is the kernel of an FLC system, and it possesses the capability to simulate human decisions by performing approximate reasoning to achieve the desired control strategy.
- The defuzzifier converts the fuzzy quantities into crisp quantities from the inference engine's inferred fuzzy control action.

Applications of Fuzzy Logic Controller

- Traffic Control
- Steam Engine
- Aircraft Flight Control
- Missile Control
- Liquid-Level Control
- Helicopter Model
- Automobile Speed Controller
- Braking System Controller
- Process Control
- Robotic Control
- Elevator (Automatic Lift) control;
- Automatic Running Control
- Cooling Plant Control
- Water Treatment
- Boiler Control;
- Nuclear Reactor Control;
- Power Systems Control;
- Air Conditioner Control (Temperature Controller)
- Biological Processes
- Washing Machine
- Fault Detection Control Unit

Fuzzy Inference Systems

Fuzzy Inference Systems



Fuzzy Inference Systems

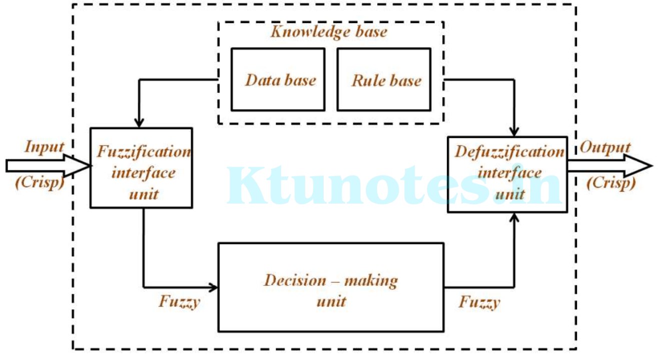


Figure Block diagram of FIS

Working of Fuzzy Inference Systems

- The following five functional blocks will help you understand the construction of FIS –
- Rule Base – It contains numerous fuzzy IF-THEN rules.
- Database – It defines the membership functions of fuzzy sets used in fuzzy rules.
- Decision-making Unit – It performs operations on rules.
- Fuzzification Unit – It converts the crisp quantities into fuzzy quantities.
- Defuzzification Unit – It converts the fuzzy quantities into crisp quantities

Two approaches of Fuzzy Inference Systems

- **Fuzzy Inference System (FIS)** is a process to interpret the values of the input vector and, on the basis of some sets of fuzzy rules, it assigns corresponding values to the output vector.
- This is a method to map an input to an output using fuzzy logic.
- Based on this mapping process, the system takes decisions and distinguishes patterns.
- Following are the two important methods of FIS, having different consequent of fuzzy rules –
  - Mamdani Fuzzy Inference System(1975)
  - Takagi-Sugeno Fuzzy Model (TS Method)(1985)

Mamdani Fuzzy Inference System

- The Mamdani fuzzy inference system was proposed by Ebhasim Mamdani.
- Firstly it was designed to control a steam engine and boiler combination by a set of linguistic control rules obtained from experienced human operators.
- In Mamdani inference system, the output of each rule is a fuzzy logic set.
- Since Mamdani systems have more intuitive and easier to understand rule bases, they are well-suited to expert system applications where the rules are created from human expert knowledge, such as medical diagnostics.

Mamdani Fuzzy Inference System-Steps

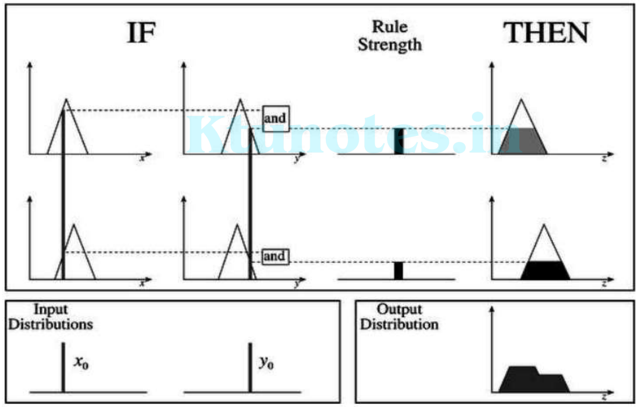
- Following steps need to be followed to compute the output from this FIS –
- **Step 1** – A set of fuzzy rules need to be determined in this step.
- **Step 2** – In this step, by using the input membership function, the input would be made fuzzy.
- **Step 3** – Now establish the rule strength by combining the fuzzified inputs according to fuzzy rules.
- **Step 4** – In this step, determine the consequence of the rule by combining the rule strength and the output membership function.
- **Step 5** – For getting output distribution combine all the consequents.
- **Step 6** – Finally, a defuzzified output distribution is obtained.

Mamdani Fuzzy Inference System-Steps

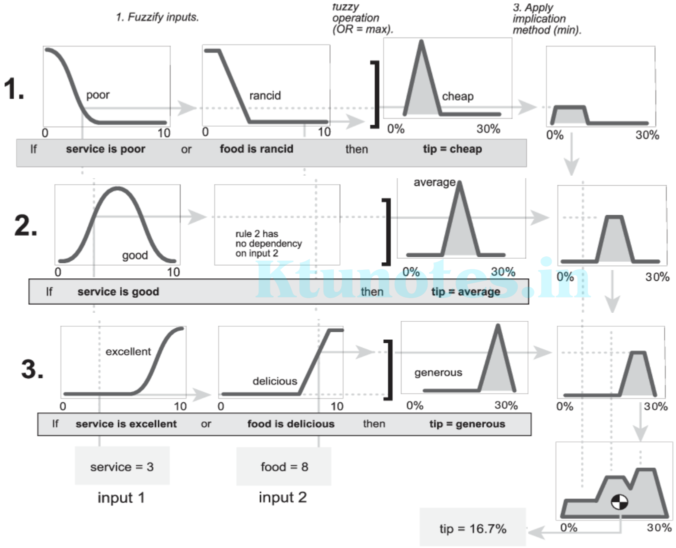
- The fuzzy rules are formed using ” IF – THEN ” statements and ” AND/OR ” connectives.
- The consequence of the rule can be obtained in two steps:
  1. by computing the rule strength completely using the fuzzified inputs from the fuzzy combination.
  2. by clipping the output membership function at the rule strength.
- The output of all the fuzzy rules are combined to obtain one fuzzy output distribution

We examine a simple two-input one-output problem that includes three rules:

Rule: 1	IF x is A3	OR	y is B1	THEN	z is C1
Rule: 2	IF x is A2	AND	y is B2	THEN	z is C2
Rule: 3	IF x is A1			THEN	z is C3



- Third case not considered in the example


1. Fuzzify inputs.
tuzzy operation (OR = max).
3. Apply implication method (min).
1.
poor
rancid
cheap
0
10
0%
30%
If service is poor or food is rancid then tip = cheap
2.
good
rule 2 has no dependency on input 2
average
If service is good then tip = average
3.
excellent
delicious
generous
If service is excellent or food is delicious then tip = generous
4. Apply aggregation method (max).
service = 3
input 1
food = 8
input 2
tip = 16.7%
5. Defuzzify (centroid).

Linguistic variables

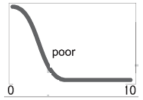
SERVICE →poor,good,excellent

FOOD→rancid,delicious

TIP→cheap,average,generous

P: Service is poor

T(P) = 0.0	: Absolutely false
T(P) = 0.2	: Partially false
T(P) = 0.4	: May be false or not false
T(P) = 0.6	: May be true or not true
T(P) = 0.8	: Partially true
T(P) = 1.0	: Absolutely true.



Mamdani Fuzzy Inference System-Example

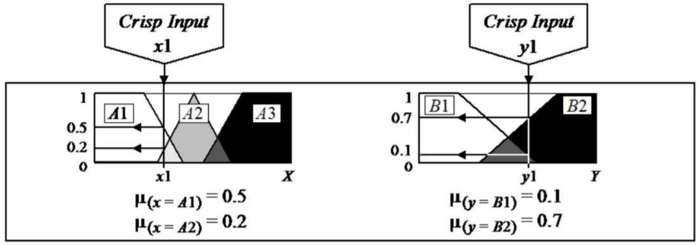
- The inference process of a Mamdani system is described in Fuzzy Inference Process and summarized in the figure.
- The output of each rule is a fuzzy set derived from the output membership function and the implication method of the FIS.
- These output fuzzy sets are combined into a single fuzzy set using the aggregation method of the FIS.
- Then, to compute a final crisp output value, the combined output fuzzy set is defuzzified using one of the methods described in Defuzzification Methods.

Real-life example for these kinds of rules:

Rule: 1	IF project_funding is adequate	OR	project_staffing is small	THEN	risk is low
Rule: 2	IF project_funding is marginal	AND	project_staffing is large	THEN	risk is normal
Rule: 3	IF project_funding is inadequate			THEN	risk is high

Step 1: Fuzzification

The first step is to take the crisp inputs, x1 and y1 (*project funding* and *project staffing*), and determine the degree to which these inputs belong to each of the appropriate fuzzy sets.



Step 2: Rule Evaluation

- The second step is to take the fuzzified inputs, $\mu_{(x=A1)} = 0.5$, $\mu_{(x=A2)} = 0.2$, $\mu_{(y=B1)} = 0.1$ and $\mu_{(y=B2)} = 0.7$, and apply them to the antecedents of the fuzzy rules.
- If a given fuzzy rule has multiple antecedents, the fuzzy operator (AND or OR) is used to obtain a single number that represents the result of the antecedent evaluation.

RECALL: To evaluate the disjunction of the rule antecedents, we use the **OR** fuzzy operation. Typically, fuzzy expert systems make use of the classical fuzzy operation union:

$$\mu_{A\cup B}(x) = \max\,[\mu_A(x),\ \mu_B(x)]$$

Similarly, in order to evaluate the conjunction of the rule antecedents, we apply the **AND** fuzzy operation intersection:

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

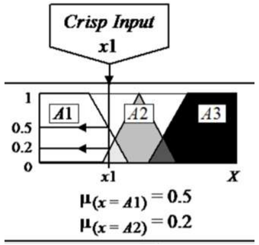
Rule: 1	IF x is A3	OR	y is B1	THEN	z is C1
Rule: 2	IF x is A2	AND	y is B2	THEN	z is C2
Rule: 3	IF x is A1			THEN	z is C3

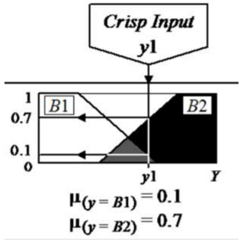
first step is to take the crisp inputs, x1 and y1 (*project funding* and *project staffing*),

Rule: 1 IF project_funding is adequate OR project_staffing is small THEN risk is low

Rule: 2 IF project_funding is marginal AND project_staffing is large THEN risk is normal

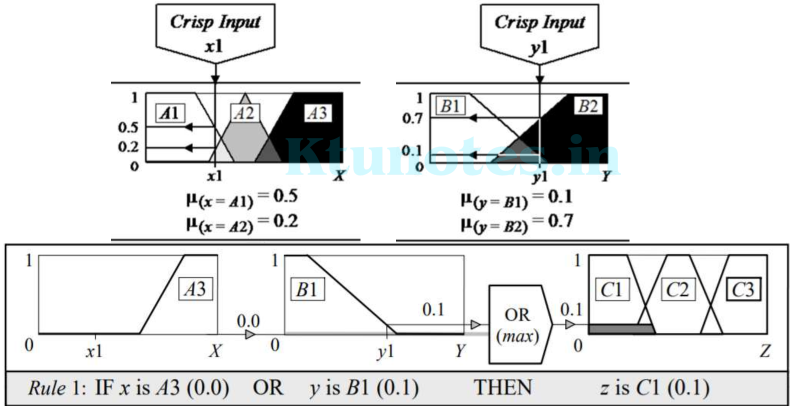
Rule: 3 IF project_funding is inadequate THEN risk is high





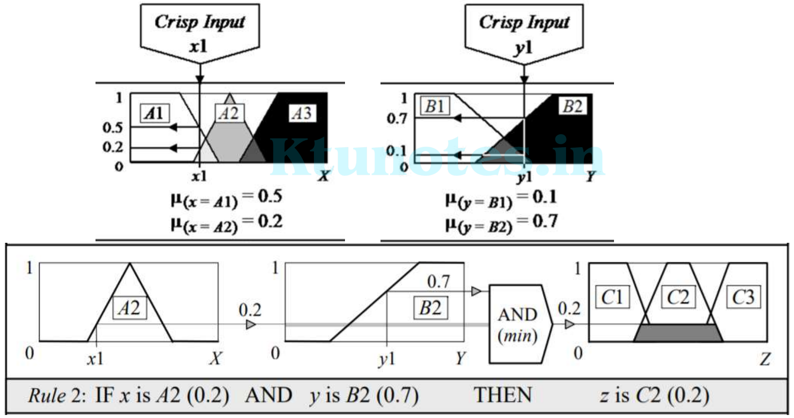
- A1→Inadequate
- A2→Marginal
- A3→Adequate
- B1→Small
- B2→Large
- C1→Low
- C2→Normal
- C3→High

Rule: 1	IF x is A3	OR	y is B1	THEN	z is C1
Rule: 2	IF x is A2	AND	y is B2	THEN	z is C2
Rule: 3	IF x is A1			THEN	z is C3



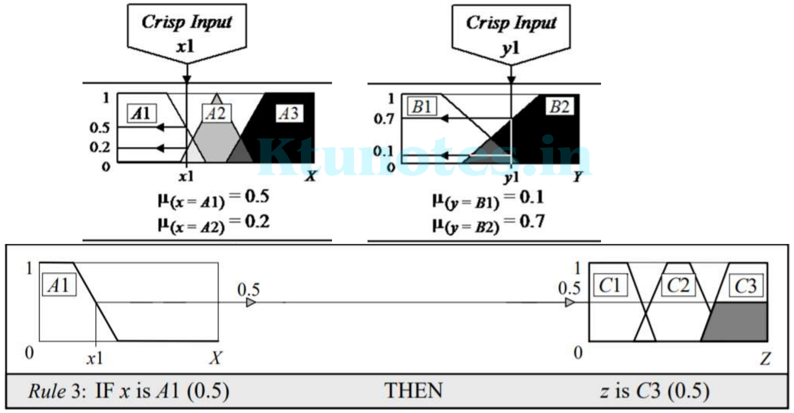
Rule 1: IF *x* is *A*3 (0.0) OR *y* is *B*1 (0.1) THEN *z* is *C*1 (0.1)

Rule: 1	IF x is A3	OR	y is B1	THEN	z is C1
Rule: 2	IF x is A2	AND	y is B2	THEN	z is C2
Rule: 3	IF x is A1			THEN	z is C3



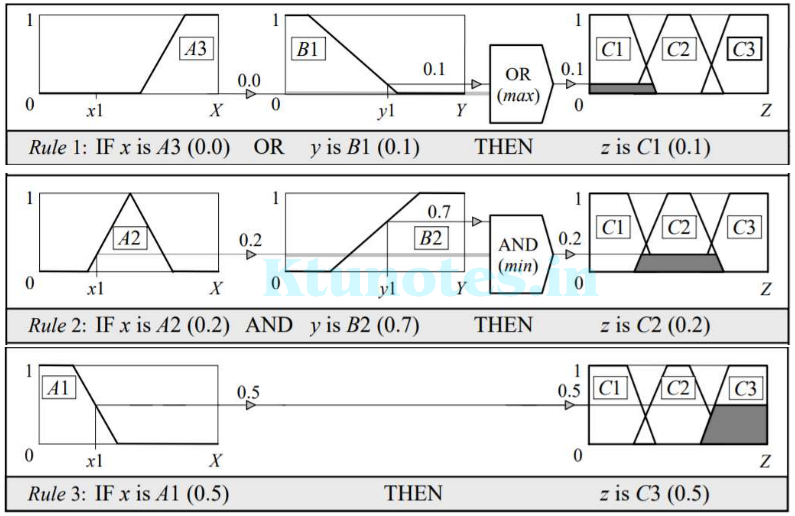
Rule 2: IF *x* is *A*2 (0.2) AND *y* is *B*2 (0.7) THEN *z* is *C*2 (0.2)

Rule: 1 IF x is A3 OR y is B1 THEN z is C1
Rule: 2 IF x is A2 AND y is B2 THEN z is C2
Rule: 3 IF x is A1 THEN z is C3



$\mu_{(x=A1)} = 0.5$
$\mu_{(x=A2)} = 0.2$
$\mu_{(y=B1)} = 0.1$
$\mu_{(y=B2)} = 0.7$

Rule 3: IF *x* is *A*1 (0.5) THEN *z* is *C*3 (0.5)



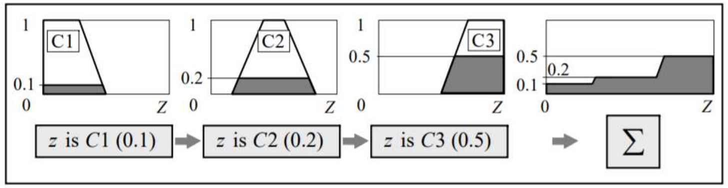
Rule 1: IF *x* is *A*3 (0.0) OR *y* is *B*1 (0.1) THEN *z* is *C*1 (0.1)

Rule 2: IF *x* is *A*2 (0.2) AND *y* is *B*2 (0.7) THEN *z* is *C*2 (0.2)

Rule 3: IF *x* is *A*1 (0.5) THEN *z* is *C*3 (0.5)

Step 3: Aggregation of the Rule Outputs

- Aggregation is the process of unification of the outputs of all rules.
- We take the membership functions of all rule consequents previously clipped or scaled and combine them into a single fuzzy set.
- The input of the aggregation process is the list of clipped or scaled consequent membership functions, and the output is one fuzzy set for each output variable.



z is *C*1 (0.1) → *z* is *C*2 (0.2) → *z* is *C*3 (0.5) →

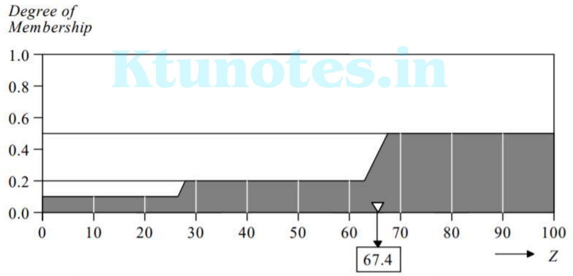
Step 4: Defuzzification

- The last step in the fuzzy inference process is defuzzification.
- Fuzziness helps us to evaluate the rules, but the final output of a fuzzy system has to be a crisp number.
- The input for the defuzzification process is the aggregate output fuzzy set and the output is a single number.
- There are several defuzzification methods, but probably the most popular one is the **centroid technique**. It finds the point where a vertical line would slice the aggregate set into two equal masses. Mathematically this **centre of gravity** (**COG**) can be expressed as:

$$COG = \frac{\int_a^b \mu_A(x)\, x\, dx}{\int_a^b \mu_A(x)\, dx}$$

Step 4: Defuzzification

- Centroid defuzzification method finds a point representing the centre of gravity of the aggregated fuzzy set *A*, on the interval [*a*, *b*].
- A reasonable estimate can be obtained by calculating it over a sample of points.



$$COG = \frac{(0+10+20)\times 0.1+(30+40+50+60)\times 0.2+(70+80+90+100)\times 0.5}{0.1+0.1+0.1+0.2+0.2+0.2+0.2+0.5+0.5+0.5+0.5} = 67.4$$

Takagi-Sugeno Fuzzy Model (TS Method)

Takagi-Sugeno Fuzzy Model (TS Method)

- This model was proposed by Takagi, Sugeno, and Kang in 1985.
- Also known as the TSK fuzzy model.
- For developing a systematic approach to generating fuzzy rules from a given input-output data set.
- Sugeno-style fuzzy inference is very similar to the Mamdani method.
- Sugeno changed only a rule consequent: instead of a fuzzy set, he used a mathematical function of the input variable
- The rule format of the Sugeno form is given by

$$IF\ x\ is\ A\ and\ y\ is\ B\ THEN\ z = f(x, y)$$

where A and B are fuzzy sets in the antecedents

$z = f(x, y)$ is a crisp function in the consequent.

Takagi-Sugeno Fuzzy Model (TS Method)

$$IF\ x\ is\ A\ and\ y\ is\ B\ THEN\ z = f(x, y)$$

where A and B are fuzzy sets in the antecedents

$z = f(x, y)$ is a crisp function in the consequent.

- Generally, $f(x, y)$ is a polynomial in the input variables x and y.
- If $z = f(x, y)$ is a first order polynomial, we get *first−order Sugeno fuzzy model.*
- If f is a constant, we get *zero−order Sugeno fuzzy model.*
- The most commonly used **zero-order Sugeno fuzzy model** applies fuzzy rules in the following form:

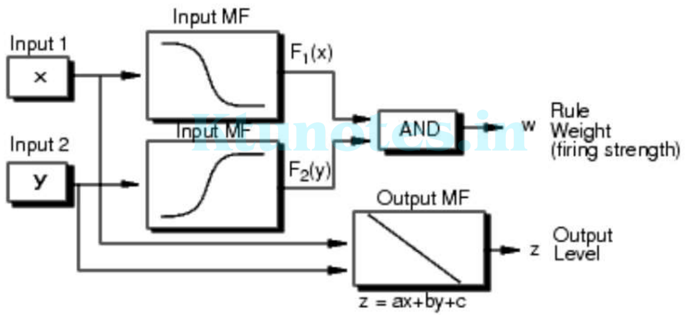
IF	***x* is *A***	**AND**	***y* is *B***	**THEN**	***z* is *k***

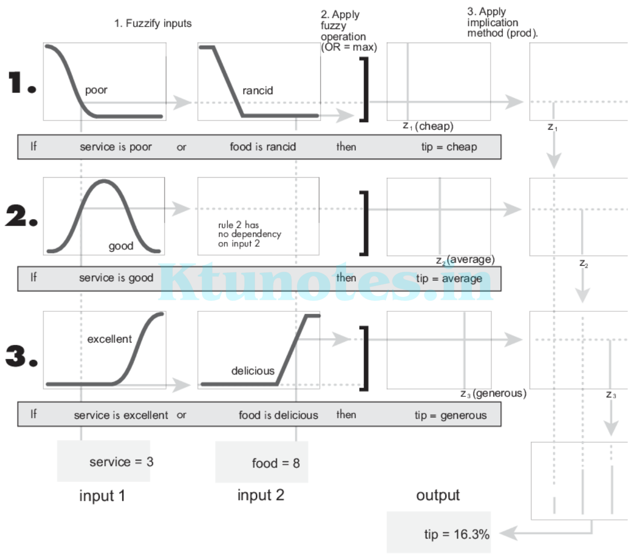
where *k* is a constant.

Takagi-Sugeno Fuzzy Model (TS Method)

- Sugeno fuzzy inference, also referred to as Takagi-Sugeno-Kang fuzzy inference, uses *singleton* output membership functions that are either constant or a linear function of the input values.
- The defuzzification process for a Sugeno system is more computationally efficient compared to that of a Mamdani system, since it uses a weighted average or weighted sum of a few data points rather than compute a centroid of a two-dimensional area.
- Each rule in a Sugeno system operates as shown in the following diagram, which shows a two-input system with input values *x* and *y*.

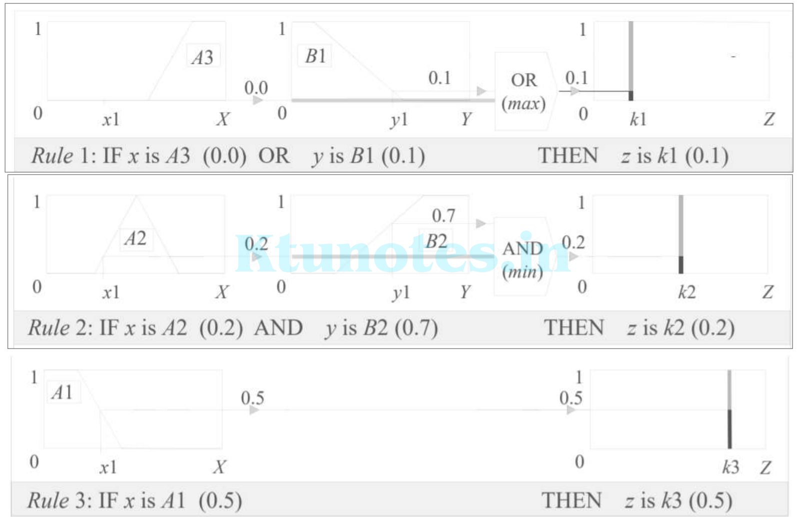
Takagi-Sugeno Fuzzy Model (TS Method)




1. Fuzzify inputs
2. Apply fuzzy operation (OR = max)
3. Apply implication method (prod).
1.
poor
rancid
z1 (cheap)
z1
If service is poor or food is rancid then tip = cheap
2.
good
rule 2 has no dependency on input 2
z2 (average)
z2
If service is good then tip = average
3.
excellent
delicious
z3 (generous)
z3
If service is excellent or food is delicious then tip = generous
service = 3
input 1
food = 8
input 2
output
tip = 16.3%

Takagi-Sugeno Fuzzy Model (TS Method)

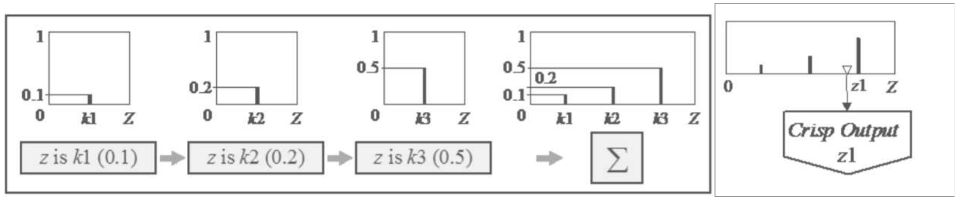
- Each rule generates two values:
- z_i Rule output level, which is either a constant value or a linear function of the input values: $z_i = a_i x + b_i y + c_i$
- Here, *x* and *y* are the values of input 1 and input 2, respectively, and a_i, b_i, and c_i are constant coefficients.
- For a zero-order Sugeno system, z_i is a constant ($a = b = 0$).
- w_i - Rule firing strength derived from the rule antecedent
- $w_i = AndMethod(F_1(x), F_2(y))$
- Here, $F_1(...)$ and $F_2(...)$ are the membership functions for inputs 1 and 2, respectively.
- The output of each rule is the weighted output level, which is the product of w_i and z_i



Rule 1: IF *x* is *A*3 (0.0) OR *y* is *B*1 (0.1) THEN *z* is *k*1 (0.1)

Rule 2: IF *x* is *A*2 (0.2) AND *y* is *B*2 (0.7) THEN *z* is *k*2 (0.2)

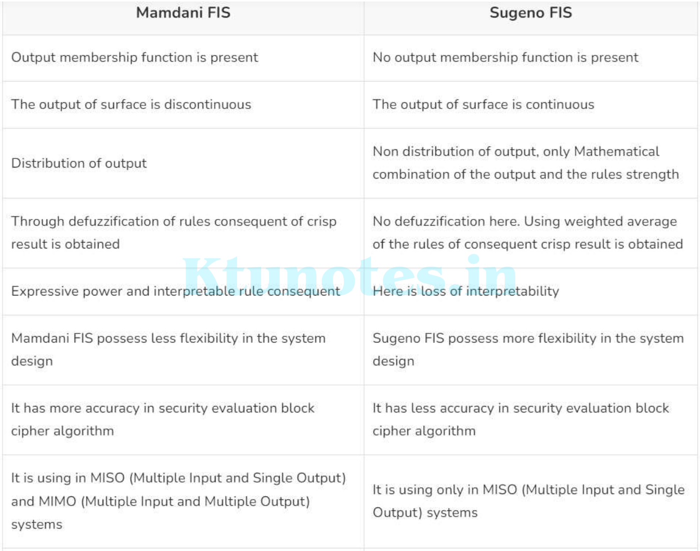
Rule 3: IF *x* is *A*1 (0.5) THEN *z* is *k*3 (0.5)



COG becomes Weighted Average (WA)

$$WA = \frac{\mu(k1)\times k1 + \mu(k2)\times k2 + \mu(k3)\times k3}{\mu(k1)+\mu(k2)+\mu(k3)}$$

$$= \frac{0.1\times 20 + 0.2\times 50 + 0.5\times 80}{0.1+0.2+0.5} = 65$$



Mamdani FIS	Sugeno FIS
Output membership function is present	No output membership function is present
The output of surface is discontinuous	The output of surface is continuous
Distribution of output	Non distribution of output, only Mathematical combination of the output and the rules strength
Through defuzzification of rules consequent of crisp result is obtained	No defuzzification here. Using weighted average of the rules of consequent crisp result is obtained
Expressive power and interpretable rule consequent	Here is loss of interpretability
Mamdani FIS possess less flexibility in the system design	Sugeno FIS possess more flexibility in the system design
It has more accuracy in security evaluation block cipher algorithm	It has less accuracy in security evaluation block cipher algorithm
It is using in MISO (Multiple Input and Single Output) and MIMO (Multiple Input and Multiple Output) systems	It is using only in MISO (Multiple Input and Single Output) systems

GENETIC ALGORITHM

GENETIC ALGORITHM HISTORY

- Genetic algorithms are a part of evolutionary computing,
- It is a rapidly growing area of artificial intelligence.
- Charles Darwin has formulated the fundamental principles of natural selection as the main evolutionary tool.
- genetic algorithms are inspired by Darwin's theory about evolution.
- Gregor Mendel discovered hereditary principles
- Morgan experimentally found that chromosomes were the carriers of hereditary information and genes representing the hereditary factors were lined up on chromosomes
- Combination of Darwin’s and Mendel’s ideas lead to the modern evolutionary theory.

GENETIC ALGORITHM HISTORY

- Genetic Algorithms (GAs) were invented by John Holland and are inspired by Darwin's theory about evolution.
- In 1992 John Koza used a genetic algorithm to evolve programs to perform certain tasks. He called his method "genetic programming" (GP).
- Genetic Algorithm (GA) is a search-based algorithm based on Genetics and Natural Selection principles.
- It is frequently used to find optimal or near-optimal solutions to difficult problems that otherwise would take a lifetime.
- It is frequently used to solve optimization problems, in research, and in machine learning.
- GAs are a subset of a much larger branch of computation known as **Evolutionary Computation**.

INTRODUCTION TO GA

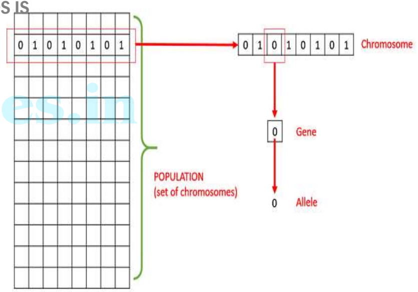
- GAs Algorithm is started with a **pool or a population of possible solutions** to the given problem.
- These solutions then undergo recombination and mutation (like in natural genetics), producing new children, and the process is repeated over various generations.
- Each individual (or candidate solution) is assigned a fitness value (based on its objective function value) and the fitter individuals are given a higher chance to mate and yield more “fitter” individuals. This is in line with the Darwinian Theory of “Survival of the Fittest”.
- In this way we keep “evolving” better individuals or solutions over generations, till we reach a stopping criterion(for example a number of populations or improvement of the best solution) is satisfied.
- So solutions from one population are taken and used to form a new population.

Concepts of genetic algorithm

- **Cells**
  - All living organisms consist of cells. In each cell, there is the same set of **chromosomes**.
- **Chromosome**
  - Chromosomes are strings of DNA and serve as a model for the whole organism
- Genes
  - A chromosome consists of **genes**, and blocks of DNA.
  - Each gene encodes a particular protein.
  - Each gene encodes a **trait**, for example, the colour of the eyes.
- **alleles**
  - Possible settings for a trait (e.g. Dark skin, blue eyes, brown hair.) are called **alleles**.
- Each gene has its own position in the chromosome. This position is called **locus**.
- **genome**
  - Complete genetic information (all chromosomes) is called the **genome**.
  - Particular set of genes in the genome is called **genotype**.

Concepts of genetic algorithm

- The basic building blocks in living bodies are cells.
- Each cell carries the basic unit of heredity, called gene
- For a particular specie, the number of chromosomes is fixed.
- Examples
  - Mosquito:6
  - Frogs: 26
  - Human: 46
  - Goldfish: 94



Genetic algorithm is a population-based probabilistic search and optimization technique, which works based on the mechanisms of natural genetics and natural evaluation.

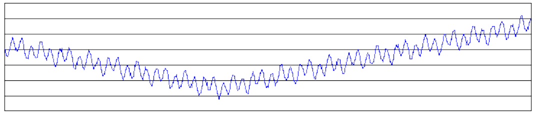
Optimization

- Optimization is the process of **making something better**.
- Optimization refers to finding the values of inputs in such a way that we get the “best” output values.
- The definition of “best” varies from problem to problem, it refers to maximizing or minimizing one or more objective functions, by varying the input parameters.
- The set of all possible solutions or values which the inputs can take make up the search space.
- In this search space, lies a point or a set of points that gives the optimal solution. The aim of **optimization is to find that point or set of points in the search space.**

Set of Inputs
Process
Set of Outputs

Search space

- If we are solving some problem, we are usually looking for some solution, which will be the best among others.
- The space of all feasible solutions is called **search space** (also state space).
- A population of individuals is maintained within the search space for GA, each point in the search space represents one feasible solution to a given problem.
- Each feasible solution can be "marked" by its value or fitness for the problem.
- We are looking for our solution, which is one point (or more) among feasible solutions - that is one point in the search space.



Example of a search space

Search space

- A chromosome (solution) comprises several genes(variables).
- A fitness score is assigned to each solution representing the abilities of an individual to compete.
- The individual with the optimal fitness score is sought.
- The GA aims to use selective breeding of the solutions to produce offspring better than the parents by combining information from the chromosomes.
- GA maintains a population of n chromosomes(solutions) with associated fitness values.
- Parents are selected to mate, on the basis of their fitness, producing offspring via a reproductive plan and offspring inherits characteristics from each parent.
- As parents mate and produce offspring, a room must be made for the new arrivals. Individuals in the population die and are replaced by new solutions, eventually creating a new generation, once all mating opportunities in the old population have been exhausted
- Least-fit solutions would die out and better solutions will thrive.

Outline of Basic Genetic Algorithm

- **[Start]** Generate a random population of *n* chromosomes (suitable solutions for the problem)
- **[Fitness]** Evaluate the fitness *f(x)* of each chromosome *x* in the population
- **[New population]** Create a new population by repeating the following steps until the new population is complete
  - **[Selection]** Select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to be selected)
  - **[Crossover]** With a crossover probability cross over the parents to form a new offspring (children). If no crossover was performed, the offspring is an exact copy of the parents.
  - **[Mutation]** With a mutation probability mutate new offspring at each locus (position in chromosome).
  - **[Accepting]** Place new offspring in a new population
- **[Replace]** Use the newly generated population for a further run of the algorithm
- **[Test]** If the end condition is satisfied, **stop**, and return the best solution in the current population
- **[Loop]** Go to step **2**.

Outline of Basic Genetic Algorithm

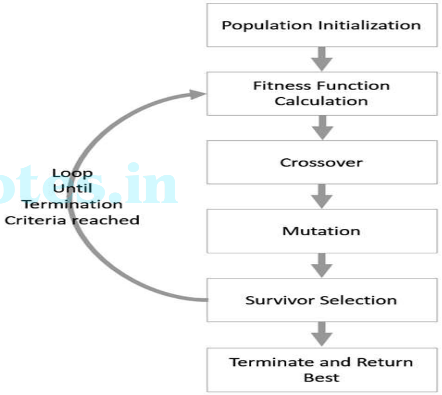
```
function sga ()
        {
        Initialize population;
        Calculate fitness function;

        While(fitness value != termination criteria)
                {
                  Selection;

                  Crossover;

                  Mutation;

                Calculate fitness function;
                }
        }
```


Population Initialization
Fitness Function Calculation
Crossover
Mutation
Survivor Selection
Terminate and Return Best
Loop Until Termination Criteria reached

Encoding in GA

Encoding in Genetic Algorithm

- **Encoding using string :**
  Encoding of chromosomes is the first step in solving the problem and it depends entirely on the problem heavily.
- The process of representing the solution in the form of a string of bits that conveys the necessary information.
- just as in a chromosome, each gene controls particular characteristics of the individual, similarly, each bit in the string represents characteristics of the solution.
- The chromosome should somehow contain information about the solution it represents.
- The most used way of encoding is a binary string.

Encoding in Genetic Algorithm

- In **binary encoding** every chromosome is a string of **bits**, **0** or **1**.

Chromosome A	101100101100101011100101
Chromosome B	111111100000110000011111

- Binary encoding gives many possible chromosomes even with a small number of alleles
- Example of Problem: Knapsack problem
- The problem: There are things with a given value and size. The knapsack has given capacity. Select things to maximize the value of things in knapsack, but do not extend the knapsack capacity.
- Encoding: Each bit says, if the corresponding thing is in knapsack.

Encoding in Genetic Algorithm

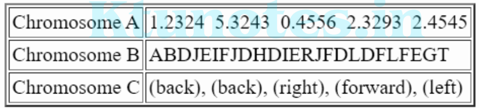
- **Permutation Encoding**
- Permutation encoding can be used in ordering problems, such as travelling salesman or task ordering problems.
- In **permutation encoding**, every chromosome is a string of numbers representing numbers in a **sequence**.

Chromosome A	1 5 3 2 6 4 7 9 8
Chromosome B	8 5 6 7 2 3 1 4 9

- Permutation encoding is only useful for ordering problems.
- **Example of Problem:** Travelling salesman problem (TSP)
  The problem: There are cities and given distances between them.
- Travelling salesman has to visit all of them, but he does not travel very much. Find a sequence of cities to minimize travelled distance.
- **Encoding:** Chromosome says order of cities, in which the salesman will visit them.

Encoding in Genetic Algorithm

- **Value Encoding**
- Used in problems where complicated values, such as real numbers, are used and where binary encoding would not suffice.
- Good for some problems, but often necessary to develop some specific crossover and mutation techniques for these chromosomes.



Chromosome A	1.2324 5.3243 0.4556 2.3293 2.4545
Chromosome B	ABDJEIFJDHDIERJFDLDFLFEGT
Chromosome C	(back), (back), (right), (forward), (left)

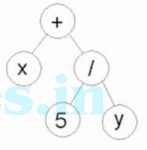
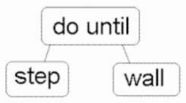
- **Example of Problem:** Finding weights for neural network
  The problem: There is some neural network with given architecture. Find weights for inputs of neurons to train the network for wanted output.
  Encoding: Real values in chromosomes represent corresponding weights for inputs.

Encoding in Genetic Algorithm

- **Tree Encoding**
- Tree encoding is used mainly for evolving programs or expressions, for **genetic programming**.
- In **tree encoding** every chromosome is a tree of some objects, such as functions or commands in programming language.
- Example of chromosomes with tree encoding
- Tree encoding is good for evolving programs. Programing language LISP is often used to this, because programs in it are represented in this form and can be easily parsed as a tree, so the crossover and mutation can be done relatively easily.

Encoding in Genetic Algorithm

- **Tree Encoding**
- Example of Problem: Finding a function from given values
- The problem: Some input and output values are given. Task is to find a function, which will give the best (closest to wanted) output to all inputs.
- Encoding: Chromosome are functions represented in a tree.

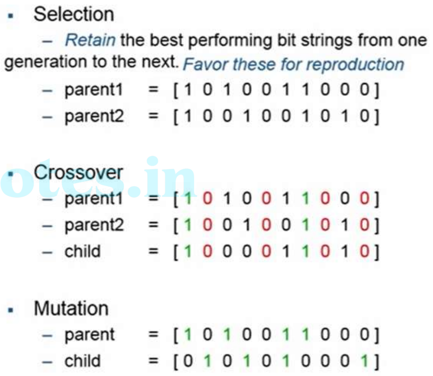
Chromosome A	Chromosome B
	
(+ x (/ 5 y))	(do_until step wall)

Operators of Genetic algorithm

Selection

Operators of Genetic Algorithm

- Selection, is the process of creating the population for the next generation from the current generation.
- Crossover, which represents mating between individuals
- Mutation, which introduces random modification



- Selection
  - *Retain* the best performing bit strings from one generation to the next. *Favor these for reproduction*
  - parent1 = [1 0 1 0 0 1 1 0 0 0]
  - parent2 = [1 0 0 1 0 0 1 0 1 0]
- Crossover
  - parent1 = [1 0 1 0 0 1 1 0 0 0]
  - parent2 = [1 0 0 1 0 0 1 0 1 0]
  - child = [1 0 0 0 0 1 1 0 1 0]
- Mutation
  - parent = [1 0 1 0 0 1 1 0 0 0]
  - child = [0 1 0 1 0 1 0 0 0 1]

OPERATORS IN GA

- Selection, which equates to the survival of the fittest. It is the process of choosing two parents from the population for crossing.
- Selection is the process of creating the population for the next generation from the current generation.
- To generate a new population:
- • Create a mating pool
- • Select a pair
- • Reproduce
- Crossover, which represents mating between individuals, is used to combine the genetic information of two parents to generate new offspring.
- Mutation, which introduces random modification. It changes the values of some genes to increase the quality of new children

Selection operator IN GA

- Selection is the process of choosing two parents from the population for crossing.
- After deciding on an encoding, the next step is to decide how to perform

selection, i.e.,

- how to choose individuals in the population that will create offspring

for the next generation_ and how many offspring each will create.

- The purpose of selection is to emphasize fitter individuals in the population in hopes that their offspring have higher fitness.
- Chromosomes are selected from the initial population to be parents for

reproduction. The problem is how to select these chromosomes.

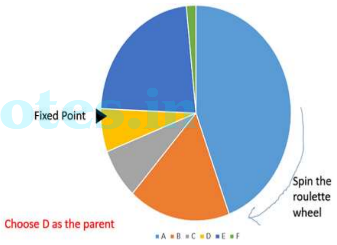
- According to Darwin's theory of evolution the best ones survive to create new offspring

Selection operator IN GA

- The key idea is to give preference to better individuals, allowing them to pass on their genes to the next generation
- The goodness of each individual depends on his fitness
- Fitness may be determined by an objective function or subjective judgment
- Higher the fitness function more the probability an individual would be selected
- The classical selection operator for GA described by Goldberg is a roulette wheel
- Two types of selection schemes
  - Proportionate based selection-it picks out individuals on the basis of their fitness values relative to the fitness of other individuals in the population
  - Ordinal-based selection-it selects individuals based on their rank within the population.

Selection operator IN GA

- 1.1roulette wheel
- Consider a circular wheel. The wheel is divided into **n pies**, where n is the number of individuals in the population.
- Each individual gets a portion of the circle which is proportional to its fitness value.
- In a roulette wheel selection, the circular wheel is divided and a fixed point is chosen on the wheel circumference as shown and the wheel is rotated.



Choose D as the parent

Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

Selection operator IN GA

- 1.1roulette wheel
- The region of the wheel which comes in front of the fixed point is chosen as the parent.
- For the second parent, the same process is repeated.
- It is clear that a fitter individual has a greater pie on the wheel and therefore a greater chance of landing in front of the fixed point when the wheel is rotated. the probability of choosing an individual depends directly on their fitness.
- The member with the greatest fitness has the highest probability of selecting.

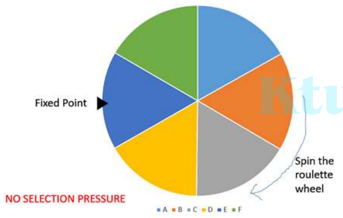
Selection operator IN GA

- 1.2 Linear Rank Selection
- Rank Selection also works with negative fitness values and is mostly used when the individuals in the population have very close fitness values.
- Each individual has an almost equal share of the pie and hence each individual no matter how fit relative to each other has approximately the same probability of getting selected as a parent.
- This leads to a loss in the selection pressure toward fitter individuals, making the GA make poor parent selections in such situations.
- Here we remove the concept of a fitness value to some extend while selecting a parent

Selection operator IN GA

- 1.2 Linear Rank Selection

Chromosome	Fitness Value	Rank
A	8.1	1
B	8.0	4
C	8.05	2
D	7.95	6
E	8.02	3
F	7.99	5



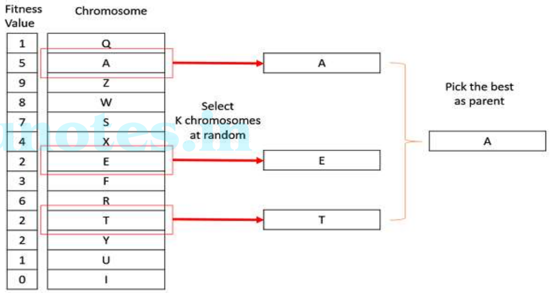
NO SELECTION PRESSURE

Chromosome	Fitness Value
A	8.1
B	8.0
C	8.05
D	7.95
E	8.02
F	7.99

- every individual in the population is ranked according to their fitness and other parameters.
- The selection of the parents depends on the rank of each individual and not their fitness.
- The higher-ranked individuals are preferred more than the lower-ranked ones.

Selection operator IN GA

- 1.3.Tournament selection
- In K-Way tournament selection, we select K individuals from the population at random and select the best out of these to become a parent.
- The same process is repeated for selecting the next parent.
- Tournament Selection is also extremely popular in literature as it can even work with negative fitness values.



Fitness Value	Chromosome
1	Q
5	A
9	Z
8	W
7	S
4	X
2	E
3	F
6	R
2	T
2	Y
1	U
0	I

Cross over

crossover operator IN GA

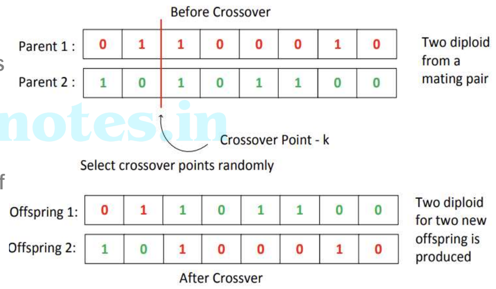
- A mating pair (each pair consists of two strings) is selected at random.
- Thus, if the size of the mating pool is N, then N/2 mating pairs are to be formed. [Random Mating]
- The pairs are checked, for whether they will participate in reproduction or not.
- Once, a pool of mating pair are selected, they undergo crossover operations.
- In the crossover, there is an exchange of properties between two parents, and as a result of which two offspring solutions are produced.
- The crossover point(s) (also called k-point(s)) is(are) decided using a random number generator generating integer(s) in between 1 and L, where L is the length of the chromosome.
- Then we perform the exchange of gene values with respect to the k-point(s)

crossover operator IN GA

- Two individuals are chosen from the population using a selection operator.
- Crossover is the exchange of genes between the chromosomes of the two parents
- Crossover means the method of merging the genetic information of two individuals.
- If the coding is chosen properly, two good parents produce good children
- Chromosomes are randomly split and merged, with the consequence that some gene of a child comes from one parent while others come from their parent. This mechanism is called crossover
- Types of crossover
  - Single-point crossover
  - Two-point crossover
  - Multi-point crossover
  - Uniform crossover
  - Shuffle crossover

crossover operator IN GA

- **Single-point crossover**
- Here, we select the K-point lying between 1 and L. Let it be k.
- A single crossover point at k on both parent's strings is selected.
- All data beyond that point in either string is swapped between the two parents.
- The resulting strings are the chromosomes of the offspring produced.



Before Crossover

Parent 1 :	0	1	1	0	0	0	1	0
Parent 2 :	1	0	1	0	1	1	0	0

Two diploid from a mating pair

Select crossover points randomly

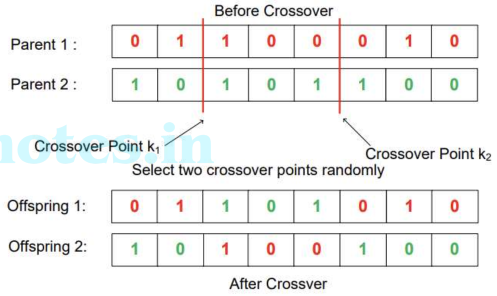
Offspring 1:	0	1	1	0	1	1	0	0
Offspring 2:	1	0	1	0	0	0	1	0

Two diploid for two new offspring is produced

After Crossver

crossover operator IN GA

- **Two-point crossover**
- Two-point crossover
- In this scheme, we select two different crossover points k_1 and k_2 lying between 1 and L at random such that $k_1 \# k_2$.
- The middle parts are swapped between the two strings.
- Alternatively, left and right parts also can be swapped.



Before Crossover

Parent 1 :	0	1	1	0	0	0	1	0
Parent 2 :	1	0	1	0	1	1	0	0

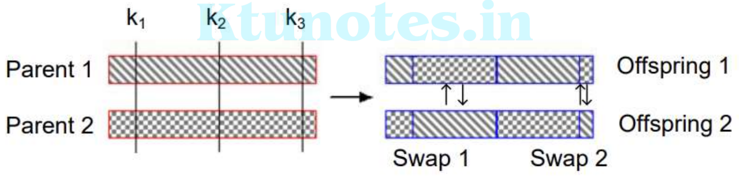
Select two crossover points randomly

Offspring 1:	0	1	1	0	1	0	1	0
Offspring 2:	1	0	1	0	0	1	0	0

After Crossver

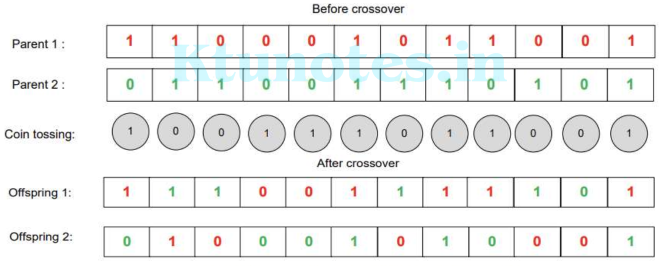
crossover operator IN GA

- **Multi-point crossover**
- In the case of multi-point crossover, a number of crossover points are selected along the length of the string, at random.
- The bits lying between alternate pairs of sites are then swapped.



crossover operator IN GA

- **Uniform crossover** is a more general version of the multi-point crossover.
- In this scheme, at each bit position of the parent string, we toss a coin (with a certain probability p_s (to determine whether there will be a swap of the bits or not. The two bits are then swapped or remain unaltered, accordingly.



Before crossover

Parent 1 :	1	1	0	0	0	1	0	1	1	0	0	1
Parent 2 :	0	1	1	0	0	1	1	1	0	1	0	1
Coin tossing:	1	0	0	1	1	1	0	1	1	0	0	1

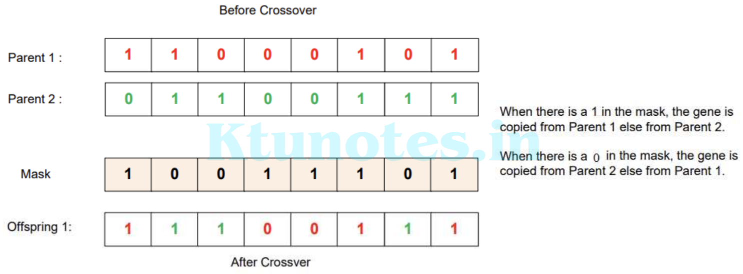
After crossover

Offspring 1:	1	1	1	0	0	1	1	1	1	1	0	1
Offspring 2:	0	1	0	0	0	1	0	1	0	0	0	1

Rule: If the toss is 0 than swap the bits between P1 and P2

Uniform crossover with crossover mask

- **Uniform crossover** is a more general version of the multi-point crossover.



Before Crossover

Parent 1 :	1	1	0	0	0	1	0	1
Parent 2 :	0	1	1	0	0	1	1	1
Mask	1	0	0	1	1	1	0	1
Offspring 1:	1	1	1	0	0	1	1	1

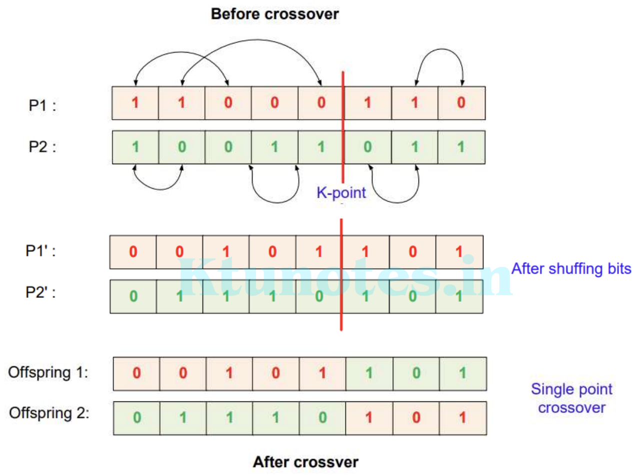
After Crossver

When there is a 1 in the mask, the gene is copied from Parent 1 else from Parent 2.

When there is a 0 in the mask, the gene is copied from Parent 2 else from Parent 1.

crossover in GA

- **Shuffle crossover**
- A single crossover point is selected. It divides a chromosome into two parts called schema.
- In both parents, genes are shuffled in each schema. Follow some strategy for shuffling bits.
- Schemas are exchanged to create offspring (as in single crossover)



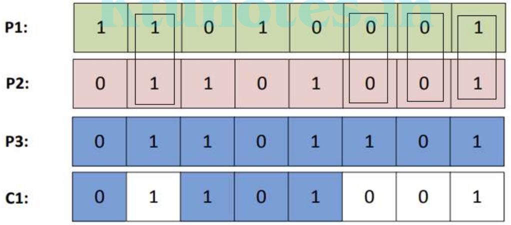
Before crossover

P1 :	1	1	0	0	0	1	1	0
P2 :	1	0	0	1	1	0	1	1
P1' :	0	0	1	0	1	1	0	1
P2' :	0	1	1	1	0	1	0	1
Offspring 1:	0	0	1	0	1	1	0	1
Offspring 2:	0	1	1	1	0	1	0	1

After crossver

crossover in GA

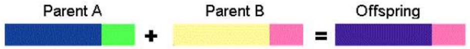
- **Three parent crossover**
- In this techniques, three parents are randomly chosen. Each bit of the first parent is compared with the bit of the second parent.
- If both are the same, the bit is taken for the offspring. Otherwise, the bit from the third parent is taken for the offspring.



P1:	1	1	0	1	0	0	0	1
P2:	0	1	1	0	1	0	0	1
P3:	0	1	1	0	1	1	0	1
C1:	0	1	1	0	1	0	0	1

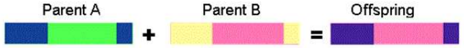
crossover operator IN GA

Single point crossover - one crossover point is selected, binary string from the beginning of chromosome to the crossover point is copied from one parent, and the rest is copied from the second parent



Ex: 11001011 + 11011111 = 11001111

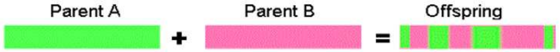
Two point crossover - two crossover points are selected, binary string from the beginning of the chromosome to the first crossover point is copied from one parent, the part from the first to the second crossover point is copied from the second parent, and the rest is copied from the first parent



Ex 11 0010 11 + 11 0111 11 = 11011111

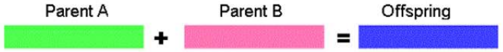
crossover operator IN GA

Uniform crossover - bits are randomly copied from the first or from the second parent



Ex: 11001011 + 11011101 = 11011111

Arithmetic crossover - some arithmetic operation is performed to make a new offspring



Ex 11001011 + 11011111 = 110010 11 (AND)

crossover operator IN GA

- In the one-point crossover, a crossover site is selected at random over the string length, and the alleles on one side of the site are exchanged between the individuals.
- In the two-point crossover, two crossover sites are randomly selected. The alleles between the two sites are exchanged between the two randomly paired individuals.
- The concept of one-point crossover can be extended to k-point crossover, where k-crossover points are used, rather than just one or two.
- In the uniform crossover, every allele is exchanged between a pair of randomly selected chromosomes with a certain probability, pe, known as the swapping probability. Usually, the swapping probability value is taken to be 0.5.

Mutation

Mutation IN GA

- Mutation adds to the diversity of a population and thereby increases the likelihood that the algorithm will generate individuals with better fitness values.
- Mutation is used to maintain genetic diversity from one generation of a population.
- It alters one or more gene values in the chromosome from its initial state
- mutation may be defined as a small random tweak in the chromosome, to get a new solution.
- Mutation is part of the GA which is related to the “exploration” of the search space.
- It has been observed that mutation is essential to the convergence of the GA while crossover is not.

Mutation IN GA

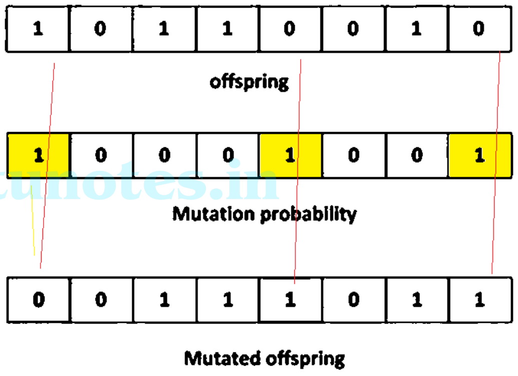
- There are many different forms of mutation for the different kinds of representation.
  - Binary-coded GA
    - Flipping
    - Interchanging
    - Reversing
  - Real-coded GA
    - Random Mutation
    - Polynomial Mutation
- For binary representation, a simple mutation can consist in inverting the value of each gene with a small probability.

Mutation IN GA

- For binary representation, a simple mutation can consist in inverting the value of each gene with a small probability.
- The mutation operator is simple and straightforward in binary coded GA Mutation of a bit involves flipping a bit, changing 0 to 1, and vice-versa
- A common method of implementing a mutation operator involves generating a random variable called mutation probability for each bit sequence.
- The probability is usually taken about 1/L, where L is the length of the chromosome. This mutation probability tells us whether or not a particular bit will be mutated .It is used to maintain and introduce diversity in the genetic population and is usually applied with a low probability – p_m.
- If the probability is very high, the GA gets reduced to a random search.

Flipping

- A mutation chromosome of the same length as the individual chromosome is created with a probability in the bit.
- For a bit in a mutation chromosome, the corresponding bit in the current chromosome is flipped and



1 0 1 1 0 0 1 0

offspring

1 0 0 0 1 0 0 1

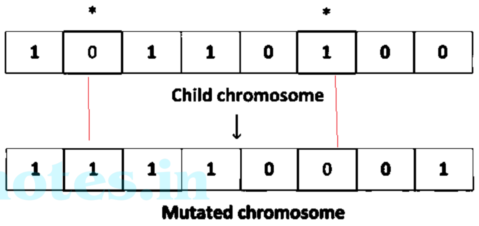
Mutation probability

0 0 1 1 1 0 1 1

Mutated offspring

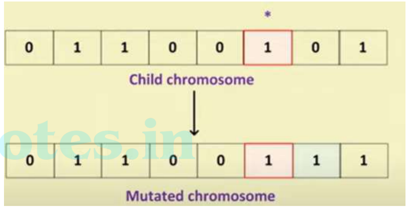
Interchanging

- Two positions of a child's chromosome are chosen randomly and the bits corresponding to that position are interchanged.



Reversing

- A position is chosen at random and the bit next to that position is reversed and the mutated child chromosome is produced.



Mutation IN GA

- **Bit Flip Mutation(inversion of single bit)**
- In this bit flip mutation, we select one or more random bits and flip them.
- This is used for binary-encoded GAs.

0 0 1 1 0 1 0 0 1 0 => 0 0 1 0 0 1 0 0 1 0

- **Random Resetting**
- Random Resetting is an extension of the bit flip for the integer representation.
- In this, a random value from the set of permissible values,is assigned to a randomly chosen gene.

Mutation IN GA

- **Swap Mutation**
- In swap mutation, we select two positions on the chromosome at random and interchange the values.
- This is common in permutation-based encodings.

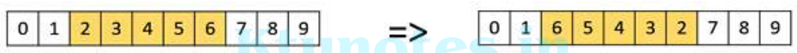
1 2 3 4 5 6 7 8 9 0 => 1 6 3 4 5 2 7 8 9 0

- **Scramble Mutation**
- Scramble mutation is also popular with permutation representations.
- In this, from the entire chromosome, a subset of genes is chosen and their values are scrambled or shuffled randomly.

0 1 2 3 4 5 6 7 8 9 => 0 1 3 6 4 2 5 7 8 9

Mutation IN GA

- **Inversion Mutation**
- In inversion mutation, we select a subset of genes like in scramble mutation, but instead of shuffling the subset, we merely invert the entire string in the subset.



0	1	2	3	4	5	6	7	8	9

=>

0	1	6	5	4	3	2	7	8	9

Stopping condition for genetic algorithm

Stopping condition for genetic algorithm

- The termination condition of a Genetic Algorithm is important in determining when a GA run will end.
- initially, the GA progresses very fast with better solutions coming in every few iterations, but this tends to saturate in the later stages where the improvements are very small.
- We want a termination condition such that our solution is close to the optimal, at the end of the run.
- we keep one of the following termination conditions –
- When there has been no improvement in the population for X iterations.
- When we reach an absolute number of generations.
- When the objective function value has reached a certain pre-defined value.

Stopping condition for genetic algorithm

- Maximum generations
- Elapsed time
- No change in fitness
- Stall generations
- Stall time limit

Stopping condition for genetic algorithm

- **Maximum generations**–The genetic algorithm stops when the specified number of generation's have evolved.
- **Elapsed time**–The genetic process will end when a specified time has elapsed.
- Note: If the maximum number of generation has been reached before the specified time has elapsed, the process will end.
- **No change in fitness**–The genetic process will end if there is no change to the population's best fitness for a specified number of generations.
- Note: If the maximum number of generation has been reached before the specified number of generation with no changes has been reached, the process will end.

Stopping condition for genetic algorithm

- **Stall generations**–The algorithm stops if there is no improvement in the objective function for a sequence of consecutive generations of length Stall generations.
- **Stall time limit**–The algorithm stops if there is no improvement in the objective function during *an interval of time in seconds* equal to Stall time limit.
- if there is no change in the best fitness value for some **time** given in seconds (**stall time limit**) or for some number of generations (**stall** generation **limit**).

Stopping condition for genetic algorithm

- The simple form of GA is given by:

1. Start with a randomly generated initial population
2. find fitness of each chromosomes in the population
3. Repeat the following steps until n offsprings have been created
    1. Select a pair of parent chromosomes from the current population
    2. crossover with probability pc at a randomly chosen point to form the offspring
    3. Mutate the two offsprings with probability pm
4. Replace the current population with the new population
5. Go to step2

- The algorithm is stopped when the population converges towards the optimal solution

Stopping condition for genetic algorithm

- For example, in a genetic algorithm we keep a counter which keeps track of the generations for which there has been no improvement in the population.
- Initially, we set this counter to zero. Each time we don't generate off-springs which are better than the individuals in the population, we increment the counter.
- However, if the fitness any of the off-springs is better, then we reset the counter to zero.
- The algorithm terminates when the counter reaches a predetermined value.

generalized pseudo-code for a GA

```
GA()
 BEGIN
     Generate the  initial population
     Compute the fitness of each individual in the population
   WHILE (termination criteria is reached) DO
     BEGIN
          Select INDIVIDUALS from old generation for mating(parent selection)
          Create offspring by applying crossover and mutation to the selected
               individuals
          Compute the fitness of new individual
          Kill the old individuals to make room for new chromosomes and insert
               offspring in the new generalization
          IF the population has converged(best) THEN finishes:=TRUE
     END
END
```

Applications of GA

Applications of genetic algorithm

- **Optimization** – Genetic Algorithms are most commonly used in optimization problems wherein we have to maximize or minimize a given objective function value under a given set of constraints. The approach to solve Optimization problems has been highlighted throughout the tutorial.
- **Economics** – GAs are also used to characterize various economic models like the cobweb model, game theory equilibrium resolution, asset pricing, etc.
- **Neural Networks** – GAs are also used to train neural networks, particularly recurrent neural networks.
- **Parallelization** – GAs also have very good parallel capabilities, and prove to be very effective means in solving certain problems, and also provide a good area for research.

Applications of genetic algorithm

- **Image Processing** – GAs are used for various digital image processing (DIP) tasks as well like dense pixel matching.
- **Vehicle routing problems** – With multiple soft time windows, multiple depots and a heterogeneous fleet.
- **Scheduling applications** – GAs are used to solve various scheduling problems as well, particularly the time tabling problem.
- **Machine Learning** – as already discussed, genetics based machine learning (GBML) is a niche area in machine learning.
- **Robot Trajectory Generation** – GAs have been used to plan the path which a robot arm takes by moving from one point to another.
- **Parametric Design of Aircraft** – GAs have been used to design aircrafts by varying the parameters and evolving better solutions.

Applications of genetic algorithm

- **DNA Analysis** – GAs have been used to determine the structure of DNA using spectrometric data about the sample.
- **Multimodal Optimization** – GAs are obviously very good approaches for multimodal optimization in which we have to find multiple optimum solutions.
- **Traveling salesman problem and its applications** – GAs have been used to solve the TSP, which is a well-known combinatorial problem using novel crossover and packing strategies.

References

- https://www.obitko.com/tutorials/genetic-algorithms/index.php
- https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_quick_guide.htm