

[Unit 7: Hybrid System]
Fuzzy Systems (CSc 613)

Jagdish Bhatta

Central Department of Computer Science & Information Technology
Tribhuvan University

FUZZY STSTEMS AND NEURAL NETWORKS

It has increasingly been recognized that the areas of fuzzy systems and neural network are strongly interconnected. For example, neural network have already been proven very useful in numerous applications of fuzzy set theory for constructing membership function of relevant fuzzy set and other context-dependent entities from sample data. They also show great promise as an efficient tool for solving fuzzy relations equations which play a fundamental role in dealing with many practical problem involving fuzzy systems.

The motivation for approximating fuzzy systems by neural network is based upon the inherent capability of neural network to perform massive parallel processing of information. This is important in those fuzzy controllers and, more generally, fuzzy expert system, that are required to process large numbers of fuzzy inference rules in real time. When the neural network representing a given fuzzy expert system is implemented in hardware, all relevant fuzzy inference rules are processing parallel. This results in high computations efficiency, which is curical in many applications. Furthermore, the neural network representation is eminently fitted for introducing suitable adaptive capabilities in to system. this is due to the inherent learning capability of neural network, which, in this context, can be utilized for modifying fuzzy inference rules if the system on the basis of experience.

FUZZY NEURAL NETWORK

Neural network are eminently suited for approximating fuzzy controllers and other types of fuzzy expert system, as well as for implemented these approximately in appropriate hardware. Although classical neural network can be employed for this purpose, attempts have been made to develop alternatives neural network, more attuned to the various procedures of approximate reasoning. These alternative neural network are usually referred to as *fuzzy neural networks*

The following features, or some of them, distinguish fuzzy neural network from their classical counterparts;

1. inputs are fuzzy numbers;
2. outputs are fuzzy numbers;
3. weights are fuzzy numbers;

Weights inputs of each neural are not aggregated by summation, but by some others aggregation operation like T-norm and Conorms.

A deviation from classical neural network in any of these feature require that a property modified learnig algorithm be developed. This, in some cases, is not an easy task. Various types of fuzzy neural network have been proposed in the literature. As an example, we describe basic characteristics of only one type. Some others are listed, with relevant references,

In fuzzy network to be describe here were proposed by hayashi, buckley, and czogala(1993). They are obtained by directly fuzzifying the classical feedforward neural network with one or more layers. The following are basic feature of the resulting networks:

1. All real numbers that characterize a classical neural network become fuzzy numbers in its fuzzified counterpart. These are number that characterize inputs to the network, outputs of neural at hidden layers and the output layers, and weight at all layers. Consider, for example, all numbers relevant to particular output neural . ON_k is a fuzzy neural, then the inputs $X_{k0}, X_{k1}, \dots, X_{km}$, the weight W_0, W_1, \dots, W_n , and the output Y_k of this neuron are all fuzzy numbers.

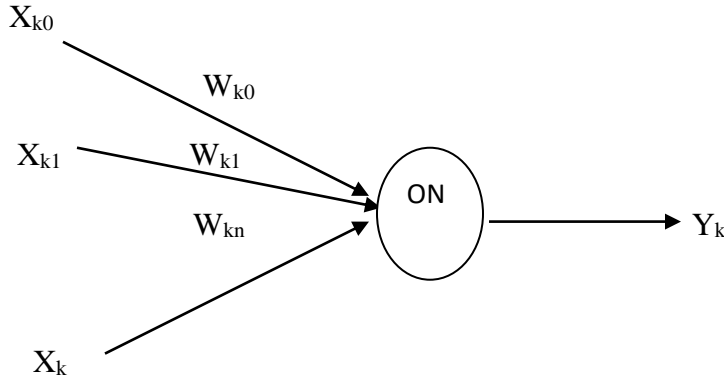


Figure: fuzzy numbers characterizing a single fuzzy neuron.

2. The output of each neuron, exemplified here by the neuron characterized in this figure is defined by the formula

$$Y_k = S_\beta (\sum_{j=0}^n W_j X_{kj}),$$

Where, S_β is a sigmoid function for some chosen value of the steepness parameters β . Symbols W_j and W_{kj} designate fuzzy numbers r , the sum.

$$A_k = \sum_{j=0}^n W_j X_{kj}$$

must be calculated by fuzzy arithmetic. the output of the neuron,

$$Y_k = S_\beta (A_k),$$

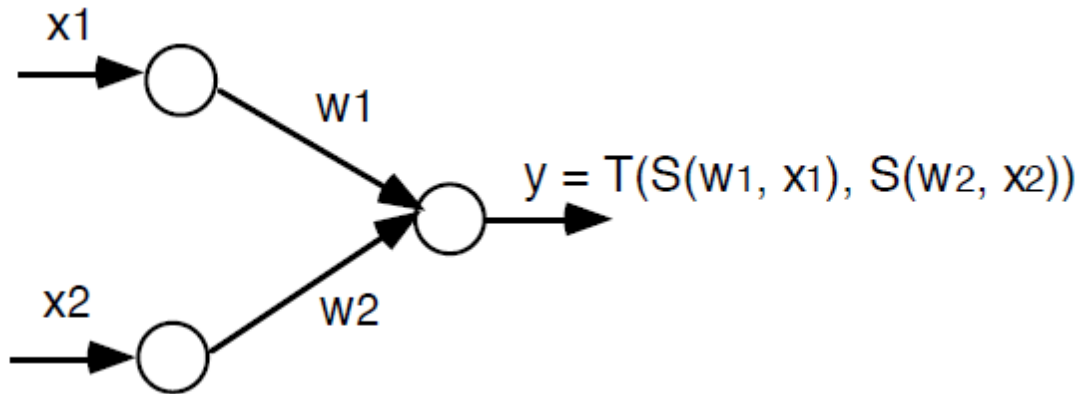
is determined by using the extension principle.

The direct fuzzification of conventional neural networks is to extend connection weights and/or inputs and/or fuzzy desired outputs (or targets) to fuzzy numbers. The extension is;

Fuzzy neural net	Weights	Inputs	Targets
<i>Type 1</i>	crisp	fuzzy	crisp
<i>Type 2</i>	crisp	fuzzy	fuzzy
<i>Type 3</i>	fuzzy	fuzzy	fuzzy
<i>Type 4</i>	fuzzy	crisp	fuzzy
<i>Type 5</i>	crisp	crisp	fuzzy
<i>Type 6</i>	fuzzy	crisp	crisp
<i>Type 7</i>	fuzzy	fuzzy	crisp

A regular fuzzy neural network is a neural network with fuzzy signals and/or fuzzy weights, sigmoidal transfer function and all the operations are defined by Zadeh's extension principle.

Consider a fuzzy neuron for And gate;



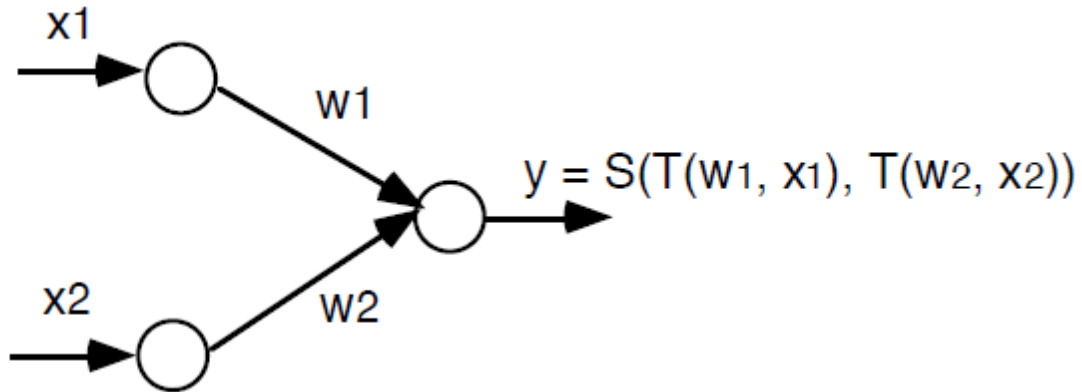
The signal x_i and w_i are combined by a triangular conorm S to produce $p_i = S(w_i, x_i)$, $i = 1, 2$.

The input information p_i is aggregated by a triangular norm T to produce the output of the neuron.

$$y = \text{AND}(p_1, p_2) = T(p_1, p_2) = T(S(w_1, x_1), S(w_2, x_2))$$

So, if $T = \min$ and $S = \max$ then the AND neuron realizes the min-max composition $y = \min\{w_1 \cdot x_1, w_2 \cdot x_2\}$.

Similarly, consider a fuzzy neuron for OR gate;



The signal x_i and w_i are combined by a triangular norm T to produce $p_i = T(w_i, x_i)$, $i = 1, 2$.

The input information p_i is aggregated by a triangular conorm S to produce the output of the neuron.

So, if $T = \min$ and $S = \max$ then the AND neuron realizes the max-min composition $y = \max\{w_1, x_1, w_2, x_2\}$.

Pseudo outer product-based fuzzy neural networks (POPFNN) are a family of neuro-fuzzy systems that are based on the linguistic fuzzy mode.

The "POPFNN" architecture is a five-layer neural network where the layers from 1 to 5 are called: input linguistic layer, condition layer, rule layer, consequent layer, output linguistic layer. The fuzzification of the inputs and the defuzzification of the outputs are respectively performed by the input linguistic and output linguistic layers while the fuzzy inference is collectively performed by the rule, condition and consequence layers.

The learning process of POPFNN consists of three phases:

1. Fuzzy membership generation
2. Fuzzy rule identification
3. Supervised fine-tuning

Genetic Algorithm

Nature has always been a great source of inspiration to all mankind. Genetic Algorithms (GAs) are search based algorithms based on the concepts of natural selection and genetics. GAs are a subset of a much larger branch of computation known as **Evolutionary Computation**.

GAs were developed by John Holland and his students and colleagues at the University of Michigan, most notably David E. Goldberg and has since been tried on various optimization problems with a high degree of success.

In GAs, we have 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 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.

The basic terminologies behind GA are;

Population – It is a subset of all the possible (encoded) solutions to the given problem. The population for a GA is analogous to the population for human beings except that instead of human beings, we have Candidate Solutions representing human beings. It is the set of chromosomes.

There are two primary methods to initialize a population in a GA. They are –

Random Initialization – Populate the initial population with completely random solutions.

Heuristic initialization – Populate the initial population using a known heuristic for the problem.

Chromosomes – A chromosome is one such solution to the given problem.

Gene – A gene is one element position of a chromosome.

Genotype – Genotype is the population in the computation space. In the computation space, the solutions are represented in a way which can be easily understood and manipulated using a computing system. Complete set of genetic material (all chromosomes) is called genome. Particular set of genes in genome is called genotype.

Phenotype – Phenotype is the population in the actual real world solution space in which solutions are represented in a way they are represented in real world situations. The genotype is with later development after birth base for the organism's phenotype, its physical and mental characteristics, such as eye color, intelligence etc.

Decoding and Encoding – for simple problems, the **phenotype and genotype** spaces are the same. However, in most of the cases, the phenotype and genotype spaces are different. **Decoding is a process of transforming a solution from the genotype to the phenotype space, while encoding is a process of transforming from the phenotype to genotype space.** Decoding should be fast as it is carried out repeatedly in a GA during the fitness value calculation.

Binary Encoding

Binary encoding is the most common one, mainly because the first research of GA used this type of encoding and because of its relative simplicity.

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

Chromosome A	101100101100101011100101
Chromosome B	111111100000110000011111

Permutation Encoding

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

In **permutation encoding**, every chromosome is a string of numbers that represent a position 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 useful for ordering problems like TSP.

Value Encoding

Direct value encoding can be used in problems where some more complicated values such as real numbers are used. Use of binary encoding for this type of problems would be difficult.

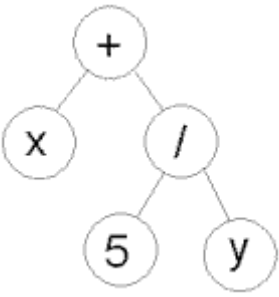
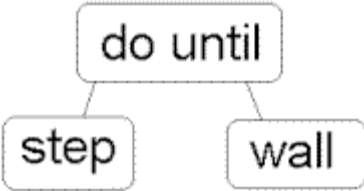
In the **value encoding**, every chromosome is a sequence of some values. Values can be anything connected to the problem, such as (real) numbers, chars or any objects.

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

Tree Encoding

Tree encoding is used mainly for evolving programs or expressions, i.e. for **genetic programming**.

In the **tree encoding** every chromosome is a tree of some objects, such as functions or commands in programming language.

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

Tree encoding is useful for evolving programs or any other structures that can be encoded in trees.

Fitness Function – A fitness function simply defined is a function which takes the solution as input and produces the suitability of the solution as the output.

Genetic Operators – They alter the genetic composition of the offspring. These include *selection, crossover, and mutation*.

Selection:

The selection phase involves the selection of individuals for the reproduction of offspring. All the selected individuals are then arranged in a pair of two to increase reproduction. Then these individuals transfer their genes to the next generation.

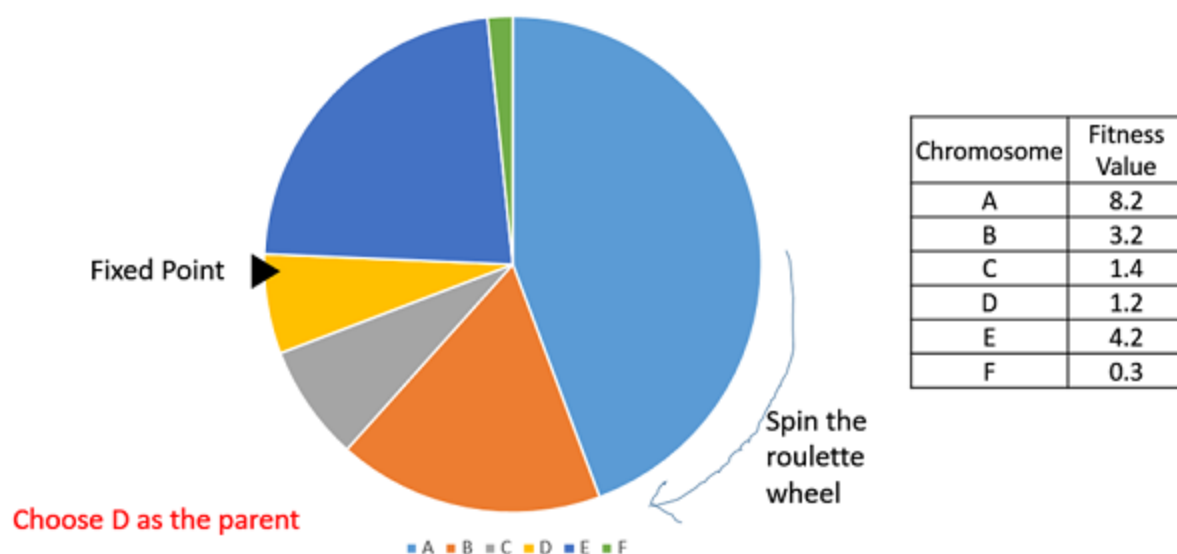
There are three types of Selection methods available, which are:

- Roulette wheel selection
- Stochastic universal sampling
- Tournament selection
- Rank-based selection

Roulette Wheel Selection

In a roulette wheel selection, the circular wheel is divided into slots of region. **A fixed point is chosen on the wheel circumference as shown and the wheel is rotated.** 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.

The probability for selection of an individual is proportional to its fitness. If all slots have same fitness probability of being selected is same. Instead, we can implement a weighted version of the roulette. **With it, the larger the fitness of an individual is, the more likely is its selection:**



If f_i is the fitness of individual I in the population, its probability of being selected is

$$p_i = \frac{f_i}{\sum_{j=1}^N f_j},$$

Here the sum of fitness is = 18.5

Here the percentage of area in the wheel for the chromosome A = $8.2/18.5=0.44=44\%$.
Similarly for D= $0.648=6.5\%$

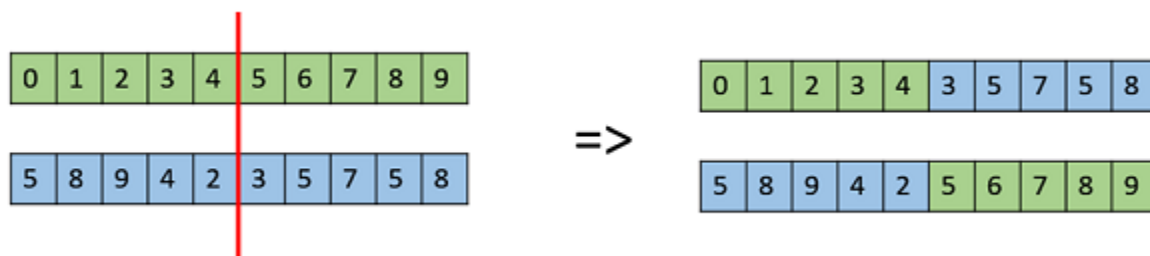
It is clear that a fitter individual has a greater pie on the wheel and therefore a greater chance of landing in front of the fixed point when the wheel is rotated. **Therefore, the probability of choosing an individual depends directly on its fitness.**

Crossover

The crossover operator is analogous to reproduction and biological crossover. In this more than one parent is selected and one or more off-springs are produced using the genetic material of the parents.

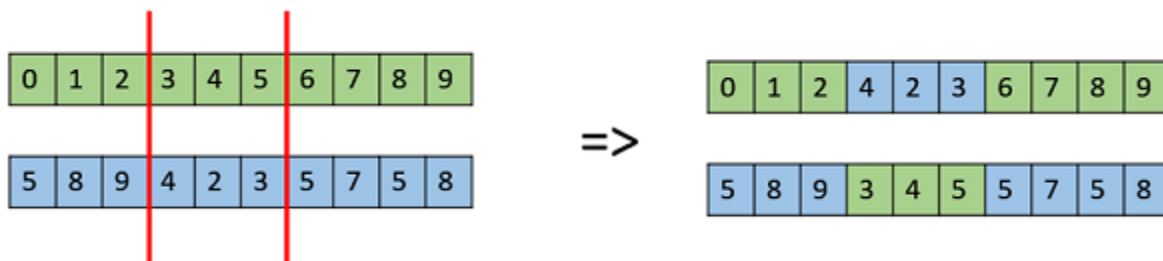
One Point Crossover

In this one-point crossover, a random crossover point is selected and the tails of its two parents are swapped to get new off-springs.



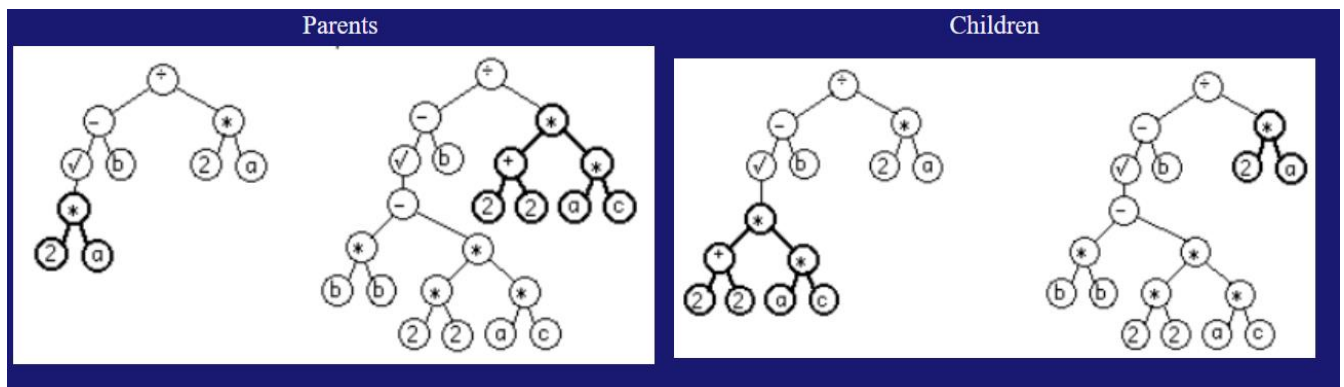
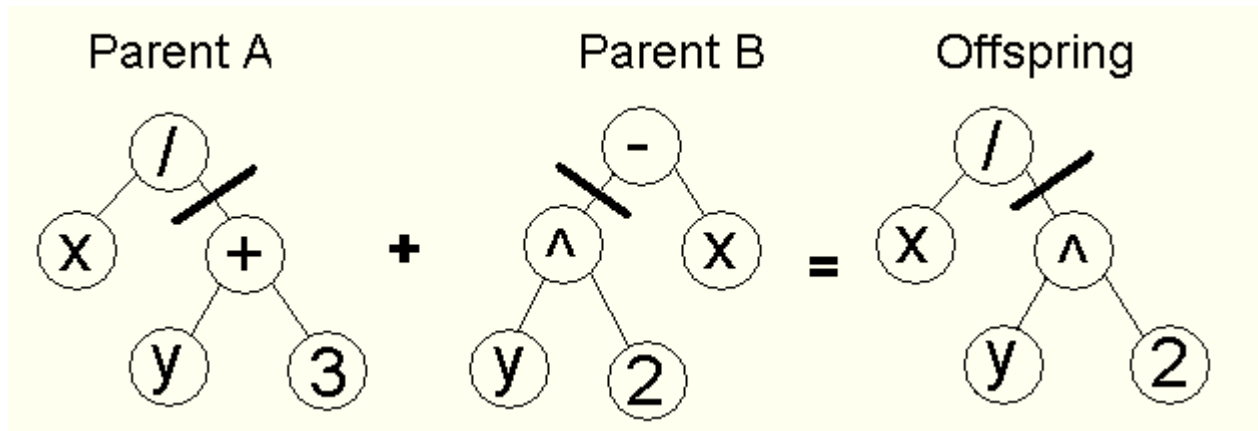
Multi Point Crossover

Multi point crossover is a generalization of the one-point crossover wherein alternating segments are swapped to get new off-springs.



Tree crossover

One crossover point is selected in both parents, parents are divided in that point and the parts below crossover points are exchanged to produce new offspring.



Mutation

Mutation may be defined as a small random tweak in the chromosome, to get a new solution. It is used to maintain and introduce diversity in the genetic population. The mutation operator inserts random genes in the offspring (new child) to maintain the diversity in the population. It can be done by flipping some bits in the chromosomes.

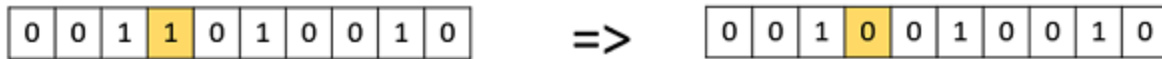
Mutation operators can be

- Bit flip mutation

- Random resetting
- Swap mutation and so on.

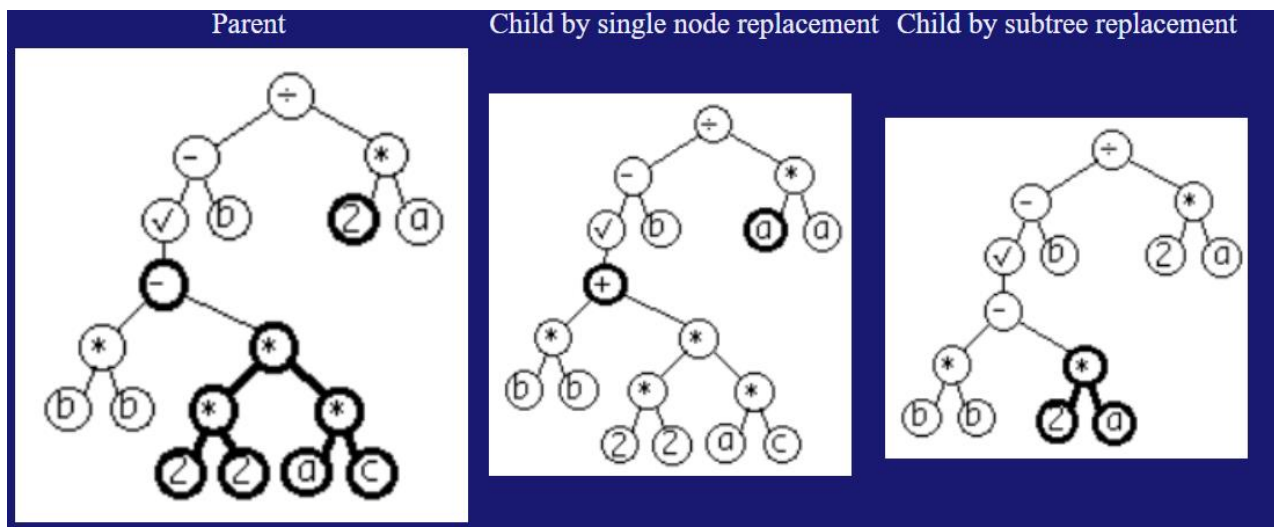
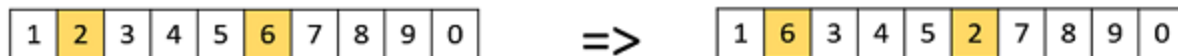
Bit Flip Mutation

In this bit flip mutation, we select one or more random bits and flip them. This is used for binary encoded GAs.



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.



There are two basic parameters of GA - crossover probability and mutation probability.

Crossover probability: how often crossover will be performed. If there is no crossover, offspring are exact copies of parents. If there is crossover, offspring are made from parts of both parent's chromosome. **If crossover probability is 100%, then all offspring are made by crossover.** If it is 0%, whole new generation is made from exact copies of chromosomes from old population.

Crossover is made in hope that new chromosomes will contain good parts of old chromosomes and therefore the new chromosomes will be better. However, it is good to leave some part of old population survive to next generation.

Mutation probability: how often parts of chromosome will be mutated. If there is no mutation, offspring are generated immediately after crossover (or directly copied) without any change. If mutation is performed, one or more parts of a chromosome are changed. **If mutation probability is 100%, whole chromosome is changed, if it is 0%, nothing is changed.**

If the chromosome is mutated, normally a few bits are changed and so the solution is a bit changed. But **if all bits in chromosome** are mutated (because of 100% mutation probability), then the chromosome **string is in fact inverted**. Then we **have inversion and no mutation, so whole population degenerates very quickly**. It behaves like with no mutation (0% mutation probability).

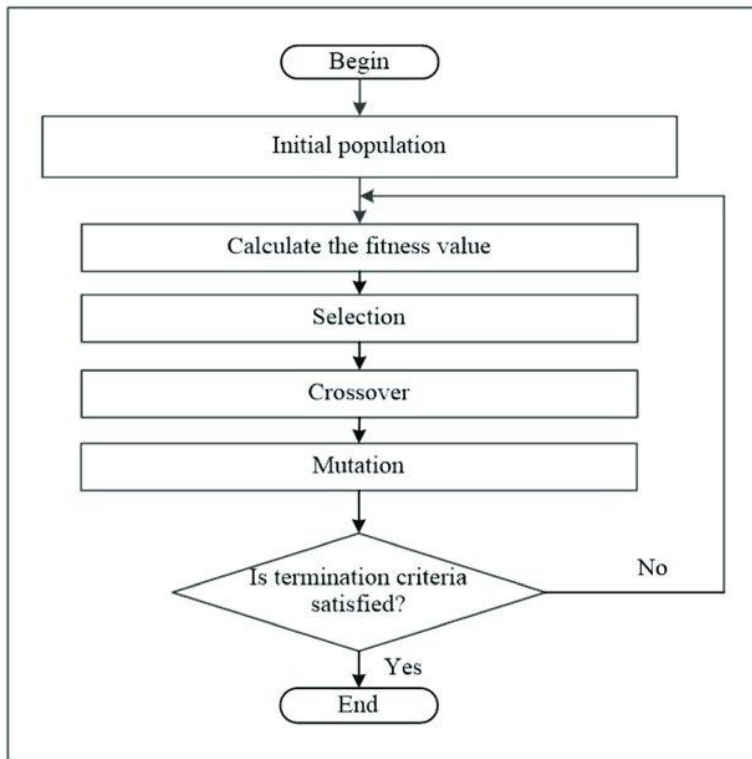
Termination Condition in GA

There are different termination conditions, which are listed below:

1. There is no improvement in the population for over x iterations.
2. We have already predefined an absolute number of generation for our algorithm.
3. When our fitness function has reached a predefined value.

GA Algorithm:

1. Generate random population of n individuals.
2. Evaluate the fitness of each individual.
3. Create a new population.
 - a. Select two parents from a population according to their fitness.
 - b. Crossover operation performed between the two parents
 - c. Mutation operation is performed this will change the trip from the output of crossover step, if not the children will be the same
 - d. Add new children in a new population.
4. Evaluate the fitness of each individual.
5. If the stopping criteria is satisfied, the algorithm stops and shows the best trip, if not it will start over from step 3 and continues the iteration.



Consider the problem of maximizing the function, $f(x) = x^2$ where x is permitted to vary between 0 to 31.

Step 1: For using genetic algorithms approach, one must first code the decision variable ‘ x ’ into a finite length string. Using a five bit (binary integer) unsigned integer, numbers between 0(00000) and 31(11111) can be obtained. The objective function here is $f(x) = x^2$ which is to be maximized.

A single generation of a genetic algorithm is performed here with encoding, selection, crossover and mutation. To start with, select initial population at random.

Here **initial population** of size 4 is chosen, but any number of populations can be elected based on the requirement and application. Table 1 shows an initial population randomly selected.

String No.	Initial population (randomly selected)	x value	Fitness value $f(x) = x^2$	Prob _i	Percentage probability	Expected count	Actual count
1	01100	12	144	0.1247	12.47%	0.4987	1
2	11001	25	625	0.5411	54.11%	2.1645	2
3	00101	5	25	0.0216	2.16%	0.0866	0
4	10011	19	361	0.3126	31.26%	1.2502	1
Sum			1155	1.0000	100%	4.0000	4
average			288.75	0.2500	25%	1.0000	1
maximum			625	0.5411	54.11%	2.1645	2

Table 1.Selection

Step 2: Obtain the decoded x values for the initial population generated. Consider string 1, thus for all the four strings the decoded values are obtained.

Step 3: Calculate the **fitness function**. This is obtained by simply squaring the ‘ x ’ value, since the given function is $f(x) = x^2$. When, $x = 12$, the **fitness value** is, $f(x) = 144$ for $x = 25$, $f(x) = 625$ and so on, until the entire population is computed

Step 4: Compute **the probability of selection**,

$$\text{Prob}_i = \frac{f(x)_i}{\sum_{i=1}^n f(x)_i}$$

where n = no of populations $f(x)$ =fitness value corresponding to a particular individual in the population

$\Sigma f(x)$ - Summation of all the fitness value of the entire population.

Considering string 1, Fitness $f(x) = 144$ $\Sigma f(x) = 1155$. The probability that string 1 occurs is given by, $P1 = 144/1155 = 0.1247$. The **percentage probability** is obtained as, $0.1247*100 = 12.47\%$.

The same operation is done for all the strings. It should be noted that, summation of probability select is 1.

Step 5: The next step is to calculate the **expected count**, which is calculated as,

$$\text{Expected count} = \frac{f(x)_i}{(\text{Avg} f(x))_i}$$

$$\text{where } (\text{Avg } f(x))_i = \left[\frac{\sum_{i=1}^n f(x)_i}{n} \right]$$

For string 1, Expected count = Fitness/Average = $144/288.75 = 0.4987$. Computing the expected count for the entire population. The **expected count gives an idea of which population can be selected for further processing in the mating pool**.

Note: Expected count is not mandatory always. Selection can be based on probabilities only.

Step 6: Now the actual count is to be obtained to select the individuals, which would participate in the crossover cycle using Roulette wheel selection.

The Roulette wheel is formed as shown in following figure. Roulette wheel is of 100% and the probability of selection as calculated in step 4 for the entire populations are used as indicators to fit into the Roulette wheel. Now the wheel may be spun and the no of occurrences of population is noted to get **actual count**.

String 1 occupies **12.47%**, so there is a chance for it to occur at least once. Hence its **actual count may be 1** since **expected count** for string 1 is **0.4987=0.5(rounded)=1(rounded)** .

With string 2 occupying 54.11% of the Roulette wheel, it has a fair chance of being selected twice. Thus its **actual count can be considered as 2** since the **expected count is 2.1645=2(rounded)**.

On the other hand, string 3 has the least probability percentage of 2.16%, so their occurrence for next cycle is very poor. As a result, it actual count is 0 since its expected count is 0.0866.

String 4 with 31.26% has at least one chance for occurring while Roulette wheel is spun, thus its actual count is 1 since its expected count is 1.2502. The above values of actual count are tabulated as shown in Table 1

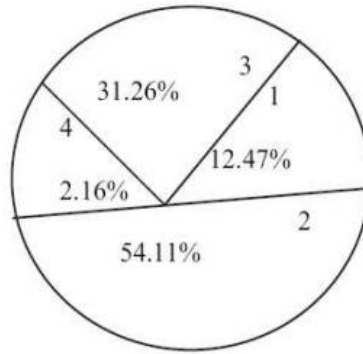


Fig: Roulette Wheel Selection

Step 7: Now, writing the mating pool based upon the actual count as shown in Table 2. The actual count of string no 1 is 1, hence it occurs once in the mating pool. The actual count of string no 2 is 2, hence it occurs twice in the mating pool. **Since the actual count of string no 3 is 0, it does not occur in the mating pool.** Similarly, the actual count of string no 4 being 1, it occurs once in the mating pool. Based on this, the mating pool is formed.

String No.	Mating pool	Crossover point	Offspring after crossover	x value	Fitness value $f(x) = x^2$
1	0 1 1 0:0	4	0 1 1 0 1	13	169
2	1 1 0 0:1	4	1 1 0 0 0	24	576
2	1 1 0 0:1	2	1 1 0 1 1	27	729
4	1 0 0:1 1	2	1 0 0 0 1	17	289
Sum					1763
average					440.75
maximum					729

Table 2.Crossover

**Crossover point should be 3 not 2 in above table for the last two strings 2 and 4.*

Step 8: Crossover operation is performed to produce new offspring (children). The crossover point is specified and based on the crossover point, single point crossover is performed and new offspring is produced.

Step 9: After crossover operations, new off springs are produced and 'x' values are decoded and fitness is calculated.

Step 10: In this step, mutation operation is performed to produce new off springs after crossover operation. As discussed in mutation-flipping operation is performed and new off springs are produced. Table 3 shows the new offspring after mutation. Once the off springs are obtained after mutation, they are decoded to x value and find fitness values are computed. This completes one generation. The mutation is performed on a bit-bit by basis.

String No.	Offspring after crossover	Mutation chromosomes for flipping	Offspring after Mutation	X value	Fitness value $F(x) = x^2$
1	0 1 1 0 1	1 0 0 0 0	1 1 1 0 1	29	841
2	1 1 0 0 0	0 0 0 0 0	1 1 0 0 0	24	576
2	1 1 0 1 1	0 0 0 0 0	1 1 0 1 1	27	729
4	1 0 0 0 1	0 0 1 0 0	1 0 1 0 0	20	400
Sum					2546
average					636.5
maximum					841

Table 3. Mutation

Once selection, crossover and mutation are performed, the new population is now ready to be tested. This is performed by decoding the new strings created by the simple genetic algorithm after mutation and calculates the fitness function values from the x values thus decoded.

The results for successive cycles of simulation are shown in Tables 1–3. From the tables, it can be observed how genetic algorithms combine high performance notions to achieve better performance.

In the tables, it can be noted how maximal and average performance has improved in the new population. The population average fitness has improved from 288.75 to 636.5 in one generation. The maximum fitness has increased from 625 to 6 841 during same period.

Although random processes make this best solution, its improvement can also be seen successively.

The best string of the initial population (1 1 0 0 1) receives two chances for its existence because of its high, above-average performance.

When this combines at random with the next highest string (1 0 0 1 1) and is crossed at crossover point 2 (as shown in Table 3.2), one of the resulting strings (1 1 0 1 1) proves to be a very best solution indeed.

Thus after mutation at random, a new offspring (1 1 1 0 1) is produced which is an excellent choice. This example has shown one generation of a simple genetic algorithm.