BITS Pilani
Pilani Campus

BITS F343

Fuzzy logic and applications

Assignment

Aggregation Operations and Fuzzy Genetic Algorithm

Submitted By:

Naman Deep Srivastava	2016B4A70891P
Mohit Kriplani	2016B1A70870P
Pranjal Dave	2016B4A30499P

Teaching Assignment

Aggregation Operations

Aggregation functions are functions with special properties. Aggregation functions that take real arguments from the closed interval [0, 1] and produce a real value in [0, 1]. This is usually denoted as

$$f: [0,1]^n \rightarrow [0,1]$$

for functions that take arguments with n components.

The purpose of aggregation functions (also called as aggregation operators) is to combine inputs that are typically interpreted as degrees of membership in fuzzy sets.

Use of Aggregation function

Consider the following prototypical situations. We have several criteria with respect to which we assess different options, and every option fulfills each criterion only partially (i.e it has a score on [0, 1] scale). The aim is to evaluate the combined score for each option (possibly to rank the options).

Example (A group decision making problem). Let's say there are two (or more) alternatives, and n decision makers(experts) who evaluate each alternative as $x_1, x_2, \ldots, x_n$. The goal is to combine these evaluations using some aggregation function f, to obtain a global score $f(x_1, x_2, \ldots, x_n)$ for each alternative.

Aggregation function: An aggregation function is a function of n > 1 arguments that maps the (n-dimensional) unit cube onto the unit interval $f: [0,1]^n \rightarrow [0,1]$, with the properties

$(i)\ f(0,0,\ldots n - times) = 0 and f(1,1,\ldots n - times) = 1\ (ii)\ x \leq y\ implies\ f(x) \leq f(y)\ for\ all\ x, y \in [0,1]^n$

Classification and general properties

Main classes

There are four main classes of aggregation functions are:

- Averaging
- Conjunctive
- Disjunctive
- Mixed

Averaging aggregation: An aggregation function f has averaging behavior (or is averaging) if for every x it is bounded by

$$min(x) \leq f(x) \leq max(x)$$

Conjunctive aggregation: An aggregation function f has conjunctive behavior (or is conjunctive) if for every x it is bounded by

$$f(x) \leq min(x) = min(x_1, x_{2,\dots} x_n)$$

Disjunctive aggregation: An aggregation function f has disjunctive behavior (or is disjunctive) if for every x it is bounded by

$$f(x) \geq max(x) = max(x_1, x_{2,} \dots x_n)$$

Mixed aggregation: An aggregation function f is mixed if it does not belong to any of the above classes, i.e., it exhibits different types of behavior on different parts of the domain.

Fuzzy Union

$\cup$: $[0,1] \times [0,1] \rightarrow [0,1]$, $max\{\mu_A(a), \mu_A(b)\}$

Properties:

A1: $\cup(0,0) = 0, \cup(0,1) = 1, \cup(1,0) = 1, \cup(1,1) = 1 \quad (BoundaryCondition)$

A2: $If a < a', b < b' \; then \cup(a,b) \leq \cup(a',b') \quad (Monotonic)$

A3: $\cup(a,b) = \cup(b,a) \quad (Commutative)$

A4: $\cup(\cup(a,b),c) = \cup(a,\cup(b,c))$ $(Associative)$

A5: $\cup\ is\ continuous$

A6: $\cup(a,a) = a$ $(IdempotentLaw)$

Fuzzy Intersection

$I: [0,1]\times[0,1] \to [0,1],\ min\{\mu_A(a), \mu_A(b)\}$

Properties:

A1: $I(0,0) = 0, I(0,1) = 0, I(1,0) = 0, I(1,1) = 1$ $(BoundaryCondition)$

A2: $If a < a', b < b'\ then\ I(a,b) \le I(a',b')$ $(Monotonic)$

A3: $I(a,b) = I(b,a)$ $(Commutative)$

A4: $I(I(a,b),c) = I(a, I(b,c))$ $(Associative)$

A5: $I\ is\ continuous$

A6: $I(a,a) = a$ $(Idempotent\ Law)$

Other Union Operations

1. **Algebraic Sum** $(A^{\sim} \hat{+} B^{\sim})$

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

2. **Bounded Sum** $(A^{\sim} \oplus B^{\sim})$

$$\mu_{A\oplus B}(x) = min\{1, \mu_A(x) + \mu_B(x)\}$$

3. **Drastic Sum** $(A^{\sim} \cup B^{\sim})$

$$\mu_{A\cup B}(x) = \{\mu_A(x); \mu_B(x) = 0\ \mu_B(x); \mu_A(x) = 0\ 1; elsewhere$$

4. **Himachor's Sum** $(A^{\sim} \cup B^{\sim})$

$$\mu_{A\cup B}(x) = \frac{\mu_A(x)+\mu_B(x)-(2-\gamma)\mu_A(x)\mu_B(x)}{1-(1-\gamma)\mu_A(x)\mu_B(x)}; \gamma \geq 0$$

5. **Yager Class Union**

$$U_w(a,b) = min\{1, (a^w + b^w)^{\frac{1}{w}}\}; w \in (0,\infty) \ \ [Satisfies A1, A2, A3, A4, A5]$$

For different values of w:

$$U_{w=1}(a,b) = \{1, a+b\}$$
$$U_{w=2}(a,b) = \left\{1, \sqrt{a^2+b^2}\right\}$$
$$U_{w\to\infty}(a,b) = max(a,b) \ \ [StandardUnion]$$

Simple Disjoint Sum

$A^{\sim} \triangle B^{\sim} = (A^{\sim} \cap B^{\sim c}) \cup (A^{\sim c} \cap B^{\sim})$

It is equivalent to exclusive OR operation (XOR) between the two sets.

Disjunctive Sum

$A^{\sim} \square B^{\sim} = (A^{\sim} \cup B^{\sim c}) \cap (A^{\sim c} \cup B^{\sim})$

Disjoint Sum

$\mu_{A\triangle B}(x) = |\mu_A(x) - \mu_B(x)|$

Difference in Fuzzy Theory

$A^{\sim} \ominus B^{\sim} = max\{0, \mu_A(x) - \mu_B(x)\} \ \ [Bounded\ Difference]$

Other Intersection Operations

1. **Algebraic Product** $(A^{\sim} \wedge B^{\sim})$

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

2. **Bounded Product** $(A^{\sim} \odot B^{\sim})$

$$\mu_{A \odot B}(x) = max\{0, \mu_A(x) + \mu_B(x) - 1\}$$

3. **Drastic Product** $(A^{\sim} \cap B^{\sim})$

$$\mu_{A \cap B}(x) = \{\mu_A(x); \mu_B(x) = 1 \ \mu_B(x); \mu_A(x) = 1 \ 0; elsewhere$$

4. **Himachor's Product** $(A^{\sim} \cap B^{\sim})$

$$\mu_{A \cap B}(x) = \frac{\mu_A(x)\mu_B(x)}{\gamma + (1-\gamma)(\mu_A(x) + \mu_B(x) - \mu_A(x)\mu_B(x))}; \gamma \geq 0$$

5. **Yager Class Intersection**

$$I_w(a,b) = 1 - min\{1, ((1-a)^w + (1-b)^w)^{\frac{1}{w}}; w \in (0, \infty) \ [Satisfies A1, A2, A3, A4]$$

For different values of w:

$$I_{w=1}(a,b) = 1 - \{1, 2 - a - b\}$$

$$I_{w=2}(a,b) = 1 - \left\{1, \sqrt{(1-a)^2 + (1-b)^2}\right\}$$

$$I_{w \to \infty}(a,b) = min\{a,b\} \ [Standard\ Intersection]$$

Averaging Operations

1. **Fuzzy AND**

$$\mu_{AND}(A^{\sim}, B^{\sim})(x) = \gamma . min\{\mu_A(x), \mu_B(x)\} + (1-\gamma)(\frac{\mu_A(x) + \mu_B(x)}{2}); \gamma \in [0,1], x \in U$$

If $\gamma = 1$, then it is equivalent to minimum operator.

If $\gamma = 0$, then it is equivalent to arithmetic mean.

2. **Fuzzy OR**

$$\mu_{OR}(A^{\sim}, B^{\sim})(x) = \gamma . max\{\mu_A(x), \mu_B(x)\} + (1-\gamma)(\frac{\mu_A(x) + \mu_B(x)}{2}); \gamma \in [0,1], x \in U$$

If $\gamma = 1$, then it is equivalent to maximum operation.

If $\gamma = 0$, then it is equivalent to arithmetic mean.

Triangular Norms and Conorms

Two special and well–known classes of conjunctive and disjunctive aggregation functions are the triangular norms and conorms. Triangular norms have become especially popular as models for fuzzy sets intersection. They are also applied in studies of probabilistic metric spaces, many-valued logic, non-additive measures and integrals, etc.

Triangular norm (t-norm)

A triangular norm (t-norm) is a bivariate aggregation function $T: [0,1]^2 \rightarrow [0,1]$ which is associative, symmetric and has neutral element 1.

It satisfies the following axioms:

A1: $t(x,0) = 0, t(x,1) = x \ [BoundaryConditions]$

A2: $t(x,y) = t(y,x) \ [Commutative]$

A3: $If x \leq x', y \leq y' then\ t(x,y) \leq t(x',y')$

A4: $t(t(x,y),z) = t\big(x, t(y,z)\big) \ [Associative]$

A5: $t(x,y) \leq t(x,z) \ \ if \ \ y \leq z$

Classification

1. **Strict t-norm:** A t-norm T is called strict if it is continuous and strictly monotone on $[0,1]^2$,i.e., $T(t,u) < T(t,v)$ whenever $t > 0\ and\ u < v$.

2. **Nilpotent t-norm:** A t-norm T is called nilpotent if it is continuous and each element $a \in [0,1]$ is a nilpotent element of T, i.e., if there exists an $n \in \{1,2,\ldots\}$ such that $T(a, \ldots n-times \ldots, a) = 0$ for any $a \in]0,1[$.

3. **Archimedian t-norm:** A t-norm is called Archimedian if for each $(a,b) \in]0,1[^2$ there is an $n \in \{1,2,\ldots\}\ with\ T(a, \ldots n - times \ldots, a) < b$.

Examples

1. **Standard Intersection**

$$T_m(a,b) = min(a,b)$$

2. **Bounded Product**

$$T_L(a,b) = max\{0, a+b-1\}$$

3. **Algebraic Product**

$$T_p(a,b) = a.b$$

4. **Drastic Product**

$$T_D(a,b) = \{a; if b = 1\ b; if a = 1\ 0; elsewhere$$

5. **Yager Intersection**

$$T_w(a,b) = 1 - min\{1, ((1-a)^w + (1-b)^w)^{\frac{1}{w}}\}$$

6. **Nilpotent Minimum**

$$T_{nm}(a,b) = \{min\{a,b\}; if a + b \geq 1\ 0; elsewhere$$

7. **Dubois and Prade Intersection**

$$T_{Dub}(a,b) = \frac{ab}{max\{a,b,\alpha\}}; \alpha \in [0,1]$$

When $\alpha = 1$, then $T_{Dub}(a,b) = T_p(a,b)$.

Property

$T_D(a,b) \leq T(a,b) \leq T_m(a,b)$

Triangular conorm (t-conorm)

A triangular conorm is a bivariate aggregation function $S{:}\,[0,1]^2 \rightarrow [0,1]$, which is associative, symmetric and has neutral element 0.

It satisfies the following axioms:

A1: $s(x,0) = x, s(x,1) = 1 \ \ [BoundaryConditions]$

A2: $s(x,y) = s(y,x) \ \ [Commutative]$

A3: $If x \leq x', y \leq y' \ then \ s(x,y) \leq s(x',y')$

A4: $s(s(x,y),z) = s\big(x, s(y,z)\big) \ \ [Associative]$

A5: $s(x,y) \leq s(x,z) \ if \ y \leq z$

Classification

1. **Strict t-conorm:** A t-conorm S is called strict if it is continuous and strictly increasing on $[0,1]^2$,i.e., $S(t,u) < S(t,v)$ whenever $t < 1 \ and \ u < v$.

2. **Nilpotent t-norm:** A t-conorm S is called nilpotent if it is continuous and each element $a \in [0,1]$ is a nilpotent element of S, i.e., if there exists an $n \in \{1,2,\dots\}$ such that $S(a, \dots n - times \dots, a) = 1$ for any $a \in]0,1[$.

3. **Archimedian t-norm:** A t-conorm is called Archimedian if for each $(a,b) \in]0,1[^2$ there is an $n \in \{1,2,\dots\} \ with \ S(a, \dots n - times \dots, a) > b$.

Examples

1. **Standard Union**

$$S_m = max\{a,b\}$$

2. **Bounded Sum**

$$S_L(a,b) = min\{a+b, 1\}$$

3. **Probabilistic Sum**

$$S_p(a,b) = a + b - ab$$

4. **Drastic Sum**

$$S_D(a,b) = \{a; if b = 0\ b; if a = 0\ 1; elsewhere$$

5. **Yager Union**

$$S_w(a,b) = min\{1, (a^w + b^w)^{\frac{1}{w}}\}; w \in [0, \infty)$$

6. **Dubois and Prade Union**

$$S_{Dub}(a,b) = \frac{a+b-ab-min\{a,b,1-\alpha\}}{max\{1-a,1-b,\alpha\}}; \alpha \in [0,1]$$

When $\alpha = 0$, then $S_{Dub}(a,b) = S_m(a,b)$.

When $\alpha = 1$, then $S_{Dub}(a,b) = S_p(a,b)$.

Property

$S_m(a,b) \leq S(a,b) \leq S_D(a,b)$

Means

Means are averaging aggregation functions.

1. Arithmetic mean: The arithmetic mean is the function

$$M(x) = \frac{1}{n}(x_1 + x_{2+\ldots.+}x_n)$$

2. Weighted arithmetic mean: Given a weighting vector w, the weighted arithmetic mean is the function

$$M_w(x) = w_1x_1 + w_2x_2 + \ldots + w_nx_n$$

where A vector w = (w_1, . . ., w_n) is called a weighting vector if $w_i \in [0, 1]$ and $\sum_{i=1}^{n} w_i = 1$.

3. Geometric mean: The geometric mean is the function

$$G(x) = \sqrt[n]{x_1 x_2 \ldots x_n}$$

4. Harmonic mean: The harmonic mean is the function

$$H(x) = n(\sum_{i=1}^{n} \frac{1}{x_i})^{-1}$$

Medians

1. Median: The median is the function

$$Med(x) = \{\frac{1}{2}(x_k + x_{k+1}), if n = 2k \ is \ even \ x_k, if n = 2k - 1 \ is \ odd,$$

$$where \ x_k \ is \ the \ kth \ largest \ (or \ smallest) \ component \ of \ x.$$

2. a-Median: Given a value a ∈ [0, 1], the a-median is the function

$$Med_a(x) = Med(x_1, \ldots. x_n, a, ..(n-1) times.., a).$$

Choosing an aggregation function

This deals with which aggregate function would be most suitable for a specific application. We should think about these:

- Is one aggregation function enough or various functions should be used for different parts of application?
- Should the aggregate function be symetric, have a neutral or absorbing element, or be idempotent?
- Is the number of inputs always the same? What is the interpretation of input values?

Answering these questions should result in a number of mathematical properties, based on which a suitable class or family can be chosen. This can be broken down to many parts.

- First of all, the selected aggregation function must be consistent with the semantics of the aggregation procedure. That is, if one models a conjunction, averaging or disjunctive aggregation functions are not suitable.

- Once the class is chosen, second issue is to choose the appropriate member of that family. It is expected that the developer of a system has some rough idea of what the appropriate outputs are for some prototype inputs.
- Now comes the issue of fitting the data. The data may come from different sources and in different forms. If there is more than one expert, their outputs could be either averaged, or translated into the range of possible output values, or the experts could be brought together to find a consensus.

MATLAB Code

```
function
[union_standard,union_algebraic,sum_bounded,sum_drastic,sum_himachor,union_yage
rclass,sum_simple_disjoint,sum_disjunctive,sum_disjoint,union_dubois_prade,inte
rsection_standard,product_algebraic,product_bounded,product_drastic,product_him
achor,intersection_yagerclass,minimum_nilpotent,intersection_dubois_prade,and_f
uzzy,or_fuzzy] = fuzzy_aggregate(A,B)
    % Union aggregation operations
    union_standard = standard_union(A,B);
    union_algebraic = algebraic_sum(A,B);
    sum_bounded = bounded_sum(A,B);
    sum_drastic = drastic_sum(A,B);
    h_gamma = 1;
    sum_himachor = himachor_sum(A,B,h_gamma);
    yc_w = 2;
    union_yagerclass = yagerclass_union(A,B,yc_w);
    sum_simple_disjoint = simple_disjoint_sum(A,B);
    sum_disjunctive = disjunctive_sum(A,B);
    sum_disjoint = disjoint_sum(A,B);
    dp_alpha = 1;
    union_dubois_prade = dubois_prade_union(A,B,dp_alpha);

    % Intersection aggregation operations
    intersection_standard = standard_intersection(A,B);
    product_algebraic = algebraic_product(A,B);
    product_bounded = bounded_product(A,B);
    product_drastic = drastic_product(A,B);
    h_gamma = 1;
    product_himachor = himachor_product(A,B,h_gamma);
    w = 2;
    intersection_yagerclass = yagerclass_intersection(A,B,w);
    minimum_nilpotent = nilpotent_minimum(A,B);
    dp_alpha = 1;
    intersection_dubois_prade = dubois_prade_intersection(A,B,dp_alpha);

    % Averaging aggregation operations
    gamma_and=1;
    gamma_or=1;
    and_fuzzy = fuzzy_and(A,B,gamma_and);
    or_fuzzy = fuzzy_or(A,B,gamma_or);

end

% Union Functions

function f=standard_union(a,b)
    f=max(a,b);
```

```
end

function f=algebraic_sum(a,b)
    f=a+b-a.*b;
end

function f=bounded_sum(a,b)
    f=min(1,a+b);
end

function f=drastic_sum(a,b)
    f=[];
    L=length(a);
    for i=1:L
        if a(i)==0
             x=b(i);
        elseif b(i)==0
            x=a(i);
        else
            x=1;
        end
        f=[f x];
    end
end

function f=himachor_sum(a,b,gamma)
    f=[];
    L=length(a);
    for i=1:L
        x=(a(i)+b(i)-(2-gamma)*a(i)*b(i))/(1-(1-gamma)*a(i)*b(i));
        f=[f x];
    end
end

function f=yagerclass_union(a,b,w)
    f=min(1,(a.^w + b.^w).^(1/w));
end

function f=simple_disjoint_sum(a,b)
    f=max(min(a,1-b),min(1-a,b));
end

function f=disjunctive_sum(a,b)
    f=min(max(a,1-b),max(1-a,b));
end

function f=disjoint_sum(a,b)
    f=abs(a-b);
end

function f=dubois_prade_union(a,b,alpha)
    f=[];
    L=length(a);
    for i=1:L
        x=(a(i)+b(i)-a(i)*b(i)-min(min(a(i),b(i)),1-alpha))/max(max(1-a(i),1-
b(i)),alpha);
        f=[f x];
    end
end

% Intersection Functions

function f=standard_intersection(a,b)
    f=min(a,b);
```

```matlab
end

function f=algebraic_product(a,b)
    f=a.*b;
end

function f=bounded_product(a,b)
    f=max(0,a+b-1);
end

function f=drastic_product(a,b)
    f=[];
    L=length(a);
    for i=1:L
        if a(i)==1
            x=b(i);
        elseif b(i)==1
            x=a(i);
        else
            x=0;
        end
        f=[f x];
    end
end

function f=himachor_product(a,b,gamma)
    f=[];
    L=length(a);
    for i=1:L
        x=(a(i)*b(i))/(gamma+(1-gamma)*(a(i)+b(i)-a(i)*b(i)));
        f=[f x];
    end
end

function f=yagerclass_intersection(a,b,w)
    f=1-min(1,((1-a).^w + (1-b).^w).^(1/w));
end

function f=nilpotent_minimum(a,b)
    f=[];
    L=length(a);
    for i=1:L
        if (a(i)+b(i))>=1
            x=min(a(i),b(i));
        else
            x=0;
        end
        f=[f x];
    end
end

function f=dubois_prade_intersection(a,b,alpha)
    f=[];
    L=length(a);
    for i=1:L
        x=(a(i)*b(i))/max(max(a(i),b(i)),alpha);
        f=[f x];
    end
end

% Averaging Operations

function f=fuzzy_and(a,b,gamma)
    f=[];
    L=length(a);
    for i=1:L
        x=gamma*min(a(i),b(i)) + (1-gamma)*((a(i)+b(i))/2);
        f=[f x];
    end
end

function f=fuzzy_or(a,b,gamma)
    f=[];
    L=length(a);
    for i=1:L
        x=gamma*max(a(i),b(i)) + (1-gamma)*((a(i)+b(i))/2);
        f=[f x];
    end
end
```

MATLAB Code for Aggregation Operations

Execution

INPUT: Fuzzy sets A, B

Run the following statement to compute all the aggregation function values by setting the input fuzzy sets A and B accordingly.

For example, let A=[0.1,0.3,0.6], B=[0.7,0.5,0.9]

```
>>>
[standard_union,algebraic_union,bounded_sum,drastic_sum,himachor_sum,yagerclass
_union,simple_disjoint_sum,disjunctive_sum,disjoint_sum,dubois_prade_union,stan
dard_intersection,algebraic_product,bounded_product,drastic_product,himachor_pr
oduct,yagerclass_intersection,nilpotent_minimum,dubois_prade_intersection,fuzzy
_and,fuzzy_or]=fuzzy_aggregate([0.1,0.3,0.6],[0.7,0.5,0.9])
```

Command to execute the above MATLAB code

Research Assignment

Fuzzy and Genetic Algorithm

Genetic Algorithm

A genetic algorithm (or GA) is a searching algorithm used to find true or approximate solutions to optimization and search problems. It is categorized as a global search heuristics. It is based on the rules of Charles Darwin's theory of evolution. They form a particular class of evolutionary algorithms that use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover (also called recombination). Thus genetic algorithms implement the optimization strategies by simulating the evolution of species through natural selection. New population is generated using these rules in each iteration. Using a genetic algorithm one can ensure that each subsequent generation is closer to the ideal/required population. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population. If the algorithm terminates due to the latter reason, a satisfactory solution may or may not have been reached.

The algorithm usually begins by generating a random population of individuals (called chromosomes). The main idea is to maintain a population of chromosomes, which represent candidate solutions to the concrete problem being solved and that evolves over time through a process of competition and controlled variation. Each chromosome in the population has an associated fitness to determine (selection) which chromosomes are used to form new ones in the competition process. New chromosomes are created using genetic operators such as crossover and mutation. During successive iterations, called generations, chromosomes in the population are rated as a solution to the optimization problem and on the basis of these evaluations, a new population of chromosomes is formed. An evaluation or fitness function must be devised for each problem to be solved, which will be used to evaluate each chromosome for its fitness as a solution to the problem. The genetic operators used in the genetic algorithm are:

- **Selection -** Selection determines which chromosomes are to be preserved and allowed to reproduce and which ones deserve to die out. Using selection we are able to ensure that chromosomes with desirable qualities pass off these qualities to future generations. Thus, the primary objective of the selection operator is to emphasize good solutions and eliminate the bad solutions in a population while keeping the population size constant. Let's consider a population P with chromosomes C_1, C_2....C_N. The selection process produces an intermediate population P', with copies of chromosomes in P. The number of copies received from each chromosome depends on its fitness value. There are a number of ways of making this selection:

- Tournament selection
- Roulette wheel selection
- Proportionate selection
- Rank selection
- Steady-state selection

The selection procedure called stochastic universal sampling is one of the most efficient, where the number of offspring of any structure is bound by the floor and ceiling of the expected number of offspring.

- **Crossover**- Crossover is a genetic operator used for sharing information between chromosomes. It combines the features of two-parent chromosomes to form two offsprings with the possibility of them being better adapted. Crossover is not usually applied to all chromosomes present in the population. The most popular crossover process selects any two chromosomes randomly from the population and some portion of the chromosomes are exchanged between them. The selection point, the point on the chromosome from which the crossover occurs, is also selected randomly. A probability of crossover is also introduced in order to give freedom to an individual solution string to determine whether the solution would go for crossover or not. This probability is defined by the crossover rate and is denoted by p_c.
- **Mutation**- Mutation is a genetic operator arbitrarily alters one or more components of a selected structure to maintain genetic diversity from one generation of a population of genetic algorithm chromosomes to the next. It is analogous to biological mutation. The mutation alters one or more gene values in a chromosome from its initial state. In mutation, the solution may change entirely from the previous solution. Hence GA can come to a better solution by using mutation. Mutation occurs during evolution according to a user-definable mutation probability. This probability is defined by the

mutation rate and called the mutation probability(p_m). It should be set low as if it is set too high, the search will turn into a primitive random search. Though crossover has the main responsibility to search for the optimal solution, mutation is also used for this purpose

- **Elitism** - Crossover and mutation may destroy the best solution to the population pool. Elitism is the preservation of a few best solutions to the population pool. Elitism is defined in percentage or in number.

Fuzzy Genetic Algorithm

The fuzzy logic-based techniques are used for either improving GA behavior or modeling GA components. Fuzzy genetic algorithms (FGAs) is the application of GAs in various optimization and search problems involving fuzzy systems. An FGA may be defined as an ordering sequence of instructions in which some of the instructions or algorithm components may be designed with fuzzy logic-based tools. A fuzzy fitness finding mechanism guides the GA through the search space by combining the contributions of various criteria/features that have been identified as the governing factors for the formation of the clusters. A single objective optimization model sometimes cannot serve the purpose of a fitness measuring index because we are looking at multiple criteria that could be responsible for stringing together data items into clusters. This is true, not only for the clustering problem but for any problem solving using GA that involves multiple criteria. In multi-criteria optimization, the notion of optimality is not clearly defined. A solution may be the best w.r.t. one criterion but not so w.r.t. the other criteria. Pareto optimality offers a set of nondominated solutions called the P-optimal set where the integrity of each of the criteria is respected.

The FGA model consists of two models which work together:

i) **Genetic Algorithm (GA)** - It is a genetic representation for the potential solutions to the problem. It involves the generation of initial population, evaluation function to rate solutions in terms of fitness, selection and other genetic operators to generate the subsequent generations of population. For a successful run of a GA, values for parameters like population size, genetic operators and the terminating condition needs to be defined.

ii) **Fuzzy Fitness Finder (FFF)**- As the population undergoing genetic algorithm evolves to a new generation, the relatively ‘good’ solutions reproduce while the relatively ‘bad’ solutions die. An objective function distinguishes between solutions. In simple cases, there is only one criterion for optimization like maximization of profit or minimization of cost. But in real-world decision making, there are multiple objectives that need to be optimized.

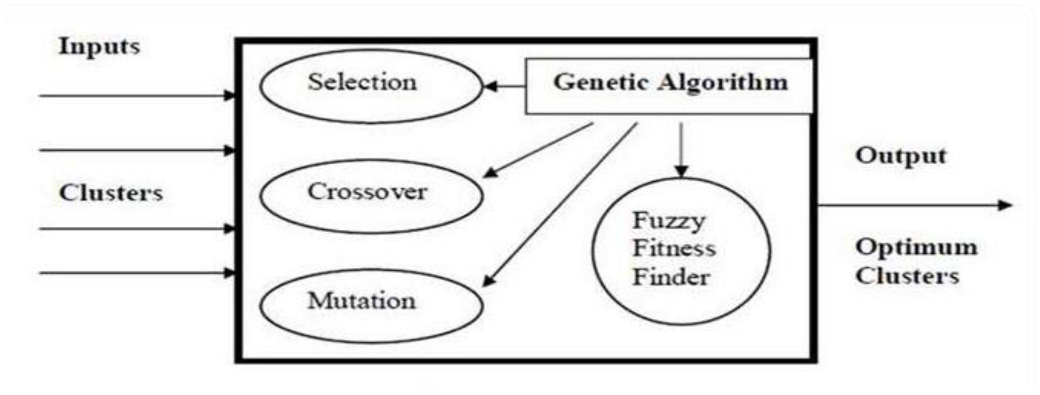


Fig. Model of a Fuzzy Genetic Algorithm (FGA)

Genetic Fuzzy Systems

The use of genetic/evolutionary algorithms (GAs) to design fuzzy systems is known as genetic fuzzy systems (GFSs). It constitutes one of the branches of the Soft Computing paradigm. The most known approach of GFS is that of genetic fuzzy rule-based systems, where some components of a fuzzy rule-based system (FRBS) are derived (adapted or learned) using a GA.

Genetic FRBS achieves optimization goals by generation or modification of the knowledge base of the FRBS. The generation/modification usually involves a tuning/learning process and it plays a major role in GFSs. Some other approaches of GFSs include genetic fuzzy neural networks and genetic fuzzy clustering, among others.

Fuzzy system and genetic algorithm

A Fuzzy System (FS) is any Fuzzy logic-based system, where the Fuzzy logic can either be used as the basis for the representation of different forms of system knowledge or model the interactions and relationships among the system variables. The automatic definition of an FS can be considered in many cases as an optimization or search process. Genetic algorithms are the best known and most widely used global search technique. They are known for finding near-optimal solutions in complex search spaces. A priori knowledge may be in the form of linguistic variables, fuzzy membership function parameters, fuzzy rules, number of rules, etc. The generic code structure and independent performance features of GAs make them suitable for incorporating a priori knowledge.The following figure illustrates this idea:

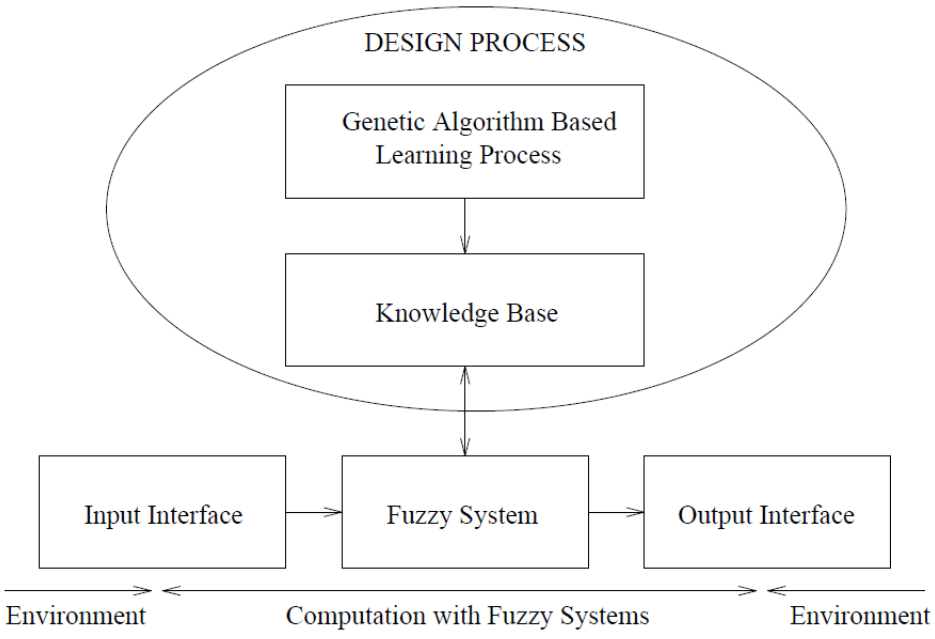


Fig. Genetic Fuzzy Systems

Design of fuzzy rule-based systems

An FRBS (regardless of whether it is a fuzzy model, a fuzzy logic controller or a fuzzy classifier) is comprised of two main components:

- **Knowledge Base** (KB), storing the available problem knowledge in the form of fuzzy rules. The KB can be obtained through human expert knowledge information or machine learning methods guided by the existing numerical information (fuzzy modeling and classification) or by the system model. It is further comprised of two components:
    - Data base (DB) - It contains the definitions of the scaling factors or functions and the membership functions associated with the labels of the fuzzy sets specifying the meaning of the linguistic terms. It also specifies the variable universes of discourse and the granularity (number of linguistic terms/labels) per variable.

    - Rule base(RB) - It is a collection of fuzzy rules.

- **Inference System**, applying a fuzzy reasoning method on the inputs and the KB rules to give a system output. It is set up by choosing a fuzzy operator for each component (conjunction, implication, defuzzifier etc.). Sometimes these operators can also be parametric and can be tuned using automatic methods.

In FRBS, there exist two different groups of optimization problems, based on whether the optimization only involves the behaviors of FRBS or whether it involves the global behavior of FRBS and an additional system. Problems such as modeling, classification, prediction, and identification form a part of the first group. The most prominent example of a second group problem is control, where the objective is to add a FRBS to a controlled system to get some desired behavior.

Classical Taxonomy of GFRBSs

Genetic FRBSs can be classified into different groups based on the components of KB, DB and/or RB, modified using the genetic algorithm:

1. **Genetic tuning of the DB**
   Several methods have been proposed in order to define the DB using GAs. Each chromosome involved in the evolution process represents different DB definitions,i.e. Each chromosome contains a coding of the whole set of membership functions giving meaning to the linguistic terms. This makes it possible to parameterize the scaling functions or the membership functions and to adapt them using GAs to deal with their parameters according to a fitness function. It is usually performed on a predefined DB definition.

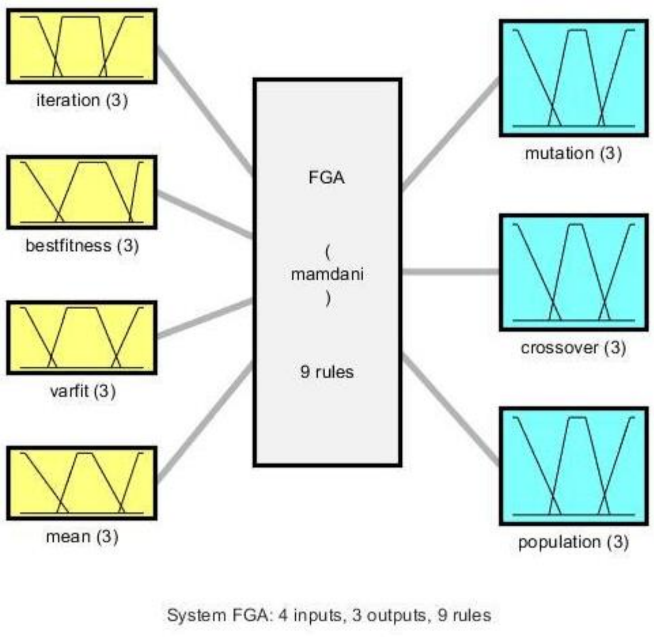
2. **Genetic learning of the RB**
   These involve the existence of a predefined collection of fuzzy membership functions giving meaning to linguistic labels i.e. a DB. On this basis GAs are applied to obtain a suitable rule base, using chromosomes that code single rules or complete rule bases. The fuzzy rules derived from this are generally of Mamdani-type.

3. **Genetic learning of the KB**

A variety of approaches are available for the genetic learning of KB (RB + DB). These include approaches with variable chromosome lengths, others coding a fixed number of rules and their membership functions and some using chromosomes encoding single control rules instead of complete KBs. It properly addresses the strong dependency existing between the RB and the DB.

MATLAB Code

The following is a MATLAB code for optimization using the genetic fuzzy system. The genetic algorithm gives inputs to the fuzzy system which evaluates them and gives defuzzified outputs. The following figures illustrate the membership functions of the input and output variables:



The rules of the fuzzy system are:

1. If (iteration is low) then (mutation is med) (crossover is med) (population is med)

2. If (iteration is high) then (mutation is low) (crossover is high) (population is low)

3. If (iteration is high) and (bestfitness is med) then (mutation is high) (crossover is low) (population is high)

4. If (bestfitness is low) then (mutation is high) (crossover is low) (population is high)

5. If (bestfitness is med) then (mutation is high) (crossover is med) (population is med)

6. If (bestfitness is high) and (varfit is med) and (mean is med) then (mutation is med) (crossover is high)

7. If (bestfitness is high) and (varfit is low) and (mean is low) then (mutation is low) (crossover is high)

8. If (bestfitness is med) then (mutation is high) (crossover is med) (population is high)

9. If (varfit is not high) and (mean is not high) then (mutation is low) (crossover is high) (population is med)

Main Program:

```
% Start of Program
clc
clear
w=[];
tic
fismat=readfis('FGA');
% Algorithm Parameters
SelMethod = 1;
CrossMethod = 1;

PopSize = 100;
MaxIteration = 1000;

CrossPercent = 70;
MutatPercent = 20;
ElitPercent = 100 - CrossPercent - MutatPercent;

CrossNum = round(CrossPercent/100*PopSize);

if mod(CrossNum,2)~=0; % makes sure that the number of crossover chromosomes is
even
    CrossNum = CrossNum - 1;
end

MutatNum = round(MutatPercent/100*PopSize);
ElitNum = PopSize - CrossNum - MutatNum;

% Problem Satement
VarMin = -100;
VarMax = 100;
DimNum = 30;
CostFuncName = @Cost_func; %select the cost function i.e. the function to be
optimized

% Initial Population
Pop = rand(PopSize,DimNum) * (VarMax - VarMin) + VarMin;  %creates initial
random population of chromosomes of size 30 between 100 and -100
Cost = feval(CostFuncName,Pop);%evaluates cost for each member of population
[Cost Indx] = sort(Cost); %sorts cost in ascending order and returns the list
of corresponding indices
```

```
Pop = Pop(Indx,:); %arranges the population according in increasing order of
their cost

% Main Loop
MeanMat = [];
it=0;
for Iter = 1:MaxIteration
    % Elitism
    ElitPop = Pop(1:ElitNum,:); %selects the elite population as we are min the
initial ElitNum elements of pop are selected

    % Cross Over
    CrossPop = [];
    ParentIndexes = SelectParents(Cost,CrossNum,SelMethod);%randomly selects
indexes to be used for crossover

    for ii = 1:CrossNum/2
        Par1Indx = ParentIndexes(ii*2-1);
        Par2Indx = ParentIndexes(ii*2);

        Par1 = Pop(Par1Indx,:);
        Par2 = Pop(Par2Indx,:);

        [Off1 Off2] = Crossover(Par1,Par2,CrossMethod);%creates two offsprings
by combining some random percent of parent 1 and parent 2
        CrossPop = [CrossPop ; Off1 ; Off2];
    end

    % Mutation
    MutatPop = rand(MutatNum,DimNum) * (VarMax - VarMin) + VarMin;%creates
mutation by creating new random chromosomes

    % New Population
    Pop = [ElitPop ; CrossPop ; MutatPop];
    Cost = feval(CostFuncName,Pop);
    [Cost Indx] = sort(Cost);
     varcost=var(Cost);%calculates variance in cost
     z=0;
   for ii=1:100
       z=Cost(ii)+z;
   end
       mean1=z/100; %average for current iteration
       ma=max(Cost); %max cost for current iteration
       mi=min(Cost); %min cost for current iteration
       mean=1-((mean1-mi)/(ma-mi)); %mean fitness for current generation
     Pop = Pop(Indx,:);%again sort the new population according to the cost



    % Algorithm Progress
   disp('-----------------------------------------------')
    BestP = Pop(1,:) %display best population for current iteration
    BestC = Cost(1)  %display best cost for current iteration
    MinMat(Iter) = Cost(1);%creates a matrix of minimum cost of each iteration
    best=1-(BestC/ma); %best fitness for the current iteration
    w=evalfis([Iter;best;varcost;mean],fismat);% sends the values of the
parameters to be evaluated by the fuzzy system
    MutatPercent = w(1);%returns mutation percent for next iteration
    CrossPercent = w(2);% returns crossover percent for next iteration
    PopSize= w(3);

    it=it+1;
    semilogy(Iter,MinMat(Iter),'r.')%plots iteration v/s min cost for the
iteration
```

```
 hold on
end
% Results
BestSolution = Pop(1,:)
BestCost = Cost(1,:)

% End of Program
toc
```

Crossover function:

```
function [Off1 Off2] = Crossover(Par1,Par2,CrossMethod)

switch CrossMethod
    case 1
        Beta1 = rand;
        Beta2 = rand;

        Off1 = Beta1*Par1 + (1-Beta1)*Par2;
        Off2 = Beta2*Par1 + (1-Beta2)*Par2;

end
```

Cost function:

```
function Cost = Cost_func(Pop)

Cost = zeros(size(Pop,1),1);

for ii = 1:size(Pop,1)
    p = Pop(ii,:);
    C = sum(p.^2);
    Cost(ii,1) = C;
end

end
```

Parents select function:

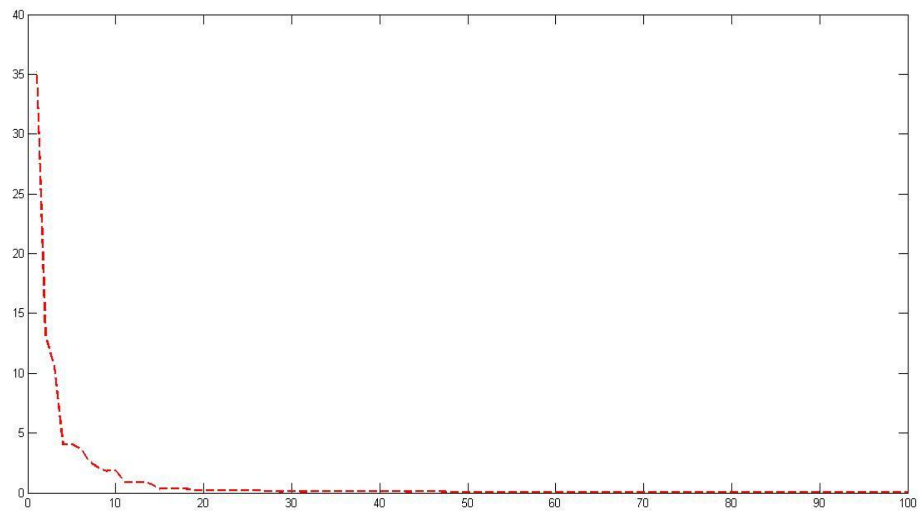
```
function ParIndexes = SelectParents(Cost,SelectionNum,SelMethod)
PopSize = size(Cost,1);

switch SelMethod
    case 1
        R = randperm(PopSize); ParIndexes = R(1:SelectionNum);
end

end
```

Output:

The code generates the following output:



It shows the minimum value of the cost function for each iteration versus the number of iterations. We can see that as the number of iterations increases the minimum value keep on decreasing thus optimizing the cost function.

References

1. Oscar Cordón (9-13 July 2007) *Genetic Fuzzy Systems Fuzzy Knowledge Extraction by Evolutionary Algorithms*, : European Centre for Soft Computing .
2. Herrera, Francisco. (1997). Genetic Fuzzy Systems: A Tutorial.
3. Dr. Rajib Kumar Bhattacharjya (7 November 2013) *Introduction To Genetic Algorithms*, Department of Civil Engineering: IIT Guwahati.
4. Francisco Herrera () *Genetic Algorithms: Optimization, Search and Learning,* Available at: *http://www.bioinf.jku.at/people/bodenhofer/private/publications/pdf/BodenhoferHerrera97.pdf* (Accessed: 15 November 2019).
5. *Fuzzy Genetic Algorithm,* Available at: *https://www.slideshare.net/khanpin2/fga-30052722* (Accessed: 15 November 2019).
6. Beliakov, Gleb, Pradera, Ana, Calvo, Tomasa (2007) *Aggregation Functions: A Guide for Practitioners*, 1 edn., : Springer-Verlag Berlin Heidelberg.
7. Class notes