Multiple Null Values Estimating In Generating Weighted Fuzzy Rules Using Genetic Simulated Annealing

Irfan Subakti
Department of Informatics, Faculty of Information Technology
Institute Technology of Sepuluh Nopember Surabaya (ITS)
Kampus ITS, Keputih, Sukolilo, Surabaya, Indonesia
yifana@gmail.com

Abstract

Several methods are proposed to estimate null values in relational database systems. Usually it's only for one or several null values. In this paper, we present a method to generate weighted fuzzy rules from relational database systems for estimating multiple null values using Genetic Simulated Annealing (GSA), where the attributes appearing in the antecedent part of generated fuzzy rules have different weights. The only constraint is at least there's one value in the column/field at the database is not null value to estimate another values (if it is a null value).

Keywords: Fuzzy sets, Genetic Simulated Annealing, null values

1. Introduction

The researchers have been proposed many methods to generate fuzzy rules from training instances [2], [5], [3], [10], [12], [13], [7] based on the fuzzy set theory [14]. In [7] we used Genetic Simulated Annealing (GSA) [8], where the attributes appearing in the antecedent parts of the generated fuzzy rules have different weights. The proposed improving method uses GSA to adjust the weights of the attributes for estimating null values in relational database systems. It can get a higher average estimated accuracy than the methods in [6]. Here, we focused only one null value at the database.

In this paper, we present a method to generate weighted fuzzy rules from relational database systems for estimating multiple null values using GSA, where the attributes appearing in the antecedent part of generated fuzzy rules have different weights. The only constraint is at least there's one value in the column/field at the database to estimate another (if it is a null value).

The rest of this paper is organized as follows. In Section 2, we briefly review the basic concepts of fuzzy sets from [14]. In Section 3, we briefly review our method for tuning the weights of the attributes appearing in the antecedent parts of the generated fuzzy rules using GSA for estimating null values in relational database systems [7]. In Section 4, we describe our method full estimating multiple null values in generating weighted fuzzy rules using GSA. Finally we conclude our research in Section 5.

2. Basic Concepts of Fuzzy Sets

In 1965, the theory of fuzzy sets is proposed by Zadeh [14]. In a fuzzy set, each element in the set is associated with a membership value between 0 and 1 described by a membership function to indicate the grade of membership of the element in the fuzzy set. There are two types of membership functions to represent fuzzy sets. One is the discrete type membership function, and the other is the continuous type membership function. Let U be the universe of discourse, $U = \{u_1, u_2, \ldots, u_n\}$. A fuzzy subset A of the universe of discourse U can be represented as follows:

$$A = \mu_A(u_1)/u_1+\mu_A(u_2)/u_2+ \ldots +\mu_A(u_n)/u_n \qquad (1)$$

where μ_A is the membership function of the fuzzy subset A, μ_A: U → [0,1], and $\mu_A(u_i)$ indicates the grade of membership of u_i in the fuzzy subset A. If the universe of discourse U is a continuous set, then the fuzzy subset A can be represented as follows:

$$A = \int_U \mu_A(u)/u, \quad u \in U \qquad (2)$$

A linguistic term can be represented by a fuzzy set represented by a membership function. In this paper, the membership functions of the linguistic terms "L," "SL," "M." "SH," and "H" of the attributes "Salary" and "Experience" in relational database system are adopted from [3] as shown in Fig. 1 and Fig. 2, respectively, where "L" denotes "Low," "SL" denotes "Somewhat Low," "M" denotes "Medium", "SH" denotes "Somewhat High," and "H" denotes "High". Assume that there is an employee whose salary is 48,000 dollars per month, and assume that his working experience is 4 years, then according to Fig. 1, we can see that the degree of membership that his salary (i.e., 48,000 dollars) belonging to the linguistic terms "Medium" (M) and "Somewhat High" (SH) are 0.7 and 0.3, respectively.

According to Fig. 2, we can see that the degrees of membership that his experience (i.e., 4 years) belonging to the linguistic terms of "Somewhat Low" (SL) and "Medium" (M) are 0.5 and 0.5, respectively. Let x be a real value, and let X and Y be two linguistic terms. Assume that the degrees of membership of x belonging to X and Y are a and b, respectively, where a ∈ [0,1] and b ∈ [0,1]. Based on [3] and [13], if a ≥ b, then the real employee's salary (i.e., 48,000 dollars) is fuzzified into M/0.7, and his experience (i.e., 4 years) is fuzzified into SL/0.5. If p is a nonnumeric datum, where p ∈ {Bachelor, Master, PhD}, then according to [3], p is fuzzified into p/1.0.

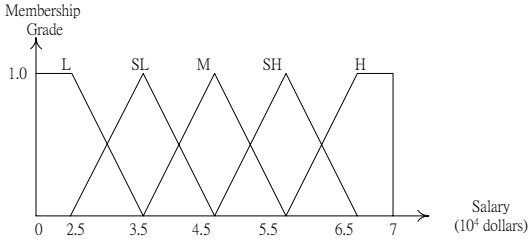

Figure 1. Membership functions of the linguistic terms of the attribute "Salary"

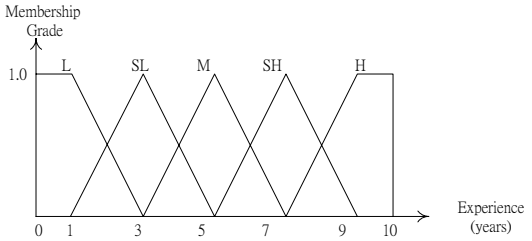

Figure 2. Membership functions of the linguistic terms of the attribute "Experience"

3. Tuning the Weight of the Attributes Using GSA [7]

3.1 Format of a Chromosome

Taken from [6], let's consider a relation of a relational database shown in Table 1. Based on Figs. 1 and 2, the values of attributes "Degree" and "Experience" shown in Table 1 can be fuzzified into Table 2. First, we define the format of a chromosome as shown in Fig. 4, where the value of each gene in a chromosome is a real value between zero and one, and the 13th gene labeled "B-L" denotes the fuzzified values of the attributes "Degree" and "Experience" are "Bachelor" (B) and "Low" (L), respectively.

Table 1. Relation in a relational database

EMP-ID	Degree	Experience	Salary
S1	Ph.D.	7.2	63,000
S2	Master	2.0	37,000
S3	Bachelor	7.0	40,000
S4	Ph.D.	1.2	47,000
S5	Master	7.5	53,000
S6	Bachelor	1.5	26,000
S7	Bachelor	2.3	29,000
S8	Ph.D.	2.0	50,000
S9	Ph.D.	3.8	54,000
S10	Bachelor	3.5	35,000
S11	Master	3.5	40,000
S12	Master	3.6	41,000
S13	Master	10.0	68,000
S14	Ph.D.	5.0	57,000
S15	Bachelor	5.0	36,000
S16	Master	6.2	50,000
S17	Bachelor	0.5	23,000
S18	Master	7.2	55,000
S19	Master	6.5	51,000
S20	Ph.D.	7.8	65,000
S21	Master	8.1	64,000
S22	Ph.D.	8.5	70,000

Table 2. Fuzzified results of the value of the attributes "Degree" and "Experience"

EMP-ID	Degree	Experience	Salary
S1	Ph.D./1.0	SH/0.9	63,000
S2	Master/1.0	L/0.5	37,000
S3	Bachelor/1.0	SH/1.0	40,000
S4	Ph.D./1.0	L/0.9	47,000
S5	Master/1.0	SH/0.75	53,000
S6	Bachelor/1.0	L/0.75	26,000
S7	Bachelor/1.0	SL/0.65	29,000
S8	Ph.D./1.0	L/0.5	50,000
S9	Ph.D./1.0	SL/0.6	54,000
S10	Bachelor/1.0	SL/0.75	35,000
S11	Master/1.0	SL/0.75	40,000
S12	Master/1.0	SL/0.7	41,000
S13	Master/1.0	H/1.0	68,000
S14	Ph.D./1.0	M/1.0	57,000
S15	Bachelor/1.0	M/1.0	36,000
S16	Master/1.0	SH/0.6	50,000
S17	Bachelor/1.0	L/1.0	23,000
S18	Master/1.0	SH/0.9	55,000
S19	Master/1.0	SH/0.75	51,000
S20	Ph.D./1.0	SH/0.6	65,000
S21	Master/1.0	H/0.55	64,000
S22	Ph.D./1.0	H/0.75	70,000

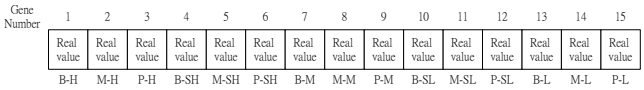

Figure 4. Format of a chromosome

From Fig. 4, we can see that each chromosome represents a combination of the weights of attributes, and it is a string of the weights of the attributes which will be used to estimate null values in relational databases systems. A population contains a set of chromosomes, and we can arbitrary set the number of chromosomes in a population. A chromosome consists of 15 genes. Because the total weights of attributes must be equal to one, the weight of attribute "Experience" must equal to one minus the weight of attribute "Degree". For example, let's see Fig. 5.

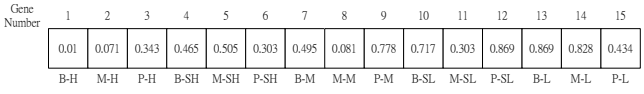

Figure 5. Example of a chromosome

Therefore, the content of the chromosome shown

in Fig. 5 can be translated into the following 15 rules.

Rule 1: **IF** Degree=Bachelor **AND** Experience=High, **THEN** the Weight of Degree=0.010 **AND** the Weight of Experience=0.99.

Rule 2: **IF** Degree=Master **AND** Experience=High, **THEN** the Weight of Degree=0.071 **AND** the Weight of Experience=0.929.

…

Rule 15: **IF** Degree=PhD **AND** Experience=Low, **THEN** the Weight of Degree=0.434 **AND** the Weight of Experience=0.566.

3.2 Calculation of the Fitness Degree

Taken from [6], assume that there are n tuples T_1, T_2, …, T_n in a relation R of a relational database system, where the value of the attribute "Salary" of tuple T_i is denoted as "T_i.Salary." Let "ET_i.Salary" denotes the estimated value of T_i.Salary. In order to derive the value of ET_i.Salary, it must find a tuple T_j which is closest to the tuple T_i regarding the values of the attributes "Degree" and "Experience."

Table 3. Degree of similarity between the values of the attribute "Degree"

	Bachelor	Master	PhD
Bachelor	1	0.6	0.4
Master	0.6	1	0.6
PhD	0/4	0.6	1

Based on a fuzzy similarity matrix, it can obtain the degree of similarity between two nonnumeric values. For example, from Table 3, we can see that the degree of similarity between the degrees "Bachelor" and "PhD" is 0.4. From [4], we can see that the ranks of the terms "Bachelor," "Master," and "PhD" are 1, 2 and 3, respectively. That is:

$$\text{Rank(Bachelor)} = 1$$
$$\text{Rank(Master)} = 1$$
$$\text{Rank(PhD)} = 1$$

Let X be a nonnumeric attribute. Based on the value T_i.X of the attribute X of tuple T_i and the value T_j.X of the attribute X of tuple T_j, where $i \neq j$, the degree of closeness Closeness(T_i, T_j) between tuples T_i and T_j can be calculated by (3) or (4), where Weight(T_j.Degree) and Weight(T_j.Experience) denote the weights of the attributes "Degree" and "Experience," respectively, obtained from the fuzzified values of the attributes "Degree" and "Experience" of tuple T_j, derived from a chromosome.

If $\text{Rank}(T_i.X) \geq \text{Rank}(T_j.X)$ then

$$\text{Closeness}(T_i, T_j) = \text{Similarity}(T_i.X, T_j.X) \times \text{Weight}(T_j.\text{Degree}) + \frac{T_i.Experience}{T_j.Experience} \times \text{Weight}(T_j.\text{Experience}) \quad (3)$$

If $\text{Rank}(T_i.X) < \text{Rank}(T_j.X)$ then

$$\text{Closeness}(T_i, T_j) = 1/\text{Similarity}(T_i.X, T_j.X) \times \text{Weight}(T_j.\text{Degree}) + \frac{T_i.Experience}{T_j.Experience} \times \text{Weight}(T_j.\text{Experience}) \quad (4)$$

where Similarity(T_i.X, T_j.X) denotes the degree of similarity between T_i.X and T_j.X, and its value is obtained from a fuzzy similarity matrix of the linguistic terms of the attribute X defined by a domain expert.

Let T_i, T_j and T_k be three tuples in a relational database. Assume that the degree of closeness between tuple T_i and T_j is denoted as Closeness(T_i, T_j), and the degree of closeness between tuples T_i and T_k is denoted as Closeness(T_i, T_k). Let $x = |\text{Closeness}(T_i, T_j)-1|$ and let $y = |\text{Closeness}(T_i, T_k)-1|$. If $x < y$, then tuple T_j is more close to tuple T_i and all the other tuples in the relational database have been calculated, the tuple whose closeness degree with respect to tuple T_i is closest to 1.0 is regarded as closest to tuple T_i, where $1 \leq i \leq n$.

After calculating the degrees of closeness of the other tuples in the database with respect to tuple T_i, the system will pick a tuple which is closest to tuple T_i. Assume that tuple T_j is closest to tuple T_i, then we can calculate the estimated value "ET_i.Salary" of the attribute "Salary" of tuple T_i as follows:

$$ET_i.\text{Salary} = T_i.\text{Salary} \times \text{Closeness}(T_i, T_j) \quad (5)$$

where T_i.Salary denotes the value of the attribute "Salary" of tuple T_j.

In the same way, it can calculate the estimated values of the attribute "Salary" of all the tuples in a relational database. After we get the estimated values of the attribute "Salary" of all tuples in a relational database, it can calculate the estimated error of each tuple by (6), where Error_i denotes the estimated error between the estimated value ET_i.Salary of the attribute "Salary" of tuple T_i and the actual value T_i.Salary of the attribute "Salary" of tuple T_i

$$\text{Error}_i = \frac{ET_i.Salary - T_i.Salary}{T_i.Salary} \quad (6)$$

Let Avg_Error denotes the average estimated error of the tuples based on the combination of weights of the attributes derived from the chromosome, where

$$\text{Avg\_Error} = \frac{\sum_{i=1}^{n} Error_i}{n} \quad (7)$$

Then, it can obtain the fitness degree of this chromosome as follows [7]:

$$\text{Fitness Degree} = 1 - \text{Absolute (Avg\_Error)} \quad (8)$$

4. Estimating Multiple Null Values in Relational Database Systems Using GSA

GSA proposed by Koakutsu *et al.* [8]. It generates the seeds of Simulated Annealing (SA) sequentially, that is the seeds of a SA local search depends of the best-so-far solutions of all previous SA local searches. This sequentially approach seems to generate better child solutions.

GSA uses fewer crossover operations since it only uses crossover operations when the SA local search reaches a flat surface and it is time to jump in the solution space.

GSA starts with a population $X = \{x_1, \ldots, X_{Np}\}$ and repeatedly applies 3 operations: SA-based [11] local search, GA-based [1] [9] crossover operation, and population update. SA-based local search produces a candidate solution x' by changing a small fraction of the state of x. The candidate solution is accepted as the new solution with probability $\min\{1, e^{-\Delta f/T}\}$. GSA preserves the local best-so-far solution x^*_L during the SA-based local search. When the search reaches a flat surface or the system is frozen, GSA produces a large jump in the solution space by using GA-based crossover. GSA picks up a pair of parent solutions x_j and x_k at random from the population X such that $f(x_j) \neq f(x_k)$, applies crossover operator, and then replace the worst solution x_i by the new solution produced by the crossover operator. At the end of each SA-based local search, GSA updates the population by replacing the current solution x_i by the local best-so-far solution x^*_L. GSA terminates when the CPU time reaches given limit, and reports the global best-so-far solution x^*_G.

We implemented GSA pseudo-code [8] by the code algorithm depicted at Figs. 6-9 below.

```
Procedure EvaluationAndBestSelection
    {find the best solution among population. Also it initializes
LocalBestChromosomeSoFar and GlobalBestChromosomeSoFar:
    X ← {x1, ..., XNp}; {initialize population}
    x*L ← the best solution among X; {initialize local best-so-far}
    x*G ← x*L {initialize global best-so-far}
    FitnessDegreeEval ← FitnessDegree from global best-so-far
    }
    for i:= 1 to number-of-generations do begin
      T ← T0;
      EvaluationAndWorstSelection; {select the worst solution xi from X}
      CrossOver; {select two solutions xj, xk from X such that f(xj) ≠ f(xk):
        xi ← Crossover(xj, xk);
      }
      Mutation; {update local best-so-far if value is better
       repeat
         for i:= 0 to number-of-mutation do begin
          f(xi) ← Get Fitness Degree from chromosome before mutation
          x' ← Mutate(xi)
          f(x') ← Get Fitness Degree from chromosome after mutation
          Δf ← f(xi) - f(x')
          r ← random number between 0 and 1
          ft ← f(x')
          if (Δf >= 0) or (r >= exp(-Δf/T)) then begin
            xi ← x';
            ft ← f(xi);
          end;
          if (ft >= FitnessDegreeEval) then begin
            x*L ← xi; {update local best-so-far}
            FitnessDegreeEval ← ft
            FDLocalBestSoFar ← ft {Get local best Fitness Degree}
          end
         end
         T ← T * α; {lower temperature}
       until T <= FrozenValue;
      }
      CountCloseness(x*L); {get FD from LocalBestChromosomeSoFar}
      AvgError:= AvgError / NumData;
      FDLocalBestSoFar:= 1 - AvgError;
      CountCloseness(x*G); {get FD from GlobalBestChromosomeSoFar}
      AvgError:= AvgError / NumData;
      FDGlobalBestSoFar:= 1 - AvgError;
      if FDLocalBestSoFar >= FDGlobalBestSoFar then begin
        x*G ← x*L; {update global best-so-far}
        FitnessDegreeEval:= FDGlobalBestSoFar;
      end;
      xi ← x*L; {update population}
    end;
```

Figure 6. GSA code algorithm

```
Procedure CountClosenessValue
    AvgError:= 0.0;
    for i:= 0 to NumData - 1 do begin  {base on all data available}
      BestClosenessEval:= MaxInt;
      IdxClosestCloseness:= i;
      for j:= 0 to NumData - 1 do
        if i <> j then begin
          if Rank(Ti.X) ≥ Rank(Tj.X) then begin
            ClosenessE(Ti,Tj)= Similarity(Ti.X,Tj.X) × Weight(Tj.Degree) +
              (Ti.Experience/Tj.Experience) × Weight(Tj.Experience);
          end
          else begin {If Rank(Ti.X) < Rank(Tj.X)}
            ClosenessE:= 1/Similarity(Ti.X,Tj.X) × Weight(Tj.Degree) +
              (Ti.Experience/Tj.Experience) × Weight(Tj.Experience);
          end;
          {find a tuples which is closest to 1.0 as a}
          {closest tuple to tuple Ti}
          ClosestCloseness:= Abs(1 - ClosenessE);
          if ClosestCloseness <= BestClosenessEval then begin
            BestClosenessEval:= ClosestCloseness;
            IdxClosestCloseness:= j;
          end;
        end;
        {Then we find Estimated Salary and Error for every record}
        {if this record was null value, so we must find}
        {another record that closest to 1}
      if IsNullValue(i) and IsNullValue(IdxClosestCloseness) then begin
        PreferIdx:= GetPreferIdx;
        ETi.Salary:= Ti. Salary × GetClosenessValue(PreferIdx);
        if Tprefer-index.Salary <> 0 then
          Errori:= (ETi.Salary – Tprefer-index.Salary)/ Tprefer-index.Salary
      end
      else begin
        ETi.Salary:= Ti. Salary × GetClosenessValue(IdxClosestCloseness);
        if Ti.Salary <> 0 then
          Errori:= (ETi.Salary – TiSalary)/ TiSalary
      end;
      AvgError:= AvgError + Abs(Errori);
    end;
```

Figure 7. Procedure CountCloseness code algorithm

```
function GetClosenessValue(Idx)
  Result ← find value in ClosenessE which have the same index with Idx
```

Figure 8. Function GetClosenessValue code algorithm

```
function GetPreferIdx
  Result ← find value in ClosenessE that closest to 1, and it's not null
value
```

Figure 9. Function GetPreferIdx code algorithm

In our previous paper [7], we only concerning about how to perform GSA to prove that it's work better than if the system used GA. Here, we estimate many null values at the database.

Recalling procedure CountCloseness, as depicted

at Fig. 7 above, we consider the part at Fig. 10 below.

```
…
…
   {Then we find Estimated Salary and Error for every record}
   {if this record was null value, so we must find}
   {another record that closest to 1}
 if IsNullValue(i) and IsNullValue(IdxClosestCloseness) then begin
   PreferIdx:= GetPreferIdx;
   ETi.Salary:= Ti. Salary × GetClosenessValue(PreferIdx);
   if Tprefer-index.Salary <> 0 then
     Errori:= (ETi.Salary – Tprefer-index.Salary)/ Tprefer-index.Salary
 end
 else begin
   ETi.Salary:= Ti. Salary × GetClosenessValue(IdxClosestCloseness);
   if Ti.Salary <> 0 then
     Errori:= (ETi.Salary – TiSalary)/ TiSalary
 end;
 AvgError:= AvgError + Abs(Errori);
end;
```

Figure 10. Part considering at Procedure CountCloseness

As an example, as depicted at Table 4, where many values are null, and with GSA we process to obtain these null values from other value.

Table 4. Relation in a relational database containing multiple null values

EMP-ID	Degree	Experience	Salary
S1	Ph.D.	7.2	63,000
S2	Master	2.0	null
S3	Bachelor	7.0	40,000
S4	Ph.D.	1.2	47,000
S5	Master	7.5	null
S6	Bachelor	1.5	26,000
S7	Bachelor	2.3	29,000
S8	Ph.D.	2.0	50,000
S9	Ph.D.	3.8	54,000
S10	Bachelor	3.5	35,000
S11	Master	3.5	null
S12	Master	3.6	41,000
S13	Master	10.0	null
S14	Ph.D.	5.0	57,000
S15	Bachelor	5.0	36,000
S16	Master	6.2	50,000
S17	Bachelor	0.5	23,000
S18	Master	7.2	55,000
S19	Master	6.5	51,000
S20	Ph.D.	7.8	65,000
S21	Master	8.1	64,000
S22	Ph.D.	8.5	null

Because there's a checking process with regarding to null values, so we can set one or many null values that we want to estimate. This process performs in function GetPreferIdx as described previous.

Of course as a constraint, there is at least one value in column/field SALARY to estimate another (if it is a null value).

We run this program base on parameters, as depicted at Table 5, each for 10 times.

Table 5. Parameters of experiment

Sixe of Popu-lation	# Gene-ration	Muta-tion Rate	Initial Tempe-rature	Alpha	Frozen Value
60	300	0.01	100	0.7	0.00001

The results of experiments are depicted in Table 6, and the summaries from all experiments are depicted in Table 7.

Table 6. Two results of the running program

(a)

Index of Null Values = 0. It means row/tuple 1st in relational database.

Running #	Avg. Estimated Error	Total Time
1	0.009122101	0h:22m:56s:46ms
2	0.009122101	0h:22m:47s:188ms
3	0.005950415	0h:22m:43s:484ms
4	0.005950415	0h:22m:45s:500ms
5	0.005608084	0h:22m:40s:922ms
6	0.005297164	0h:22m:39s:219ms
7	0.005297164	0h:22m:42s:437ms
8	0.0035237	0h:23m:13s:922ms
9	0.0035237	0h:22m:47s:47ms
10	0.0035237	0h:22m:46s:281ms
Min	**0.0035237**	**0h:22m:39s:219ms**
Average	**0.005691855**	**0h:22m:48s:205ms**
Max	**0.009122101**	**0h:23m:13s:922ms**

(b)

Index of Null Values = 0, 1. It means row/tuple 1st and 2nd in relational database.

Running #	Avg. Estimated Error	Total Time
1	0.005932516	0h:23m:0s:906ms
2	0.004748638	0h:23m:1s:281ms
3	0.004748638	0h:22m:59s:954ms
4	0.004376414	0h:22m:56s:937ms
5	0.004307416	0h:22m:53s:797ms
6	0.004307416	0h:22m:46s:844ms
7	0.002597917	0h:22m:52s:484ms
8	0.002543098	0h:22m:53s:688ms
9	0.002543098	0h:22m:53s:406ms
10	0.002036744	0h:22m:56s:109ms
Min	**0.002036744**	**0h:22m:46s:844ms**
Average	**0.003814189**	**0h:22m:55s:541ms**
Max	**0.005932516**	**0h:23m:1s:281ms**

Table 7. Summaries of all experiments

# Null Values	Avg. Estimated Error	Total Time
1	0.005691855	0h:22m:48s:205ms
2	0.003814189	0h:22m:55s:541ms
3	0.004250243	0h:23m:9s:162ms
4	0.001986583	0h:23m:21s:841ms
5	0.002852969	0h:23m:49s:88ms
6	0.007583199	0h:24m:52s:142ms
7	0.009365515	0h:25m:52s:181ms
8	0.009231593	0h:28m:57s:550ms

9	0.00619153	0h:31m:24s:880ms
10	0.011827634	0h:28m:33s:502ms
11	0.009823456	0h:31m:6s:367ms
12	0.015330053	0h:33m:39s:134ms
13	0.009790754	0h:39m:18s:45ms
14	0.013608027	0h:37m:59s:381ms
15	0.025640814	0h:42m:50s:858ms
16	0.011122631	0h:47m:47s:817ms

We can see from the result that we successfully estimate multiple null values.

Once again, as in our previous paper [7] apparently, the only constraint is time. GSA obviously takes longer time. More null values have to estimate, the longer time the result will be.

5. Conclusions

In this paper, we have presented a method to generate weighted fuzzy rules for estimating multiple null values in relational database systems using GSA, where the attributes appearing in the antecedent parts of the generated fuzzy rules have different weights.

The only constraint of this method is at least there's one value in the column/field at the database is not null value to estimate another values (if it is a null value).

6. References

[1] Brown, D., Huntley, C. and Spillane, A. , "A parallel genetic heuristic for the quadratic assignment problem," *Proc. 3rd Int. Conf. Genetic Algorithms*, 1989, pp.406-415.

[2] Burkhardt, D.G and Bonissone, P.P., "Automated fuzzy knowledge base generation and tuning," *Proc. 1992 IEEE Int. Conf. Fuzzy Systems*, San Diego, CA, 1992, pp. 179-419.

[3] Chen, S.M. and Yeh, M.S., "Generating fuzzy rules from relational database systems for estimating null values," *Cybern. Syst.*, vol. 28, no. 8, 1997, pp. 695-723.

[4] Chen, S.M. and Chen, H.H., "Estimating null values in the distributed relational database environment," *Cybern. Syst.*, vol. 31, no. 8, 2000, pp. 851-871.

[5] Chen, S.M., Lee, S.H. and Lee, C.H., "A new method for generating fuzzy rules from numerical data for handling classification problems," *Appl. Art. Intel.*, vol. 15, no. 7, 2001, pp. 645-664.

[6] Chen, S.M. and Huang, C.M., "Generating weighted fuzzy rules from relational database systems for estimating null values using genetic algorithms," *IEEE Trans.On Fuzzy Systems*, vol. 11, no. 4, 2003, pp. 495-506.

[7] Irfan, S., "Genetic Simulated Annealing For Null Values Estimating in Generating Weighted Fuzzy Rules From Relational Database Systems," *ICTS 2005*, 2005.

[8] Koakutsu, S., Kang, M. and Dai, W.W.-M, "Genetic simulated annealing and application to non-slicing floorplan design," *Proc. 5th ACM/SIGDA Physical Design Workshop*, (Virginia, USA), 1996, pp. 134-141.

[9] Lin, T.-T., Kao, C.-Y. and Hsu, C.-C., "Applying the genetic approach to simulated annealing in solving some NP-Hard problems," *IEEE Trans. System, Man, and Cybernetics*, vol. 23, no. 6, 1993, pp.1752-1767.

[10] Lin, H.L. and Chen, S.M., "Generating weighted fuzzy rules from training data for handling fuzzy classification problems," *Proc. 2000 Int. Computer Symp.: Workshop Artificial Intelligence*, Chiayi, Taiwan, R.O.C., 2000, pp. 11-18.

[11] Sigrag, D. and Weisser, P., "Toward a unified thermodynamic genetic operator," *Proc. 2nd Int. Conf. Genetic Algorithms*, 1987, pp.116-122.

[12] Tsukimoto, H., "Extracting rules from trained neural networks," *IEEE Trans. Neural Networks*, vol. 11, 2000, pp. 377-389.

[13] Wang, L.X. and Mendel, J.M., "Generating fuzzy rules by learning from examples," *IEEE Trans. Syst., Man, Cybern.*, vol. 22, 1992, pp. 1414-1427.

[14] Zadeh, L.A., "Fuzzy sets," *Inform. Control*, vol. 8, 1965, pp. 338-353.