Bisection Algorithms for Solving λ-Fuzzy Measures

Jih-Chang Wang[1] Ting-Yu Chen[2]

[1]Department of Information Management, Chang Gang University, Taiwan
[2]Department of Business Administration, Chang Gung University, Taiwan

Abstract

The theory of fuzzy measures has a great potential for real world applications, but limited by the lack of suitable identifying methods. This research proposes a bisection algorithm based on Sugeno's λ-fuzzy measures. The proposed method is simple enough to suit the practical applications for the required data is similar to the traditional weighted-sum method. The computing complexity of this method is O(n), and it is efficient to meet the huge computations in practical.

Keywords: Fuzzy measure; bisection algorithm; computing complexity

1. Introduction

The theory of fuzzy measures has a great potential for applications of subjective evaluation, information fusion, multiple criteria decision making [Wang and Klir, 1992; Grabisch, 1995]. However, this potential has not been fully utilized due to the lack of identifying methods for constructing fuzzy measure from empirical data [Yuan and Klir, 1996]. The crux is the amount of required coefficients growing exponentially with problem size n (roughly 2^n). The existed identifying methods are based on either learning data, or semantic estimations, or both, but this problem is not yet solved in a fully satisfactory way [Grabisch, 1995].

Sugeno proposed a λ-fuzzy measure satisfying the λ-additive axiom [Sugeno and Terano, 1977; Wang and Klir, 1992]. The λ-fuzzy measure reduces the difficulty of identification effectively, and has plenty applications recently, including pattern recognition, speaker verification, and public attitude analyzing. Some studies estimate this single parameter of λ-fuzzy measure from learning data by the soft-computing methods like genetic algorithm [Lee and Leekwang, 1995], neural networks [Wang and Wang, 1997]. But collecting subjective evaluations of each information source by questionnaire is an easier approach, and this approach reduces the identifying problem to an n-1 degree polynomial (see function G in fig. 1 and explanation of section 2 for detail). There are many available methods [Wierzchoń, 1983], and the Keller and Osborns' Newton's method seems the simplest among them for practical uses.

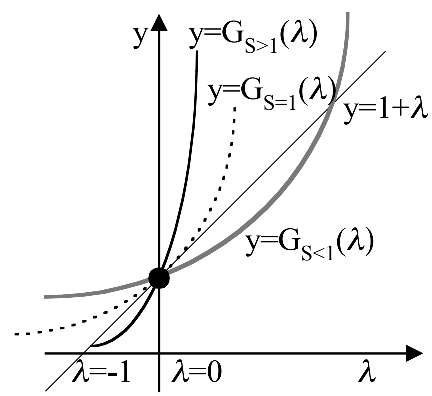

Fig. 1: The equation for identification (S is the summation of input).

However, the Newton's method is sensitive to initial solution, and a feasible initial solution is not easy to locate in the polynomial of figure 1. A bad initial solution of $G'(\lambda)\approx 1$ will lead the positive or negative infinite; and $G'(\lambda)<1$ when S<1, $G'(\lambda)>1$ when S>1 will mislead the searching sequence back to the trivial solution λ=0. Besides, an over-estimated initial solution causes a slow converging sequence. The last, the Newton's method requires the first-order differentiation having computing complexity O(n^2).

This research proposes a simple method based on bisection search and a linear transformation of traditional one (see function H in fig. 2 and explanation of section 2 for detail). The properties of this method are listed below. (1) The input is simple as the traditional weighted-sum method, and the required data is n only. (2) The executing time is short in practice and increases linearly with problem size only. A analysis of computing complexity O(n) is given in section 4. (3) The robustness is guaranteed and discussed in section 5. (4) The implementation is easy, and the source code of an executable program is opened in the appendix.

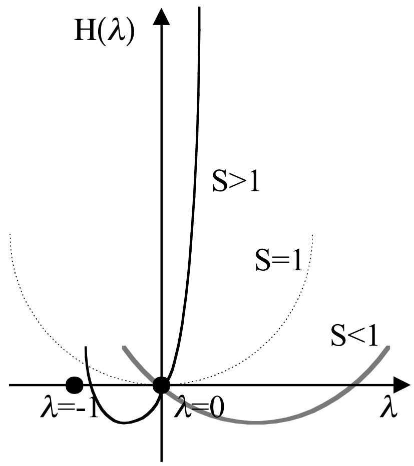

Fig. 2: The proposed identifying method based on function H(λ)=G(λ)-1-λ.

2. λ-Fuzzy Measures

In this section, some notations and required properties for our method are given. Most of them have been discussed deeply in some pioneering studies. We refer the reader to Leszczyński et al. [1986], Wang and Klir [1992].

Definition 2.1. Let **X**=$\{x_1, x_2, \ldots, x_n\}$ be a nonempty finite set, **P** is the power set of **X**. A (regular) λ-fuzzy measure μ defined on (**X**, **P**) is a set function satisfying the conditions:

[1] $\mu(\mathbf{\Phi})=0$, $\mu(\mathbf{X})=1$, where $\mathbf{\Phi}$ is the empty set; (boundary conditions)

[2] If **A**, **B**$\in$**P** and **A** $\cup$ **B**=$\mathbf{\Phi}$ then $\mu(\mathbf{A} \cup \mathbf{B})=\mu(\mathbf{A})+\mu(\mathbf{B})+\lambda\mu(\mathbf{A})\mu(\mathbf{B})$, $\lambda \in (-1, +\infty)$; (monotonicity)

Proposition 2.2. Denote $g_i=\mu(\{x_i\})$, the fuzzy measure μ satisfies the bounding conditions: (Leszczyński et al., 1986, pp.148-150; Wang and Wang,1997, p.187; Wang and Klir, 1992, pp.40-46)

[1] $g_i \in [0, 1]$ for all i;

[2] if there exists g_i=1, then g_j=0 for any $j \neq i$;

[3] if g_i<1 for all i, then there are at least two of them being positive.

Extending definition 2.1, we obtain an equation for identifying parameter λ:

$$G(\lambda)=\begin{cases}\prod_{i=1}^{n}(1+\lambda g_i)=1+\lambda & \lambda \neq 0 \\ 1 & \lambda = 0\end{cases}$$

Proposition 2.3. Denote $S=\sum_{i=1}^{n} g_i$, if $\lambda \in (-1, +\infty)$ then $G'(\lambda)>0$, $G''(\lambda)>0$, and $G'(0)$=S for $n \geq 2$ (lemma 2.4 and corollary 2.5 of Leszczyński et al. 1986).

Theorem 2.4. The parameter λ can be determined uniquely from G(λ) (theorem 3.6 of Wang and Klir, 1992):

$$\begin{cases}\lambda > 0 & \text{when S} < 1 \\ \lambda = 0 & \text{when S} = 1 \\ -1 < \lambda < 0 & \text{when S} > 1\end{cases}$$

Proposition 2.5. Let H(λ)=G(λ)-1-λ=0.

[1] When S>1, if exist u, v, $-1<u<v<0$, and H(u)>0, H(v)<0, then there exists a unique w, such that H(w)=0 and $u<w<v$.

[2] When S<1, if exist u, v, $u>v>0$, and H(u)>0, H(v)<0, then there exists a unique w, such that H(w)=0 and $u>w>v$.

[*Proof*] From the continuity of function H and uniqueness of theorem 2.4, if we can obtain a closed range having the two endpoints with different signs, then a solution exists uniquely from intermediate theory (see fig. 2).

Proposition 2.6. If there exists a small ε, and ε>0, then (1) H(-1)>0 when all g_i<1; (2) H(-ε)<0 when S>1; (3) H(ε)<0 when S<1.

[*Proof*]

(1) H(-1)=(1-g_1)(1- g_2)…(1- g_n)-1+1=(1-g_1)(1- g_2)…(1- g_n)>0 ($\because 0 \leq g_i < 1, \forall i$).

(2) $H'(\lambda)=G'(\lambda)-1>-1$, $H''(\lambda)=G''(\lambda)>0$, $H'(0)=G'(0)-1$=S-1 (from proposition 2.3), and H(0)=G(0)-1=0. When S>1, $H'(0)$=S-1>0, therefore H(-ε)<H(0)=0.

(3) Similar as the above, $H'(0)$=S-1<0 when S<1, therefore H(ε)<H(0)=0.

3. The Algorithm

Fig. 3 shows the proposed algorithm. There are three primary steps consisting of validating the input data, determining a closed range of λ, and conducting a bisection search.

Validate the input data

1-1. If n<2, then return error

1-2. If any g_i<0 or g_i>1 for I=1..n, then return error

1-3. If any $g_i<\varepsilon^{1/2}$ for i=1..n, then let g_i=0

1-4. Let c0 be the count of g_i=0, c1 be the count of g_i=1, for i=1..n

1-5. If c1>1, then return error

1-6. If c1=1 and c0<n-1 then return error

1-7. If n-c0<2, then return error

Determine a closed range of λ (S is the summation of g_i):

2-1. If |S-1|<ε, then return 0

2-2. If S>1, then { let u=-1, v=-ε, and go to step 3}

2-3 If S<1, then find a k≥1 and H(2^k)>0:
Let k=0; repeat let k=k+1 until H(2^k)>0
Let $v=\varepsilon$, $u=2^k$

Perform a bisection search between *v* and *u*:

```
Repeat  Let w=(u+v)/2
        If u=w or v=w then break-repeat
        If H(w)>0 then let u=w; else let v=w
Until   (|u-v|<d)
Return w
```

Fig. 3: The detail algorithm.

4. Computing Complexity

There are two explicit loops and one implicit loop in this method. The first explicit loop is to find the range of λ when S<1 (step 2-3). The second explicit loop is to conduct a bisection search within a closed range (step 3). The implicit loop is to compute function H each explicit loop. The following discussions begin with the implicit loop.

(1) THE IMPLICIT LOOP OF COMPUTING FUNCTION H

By definition, $H(\lambda) =(1+\lambda*g_1) *(1+\lambda*g_2) *\ldots *(1+\lambda*g_n) -1-\lambda$. Let c1 be the fixed overhead, and roughly c2 be the executing time of each input; then, we can denote the executing time of function H as O1(n) below.

$O1(n)=c1+c2*n$

(2) THE EXPLICIT LOOP OF FINDING A CLOSED RANGE

If S<1, this loop can be stated as finding a k and $H(2^k)>0$. If S>1, then k=0. Whatever S is, the value of k is predefined as a specific set of g_i is given. We denote the executing time by O2(n) below. There are two constants in O2(n), say c3, c4, where c3 is the fixed parts, and c4 is the executing time in the loop excluding the computing time of function H.

$O2(n)=c3+(c4+c1+c2*n)*(k+1)$

(3) THE EXPLICIT LOOP OF CONDUCTING A BISECTION SEARCH

There are 2 normal conditions to halt this loop, including the absolute width of searching range ($|u-v|$) is smaller than e1, or the distance of function H ($|H(w)|$) is smaller than e2. The worst case above happens when λ is large, and only the first condition can be activated. Then, the algorithm has to equally divide the range $(0, 2^k)$ k+1 times and narrow-down the searching width to 1, and equally divide $log_2(1/e1)$ times again to approach the precision e1. The executing time of this loop is denoted by O3(n). In O3(n), the constants c5, c6 are similar to c3, c4 in O2(n).

$O3(n)=c5+(c6+c1+c2*n)*(k+1+log_2(1/e1))$

(4) THE OVERALL COMPUTING COMPLEXITY

Let the fixed overhead of this method be c7, and the overall executing time is:

$O(n)=c7+c3+(c4+c1+c2*n)*(k+1)+ c5+(c6+c1+c2*n)*(k+1+log_2(1/e1))$

All the constants, c1, c2, …, c7 are fixed and dependent on computing system, and parameters e1, k are either predefined or fixed to any given data set. Finally, we can simply say the computing complexity of our algorithm is

$O(n)=n$

The term "computing complexity $O(n)=n$" implies that the computing effort increases linearly as the problems size increasing in the worst case.

5. Experimental Analysis

(1) THE EXPERIMENTAL DATA

Thousands of testing data are generated randomly in this section, and controlled by n (problem size) and S (summation of input). In a random data set, the value of S is randomly determined within a range firstly, and then another process distributes this summation to each input (g_i) randomly again. For example, in the combination of n=2 and S=0.8 ±0.2, we get the value of S randomly from (0.6, 1) at first. And then, we distribute this value of S to g_1, g_2 randomly.

There is a little trick in randomly distributing a fixed S to $g_1 \ldots g_n$ and keeping them bounded in (0, 1). A random sequence between (0, 1), say $g'_1 \ldots g'_n$, is generated at first. Assume R is the summation of $g'_1 \ldots g'_n$. If R>S or S<1, then we can simply multiply each g'_i by S/R. However, if R<S and S>1, such a process can't guarantee each g'_i *S/R is still bounded within (0, 1). Therefore, we change the view from g'_i, S, R to 1- g'_i, n-S, n-R, to make sure they do not excess their boundaries.

Table 1 shows the results of each combination n and S under the precisions d= 0.0001, $\varepsilon=10^{-12}$. The sequence of n is 2, 3, 4, 5, 6, 7, 8, then, 16, 32, at last, 64. The values of S are discussed in two ways. (1) If S<1, then S is distributed uniformly in 4 absolutely range: 0.8±0.2, 0.4±0.1, 0.2±0.05, and 0.1±0.025. (2) If S>1, then S is distributed uniformly in 4 relative range: 1+(n-1)* (0.8 ±0.2), 1+(n-1)* (0.4 ±0.1), 1+(n-1)* (0.2 ±0.05), and 1+(n-1)* (0.1 ±0.025).

Table 1. The λ^a of fixed data set, and averages of λ^b, H#[c], |H|[d] under various combinations of S and n in 30 random tests

	S=1+(n-1)* (0.8±0.2)	S=1+(n-1)* (0.4±0.1)	S=1+(n-1)* (0.2±0.05)	S=1+(n-1)* (0.1±0.025)	S=0.8±0.2	S=0.4±0.1	S=0.2±0.05	S=0.1±0.025
n=2	-0.98761[a]	-0.81635	-0.55560	-0.33063	1.24994	14.99994	79.99994	359.99994
	-0.98935[b]	-0.84819	-0.61648	-0.36284	3.17614	19.60841	183.00738	610.37085
	14.00[c]	14.00	14.00	14.00	16.93	24.07	30.07	34.20
	0.000028[d]	0.000012	0.000007	0.000002	0.000005	0.000020	0.000026	0.000030
n=3	-0.99750	-0.90424	-0.68524	-0.44281	0.87018	8.23553	34.12372	117.24982
	-0.99668	-0.92254	-0.73785	-0.48930	0.94872	15.05157	52.14021	164.26495
	14.00	14.00	14.00	14.00	16.07	23.07	27.27	30.53

	0.000033	0.000019	0.000011	0.000006	0.000006	0.000019	0.000032	0.000046
n=4	-0.99945	-0.94733	-0.77142	-0.53204	0.75458	6.60797	25.28546	79.99994
	-0.99880	-0.95911	-0.80424	-0.55506	1.14321	8.33315	34.55428	107.87764
	14.00	14.00	14.00	14.00	16.33	22.00	26.00	29.33
	0.000036	0.000025	0.000012	0.000008	0.000007	0.000026	0.000044	0.000044
n=5	-0.99994	-0.97015	-0.83087	-0.60406	0.69867	5.88751	21.66864	65.88849
	-0.99987	-0.98143	-0.86475	-0.63175	0.90443	7.37860	25.32389	78.90986
	14.00	14.00	14.00	14.00	15.87	21.87	25.27	28.47
	0.000048	0.000029	0.000016	0.000008	0.000007	0.000028	0.000039	0.000057
n=6	-0.99994	-0.98273	-0.87299	-0.66266	0.66571	5.48236	19.71783	58.57587
	-0.99986	-0.98848	-0.89694	-0.69119	0.81978	5.98209	23.73429	70.96923
	14.00	14.00	14.00	14.00	15.73	20.93	25.20	28.00
	0.000048	0.000026	0.000018	0.000012	0.000006	0.000025	0.000049	0.000056
n=7	-0.99994	-0.98981	-0.90350	-0.71112	0.64410	5.22308	18.50104	54.12616
	-0.99992	-0.99510	-0.91825	-0.75079	0.99910	6.31972	21.81824	65.61039
	14.00	14.00	14.00	14.00	16.07	21.20	24.73	28.00
	0.000048	0.000032	0.000017	0.000014	0.000008	0.000031	0.000055	0.000062
n=8	-0.99994	-0.99396	-0.92596	-0.75128	0.62872	5.04303	17.67072	51.14056
	-0.99994	-0.99703	-0.93547	-0.78022	0.78508	5.80138	19.32992	62.62416
	14.00	14.00	14.00	14.00	15.73	21.07	24.47	27.73
	0.000056	0.000026	0.000019	0.000015	0.000006	0.000029	0.000052	0.000060
n=16	-0.99994	-0.99994	-0.98944	-0.91534	0.58014	4.49432	15.21649	42.56610
	-0.99994	-0.99982	-0.99014	-0.91994	0.87316	4.75031	16.53394	47.00269
	14.00	14.00	14.00	14.00	16.07	20.40	23.87	27.13
	0.000061	0.000041	0.000033	0.000021	0.000008	0.000026	0.000052	0.000070
n=32	-0.99994	-0.99994	-0.99969	-0.98676	0.55865	4.25970	14.20282	39.13776
	-0.99994	-0.99994	-0.99966	-0.98590	0.60504	4.50196	15.54286	42.42880
	14.00	14.00	14.00	14.00	15.40	20.13	23.80	26.73
	0.000061	0.000061	0.000029	0.000026	0.000007	0.000036	0.000040	0.000085
n=64	-0.99994	-0.99994	-0.99994	-0.99957	0.54840	4.15082	13.73944	37.59357
	-0.99994	-0.99994	-0.99994	-0.99946	0.50037	4.55727	14.39713	38.74102
	14.00	14.00	14.00	14.00	15.20	20.20	23.67	26.33
	0.000061	0.000061	0.000060	0.000028	0.000006	0.000031	0.000047	0.000089

Remark: d=0.0001, $\varepsilon=10^{-12}$

Each combination is tested 30 times randomly, and the average of λ, |H|, and H# are reported. Beside the random data, the value of λ of a fixed data set is given. The purpose of fixed data is verification. Each input of fixed data has the same value; and the summation of fixed data is equal to the middle point of random range.

(2) THE ANALYSIS OF RELIABILITY

This section discusses the reliability in two ways. The first is to compare the λ of known data set with the computing result outside the method. The other is to verify the results satisfying the identification equation H(λ)=0 or not. The required precision of λ is d=0.0001 in Table 1.

There are eight combinations of n=2 in the first row of Table 1, and four numbers in each combination; the first of the four numbers is the λ of fixed data set. Usually, this algorithm is too complicated to n=2; but it is easy to verify the results manually. For example, the contents of the fixed data are S=1.8, and g_1= g_2=0.9 in the combination of n=2 and S=1+(n-1)*(0.8$\pm$0.2), and S=0.1, and g_1= g_2=0.05 in n=2 and S=0.1$\pm$0.025. We can compare the values in the Table 1 with the "true" values below; and both the results are bounded in -0.98761$\pm$0.0001, and 359.99994$\pm$0.0001 (d=0.0001).

$$0.9+0.9+\lambda*0.9*0.9=1$$

$$\lambda=-0.8/0.81=-0.98765$$

$$0.05+0.05+\lambda*0.05*0.05=1$$

$$\lambda=0.9/0.0025=360$$

In another way, we can compute the values of function H; and both errors, $H_{g1=g2=0.9}$(-0.98761)=-0.000035, $H_{g1=g2=0.05}$(359.99994)=0.000054, are relatively small. The last number of each combination is the average of |H|. All the values in the Table 1 are very small, from 0.000002 to 0.000089; that is, the values of λ are quite reliable.

6. Conclusions

A bisection algorithm given in this paper has been used successfully to identify λ-fuzzy measure. The amount of required data is small, and as same as the problem size, n. The executing time is short in practice, and increases with n. An analysis of computing complexity O(n) is given, and the reliability is shown by thousands of samples. The robustness is discussed in two way, the data exceeding the ability of computing systems are screened in the validating step, and the rests are demonstrated reliable through four types of specially designed data. The implementation of this method is easy and effective.

7. References

[1] Grabisch, M. (1995), "The application of fuzzy integrals in multicriteria decision making," *European Journal of Operational Research* 89, 445-456.

[2] Lee, K.-M., and H. Leekwang, (1995), "Identification of λ-fuzzy measure by genetic algorithms," *Fuzzy Sets and Systems* 75, 301-309.

[3] Leszczyński, K., Penczek, P., and Grochulski, W. (1985), "Sugeno's fuzzy measure and fuzzy clustering," *Fuzzy Sets and Systems* 15, 147-158.

[4] Sugeno, M., and T. Terano, (1977), "A model of learning based on fuzzy information," *Kybernetes* 6, 157-166.

[5] Wang, Jia and Z. Wang (1997), "Using neural networks to determine Sugeno measures by statistics," *Neural Networks* 10:1, 183-195.

[6] Wang, Z., and G.J. Klir, (1992), *Fuzzy Measure Theory*. Plenum Press, New York.

[7] Wierzchoń, S.T. (1983), "An algorithm for identification of fuzzy measure," *Fuzzy Sets and Systems* 9, 69-78.

[8] Yuan, B. and G.J. Klir (1996), "Constructing fuzzy measures: a new method and its application to cluster analysis," IEEE, 567-571.