Information Sciences 181 (2011) 4230–4252

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Particle swarm optimization for determining fuzzy measures from data

Xi-Zhao Wang [a], Yu-Lin He [a], Ling-Cai Dong [a], Huan-Yu Zhao [b,*]

[a] Key Lab of Machine Learning and Computational Intelligence, Department of Mathematics and Computer Science, Hebei University, Baoding 071002, Hebei, PR China

[b] Institute of Applied Mathematics, Hebei Academy of Sciences, Shijiazhuang 050000, Hebei, PR China

ARTICLE INFO

Article history:
Received 25 December 2009
Received in revised form 21 May 2011
Accepted 1 June 2011
Available online 6 June 2011

Keywords:
Learning
Fuzzy measures
Fuzzy integrals
Particle swarm optimization

ABSTRACT

Fuzzy measures and fuzzy integrals have been successfully used in many real applications. How to determine fuzzy measures is the most difficult problem in these applications. Though there have existed some methodologies for solving this problem, such as genetic algorithms, gradient descent algorithms and neural networks, it is hard to say which one is more appropriate and more feasible. Each method has its advantages and limitations. Therefore it is necessary to develop new methods or techniques to learn distinct fuzzy measures. In this paper, we make the first attempt to design a special particle swarm algorithm to determine a type of general fuzzy measures from data, and demonstrate that the algorithm is effective and efficient. Furthermore we extend this algorithm to identify and revise other types of fuzzy measures. To test our algorithms, we compare them with the basic particle swarm algorithms, gradient descent algorithms and genetic algorithms in literatures. In addition, for verifying whether our algorithms are robust in noisy-situations, a number of numerical experiments are conducted. Theoretical analysis and experimental results show that, for determining fuzzy measures, the particle swarm optimization is feasible and has a better performance than the existing genetic algorithms and gradient descent algorithms.

1. Introduction

Fuzzy measures [31,33,36,68] and fuzzy integrals [15,59,64,66,70] have been applied successfully in multi-attributes decision-making [18,53,54], classification [52,65,69], information fusion [3,5,11,43,49], nonlinear multi-regression [30], feature selection [19,44] and image processing [24,32,34,35]. The reason of success is from the highly non-additive and non-linear characteristics of fuzzy measures and fuzzy integrals. Fuzzy measure is the generalization of classical measure by using non-additivity instead of additivity, which makes fuzzy measure be able to describe the importance of each individual information source (attribute or classifier) as well as the interaction [13] among them. Mathematically, fuzzy integral is a generalization of classical Lebesgue integral. Further more, it is also the generalization of OWA [67] and WOWA [45]. Due to the non-linearity, fuzzy integral is testified to be a better tool of information fusion.

When fuzzy integrals are applied to solve real problems, values of the corresponding fuzzy measures should be known in advance. It means that the identification of fuzzy measures is a fundamental task. Given a finite space with cardinality n, then generally, $2^n - 2$ coefficients are needed to determine for a fuzzy measure, which has the exponentially computational complexity. It explicitly shows that the task of determining fuzzy measures is very difficult.

* Corresponding author.
E-mail addresses: xizhaowang@ieee.org (X.-Z. Wang), zhaohuanyu@163.com (H.-Y. Zhao).

0020-0255/$ - see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2011.06.002

Many methods of determining fuzzy measures from data can be found from references. For example, Grabisch presented linear programs and quadratic programs [17] to identify fuzzy measures and successfully applied them to classification. These algorithms have a sparse constraint matrix and much long running-time. Furthermore, Grabisch [16] introduced a gradient descent algorithm (GD) which considerably decreases the running-time and required memory. GD was later testified to be suitable for Choquet integral and Sugeno integral [21,45] and more other types of non-linear integrals. Wang proposed the use of neural networks [46,47] to determine fuzzy measures, which can be regarded as a variation of classical gradient descent algorithm [1]. Keller proposed a reward-punishment approach [23] which is similar to the neural network methods. Wang also proposed using the genetic algorithm (GA) to determine fuzzy measures [14,40,51,56–58,63]. Due to the better robustness, GAs are further used by many researchers [9,10,12,27,28,73]. References show that the design of GAs depends on the type of training data. For example, the algorithm in reference [10] was designed based on a type of data with partial information. Here we will classify the different training data as Type-I Data [27,28,61,62], Type-II Data [12,51,56–58,63] and Type-III Data [8–10] (only some notations and no practical meaning). Type-I and Type-II Data easily obtains, but Type-I Data is difficult to be given by the experts due to the $2^n - 2$ coefficients, and we must collect large Type-II Data to give sufficient information to determine fuzzy measures. According to the concept of "basic information sets" [10], the authors present the Type-III Data and emphasize the importance of data collection. From Type-III Data, we can use much less data to identify fuzzy measure without losing any information. However, the procedure of collection is hard to be understood and needs much time. So in this paper, we do not discuss this type. Recently, Mendez-Vazquez [34] presented a new method based on "Least Absolute Shrink-age and Selection Operator (LASSO)" and "Expectation-Maximization (EA)". This method can keep the monotonicity constraints easily, and does not need huge storage. But its performance is influenced by "dissimilarity measure". Beliakov converted "fitting fuzzy measure" to a linear program [2] and designed a package called fmtools to facilitate other researchers. The efficiency of the algorithm was not investigated sufficiently in that work. More recently, Alavi presented a modified gradient descent algorithm (MGD) [1] and concluded that MGD had better performance than GD [16] and GA [58] through numerical simulations. In this paper, we will concentrate only on using particle swarm optimization technique to determine fuzzy measures.

Although using soft computing techniques such as GAs and neural networks to determine fuzzy measures is successful to some extent, there exist many limitations in the application process. For example, the GD and neural network frequently fall into the local minimum, and GAs are much slower. It is necessary to mine new computational techniques for determining fuzzy measures. In reference [72], the authors have tried to use a special particle swarm algorithm to determine general set functions from the given data and find that particle swarm optimization (PSO) is suitable. However, whether PSO can be used to determine types of fuzzy measures and to revise fuzzy measures has yet to be verified. In this paper, we mainly focus on this problem. Using PSO to determine various types of fuzzy measures is addressed in this paper. The related theoretical analysis and experimental demonstrations are given. In addition, a comparison among the MGD and GA and our proposed PSO methods is conducted.

The rest of the paper is organized as follows: In Section 2, we briefly introduce the basic concepts on fuzzy measures and fuzzy integrals respectively. In Section 3, we formulate the problems to be solved. In Section 4, we design some special particle swarm algorithms to determine two special fuzzy measures and to revise fuzzy measures; to test our algorithm. In Section 5, we compare our algorithms with the basic particle swarm algorithm (BPSO), MGD and GA in literatures, and conduct a number of experiments to verify whether our algorithms are robust. In Section 6, we draw some conclusions and propose our future research.

In this paper, all algorithms are implemented in visual basic 6.0. The CPU of computer is AMD Sempron 2400+.

2. Basic concepts

In this section, we recall some basic concepts and theorems required in this paper.

2.1. Fuzzy measure

Due to the limitation of classical measure, Sugeno, the Japanese scholar, presents set functions called fuzzy measures which use monotonicity instead of additivity. In practical applications, we often use regular fuzzy measure [55] on finite sets.

Definition 2.1. Let X be a finite set, 2^X be the power set of X. If set function $\mu : 2^X \to [0,1]$ satisfies the following conditions:

(1) $\mu(\phi) = 0$, $\mu(X) = 1$.
(2) If $E \in 2^X$, $G \in 2^X$, $E \subset G$, then $\mu(E) \leqslant \mu(G)$

then μ is called a regular fuzzy measure defined on 2^X.

From definition of the fuzzy measure, we can see that, if we want to determine a general regular fuzzy measure in a space of n elements, $2^n - 2$ coefficients are needed to determine. When n is bigger, the determination is more difficult. To decrease the complexity of general fuzzy measures, several special fuzzy measures [55] are proposed.

Definition 2.2. Let X be a finite set and 2^X be the power set of X. If a fuzzy measure $\mu : 2^X \to [0,1]$ satisfies the following conditions:

(1) $\mu(\phi) = 0$, $\mu(X) = 1$;
(2) $\mu(A \cup B) = \mu(A) + \mu(B) + \lambda \cdot \mu(A) \cdot \mu(B)$

$$\forall A, \quad B \subset X, \quad A \cap B = \phi, \quad \lambda \in (-1, \infty)$$

Then μ is called a regular λ-fuzzy measure defined on 2^X.

The condition (2) of Definition 2.2 is called as λ-rule. When $\lambda = 0$,λ-rule coincides with additivity of classical measure. Denoting finite set $X = \{x_1, x_2, \ldots x_n\}$, the value $\mu_i = \mu(\{x_i\})$ is called measure density.

Theorem 2.1 [55]. *The parameter λ of a regular λ-fuzzy measure is determined by the equation:*

$$\prod_{i=1}^{n}(1 + \lambda\mu_i) = 1 + \lambda \tag{1}$$

Theorem 2.2 [55]. *Let finite set $X = \{x_1, x_2, \ldots x_n\}$, $\lambda \neq 0$, then*

$$\mu(E) = \frac{1}{\lambda}\left[\prod_{x_i \in E}(1 + \lambda\mu_i) - 1\right], \quad \forall E \subset X$$

If we know the values of λ-fuzzy measures on singleton sets, we can use Theorem 2.1 to obtain the values of λ and then use Theorem 2.2 to obtain the values on other sets. It implies that a λ-fuzzy measure can be determined by measure densities. In addition, λ-fuzzy measure should satisfy the following bounding conditions [51]:

(1) *If there exists some $\mu_i = 1$, then $\mu_j = 0$ for any $j \neq i$.*
(2) *If $\mu_i < 1$ for all μ_i, then there exist at least two of them being positive.*

Now we explain that there is only one solution λ for the problem in Theorem 2.1. This result is important later in this paper.

Theorem 2.3. *If we know the measure density μ_i on finite set $X = \{x_1, x_2, \ldots x_n\}$, then there is only one solution λ obtained from (1).*

Proof. Let $\mu(X) = 1, \mu_i = \mu(\{x_i\}) = a_i,\ i = 1, 2, \ldots, n,\ f_k(\lambda) = \prod_{i=1}^{k}(1 + a_i \cdot \lambda),\ k = 2, \ldots, n$. Without loss of generality, suppose $a_1 > 0$, $a_2 > 0$, we can know $\forall \lambda \in (-1, \infty)$, $(1 + a_k \cdot \lambda) > 0$, $k = 1, \ldots, n$.

From $f_k(\lambda) = (1 + a_k \cdot \lambda) f_{k-1}(\lambda)$, we know

$$f_k'(\lambda) = a_k \cdot f_{k-1}(\lambda) + (1 + a_k \cdot \lambda) f_{k-1}'(\lambda) \quad \text{and } f_k''(\lambda) = 2a_k \cdot f_{k-1}'(\lambda) + (1 + a_k \cdot \lambda) f_{k-1}''(\lambda).$$

It can easily be proved that $\forall k = 2, \ldots, n$, $\forall \lambda \in (-1, \infty)$, if $f_{k-1}'(\lambda) > 0$ and $f_{k-1}''(\lambda) > 0$, then $f_k'(\lambda) > 0$ and $f_k''(\lambda) > 0$.

From $f_2'(\lambda) = a_1 \cdot (1 + a_2 \cdot \lambda) + a_2 \cdot (1 + a_1 \cdot \lambda) > 0$ and $f_2''(\lambda) = 2 \cdot a_1 \cdot a_2 > 0$, we obtain $f_k''(\lambda) > 0$. This explains that function $f_n(\lambda)$ is concave at $(-1, \infty)$. The derivation of $f_n(\lambda)$ is $f_n'(0) = \sum_{i=1}^{n} a_i$.

Note that $\lim_{\lambda \to \infty} f_n(\lambda) = \infty$. We conclude that

(1) If $\sum_{i=1}^{n} a_i < 1$, there is only one intersection of the curve $f_n(\lambda)$ and the line $f(\lambda) = 1 + \lambda$ in $(0, \infty)$.
(2) If $\sum_{i=1}^{n} a_i = 1$, the line $f(\lambda) = 1 + \lambda$ is the tangent line of $f_n(\lambda)$ at $\lambda = 0$.
(3) If $\sum_{i=1}^{n} a_i > 1$, due to $f_n'(\lambda) > 0$ and $f(\lambda) = 1 + \lambda \leqslant 0$, there is only one intersection of the curve $f_n(\lambda)$ and the line $f(\lambda) = 1 + \lambda$ in $(-1, 0)$.

We have thus proved the theorem. □

In evidence combination problems, there are two important fuzzy measures which are called belief measures and plausibility measures. They are all induced by basic probability assignments.

Definition 2.3 [55]. A set function $m : 2^X \to [0,1]$ is called a basic probability assignment, if it satisfies the following conditions:

(1) $\sum_{E \subset X} m(E) = 1$
(1) $m(\phi) = 0$

Definition 2.4 [55]. If m is a basic probability assignment, a set function $Bel : 2^X \to [0,1]$ defined by

$$Bel(E) = \sum_{F \subset E} m(F) \quad \forall E \in 2^X$$

is called a belief measure induced by m.

Definition 2.5 [55]. If m is a basic probability assignment, a set function $Pl : 2^X \to [0,1]$ defined by

$$Pl(E) = \sum_{F \cap E \neq \phi} m(F) \quad \forall E \in 2^X$$

is called a plausibility measure induced by m.

Any belief measure is super-additive, while any plausibility measure is sub-additive. Functions *Bel* and *Pl* defined by the same basic probability assignment m are dual in the sense that

$$Bel(E) = 1 - Pl(E^c), \quad \forall E \in 2^X \quad \text{and } Bel(E) \leqslant Pl(E), \quad \forall E \in 2^X.$$

From the definitions of belief measure and plausibility measure, we can see that if we want to determine them, we only need to determine the basic probability assignment which induces them. In addition, if we know belief measure or plausibility measure, then we can determine its corresponding plausibility measure or belief measure by duality.

2.2. Fuzzy integral

When a classical measure is generalized to fuzzy measure, classical integral with respect to classical measure should be generalized. The generalized integrals with respect to fuzzy measures are called fuzzy integrals. Several types of fuzzy integrals have been suggested in the literature [45,55]. Two of them are the Sugeno integral and the Choquet integral.

Definition 2.6. Let X be a finite set, 2^X be the power set of X, $f : X \to [0,1]$, μ be a regular fuzzy measure defined on 2^X. Then the Sugeno integral and the Choquet integral of function f with respect to μ are defined respectively by the following formulae:

$$(s) \int f d\mu = \vee_{i=1}^{n} (f(x_i) \wedge \mu(A_i)) \text{ and } (c) \int f d\mu = \sum_{i=1}^{n} (f(x_i) - f(x_{i-1})) \cdot \mu(A_i)$$

where $0 \leqslant f(x_1) \leqslant f(x_2) \leqslant \ldots \leqslant f(x_n) \leqslant 1$, $A_i = \{x_i, x_{i+1}, \ldots x_n\}$, $f(x_0) = 0$, $\mu(A_{n+1}) = 0$.

From Definition 2.6, we can see that Sugeno integral is not the generalization of classical Lebesgue integral, while Choquet integral is. In many real applications, users often like using Choquet integral. However, it does not mean that the Choquet integral is always superior to the Sugeno integral.

3. Questions description

In this section, we formulate our problems to be solved.

3.1. Determine fuzzy measures from Type-I Data

Supposing that $X = \{x_1, x_2, \ldots, x_n\}$ is a non-empty set, φ is a class constituted by subsets of X, F is a σ–algebra constituted by subsets of X, $\varphi \subset F$, in general $F = 2^X$. Considering a given set function $v : \varphi \to [0,1]$, the values of v may be provided by domain experts, which are regarded as the Type-I Data. We hope to find a fuzzy measure μ on F such that

$$\mu(E) = v(E) \quad \forall E \in \varphi$$

In reference [62], such a fuzzy measure μ is called an extension of v from φ to F. If $\varphi = F = 2^X$, then μ is called a fitting of v from φ to F. In this paper, this extension and fitting are called revising.

Such a fuzzy measure μ may not exist. If it exists (not only one), we should find one; if it does not exist, we should find an approximate one. Now we can use the least square method to transform the above problem to a constrained optimization problem. The optimization problem [62] is described as follows:

$$\min \ e^2 = \left(\sum_{E \in \varphi} [v(E) - \mu(E)]^2 \right) \Big/ l \tag{2}$$

Subject to: μ is one special type of fuzzy measures to be determined, where l represents the cardinality of φ. A result of $e^2 = 0$ means that a precise solution is found.

3.2. Determine fuzzy measure from type-II data

If we regard fuzzy integrals as multi-input single-output systems, we can obtain the Type-II Data through handling these systems. Suppose that we have several information sources $x_1, x_2, \ldots, x_n$, $n \geqslant 2$ and a given object y. Let $X = \{x_1, x_2, \ldots, x_n\}$, we have the following data with sample size m:

$$\begin{array}{ccccc} x_1 & x_2 & \cdots & x_n & y \\ f_{11} & f_{12} & \cdots & f_{1n} & E_1 \\ f_{21} & f_{22} & \cdots & f_{2n} & E_2 \\ \vdots & \vdots & & \vdots & \vdots \\ f_{m1} & f_{m2} & \cdots & f_{mn} & E_m \end{array}$$

where f_{ij} is the ith value of source x_j, E_i is the ith value of object.

We hope to find a fuzzy measure μ on measurable space $(X, 2^X)$, such that $E_i = \int f^i d\mu, \quad \forall i = 1, 2, \ldots, m$, where function f^i is defined by $f^i(x_j) = f_{ij}$, $j = 1, 2, \ldots, n$ for $i = 1, 2, \ldots, m$.

If such a fuzzy measure μ does not exist, we hope to find the optimally approximate solution. This is just the inverse problem of synthetic evaluation. We can also use the least square method to transform the above problem to a constrained optimization problem. An optimization problem [14] is described as follows:

$$\min_{\mu} e = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(E_i - \int f^i d\mu)^2} \tag{3}$$

Subject to: μ is one special type of fuzzy measures to be determined

A result of $e = 0$ also means that a precise solution is found.

4. Using particle swarm optimization to determine fuzzy measures

Inspired by the behavior of bird flocking, Kennedy and Eberhart introduced the particle swarm optimization (PSO) [25] in 1995. They designed a new algorithm to originally simulate simple social systems and then to explain the complex social behavior. In the further investigation, researchers find that the new algorithm can be used to solve complex optimization problems.

About particle swarm optimization, there are many issues under research. We classify two aspects: (A) Improving the original particle swarm algorithm by presenting some special strategies for special problems [4], introducing other methodologies into PSO [40], designing some hybrid algorithms based PSO and other soft computing technologies [50]; (B) Using particle swarm optimization [50] or swarm intelligence [42] to solve some actual problems which are difficult to be solved by some traditional methods.

Similar to GAs, PSO exploits a population of potential solutions (called particles) to probe the search space. But in PSO, each particle changes its positions by learning itself and its neighbors in each generation instead of using traditional genetic operators. It finally can find an optimally approximate solution.

There are two versions of PSO which are called global and local respectively. About the global version, there are many specific particle swarm algorithms for different optimization problems. In this paper, we consider the algorithm with inertia weight model [41] which is referred to as the basic particle swarm algorithm (BPSO).

Suppose that the function $f(x)$ which we want to optimize is defined on D dimensions. The steps of BPSO are described as follows.

Algorithm 1. 1—BPSO

Step 1: Randomly initialize the positions $x_i = \{x_{ij}\}$ and velocities $v_i = \{v_{ij}\}$ of swarm with the population size N on D dimensions. The variables are denoted by $\{x_1^*, x_2^*, \ldots, x_D^*\}$.

Step 2: Compute the fitness of each particle which is related to $f(x)$, and then update $p_i = \{p_{ij}\}$ and $g = \{g_j\}$ according to the fitness, where p_i is the location of the best solution which a particle has achieved so far, g is the location of the best solution that the whole swarm has achieved so far. If the fitness of g is larger than a given stopping criterion or the number of iteration is larger than a given threshold, then go to step 5; otherwise, go to step 3.

Step 3: Update the velocities according to the following formula:

$$v_{ij}(t+1) = w_i \cdot v_{ij}(t) + c_1 \cdot rand() \cdot (p_{ij}(t) - x_{ij}(t)) + c_2 \cdot rand() \cdot (g_j(t) - x_{ij}(t))$$

where w_i is called inertia weight which usually adopts a linear decreasing strategy as follows:

$$w_i(t) = \left\{\frac{(t_{\max} - t)}{(t_{\max})}\right\} \cdot (w_{initial} - w_{final}) + w_{final}$$

in which t represents tth iteration, $t_{\max}$ denotes a given maximum of iteration number, $w_{initial}$ is the inertia weight at the beginning, and w_{final} is the inertia weight at the end. *rand*() represents a random number in [0,1], c_1 and c_2 are called learning factors, each of which is set to be 2 in general. If the updated velocities $v_{ij} > v_{\max}$, then let $v_{ij} = v_{\max}$; if $v_{ij} < -v_{\max}$, then $v_{ij} = -v_{\max}$, where $v_{\max}$ is a constant controlled by users.

Step 4: Update particle positions according to the formula below:

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1)$$

where t represents tth iteration. If particle positions $x_{ij} > x_{j\max}$ and $x_{ij} < x_{j\min}$, then let $x_{ij} = x_{j\max}$ and $x_{ij} = x_{j\min}$, respectively, where $x_{j\max}$ is the upper bound of $x_j^*, x_{j\min}$ is the lower bound of x_j^*. Go to step 2.

Step 5: Output g and its corresponding fitness.

From the BPSO, we can see the main factors influencing on the performance are inertia weight, learning factor and maximum velocity. The function of inertia weight is to balance global exploration and local exploitation. The function of learning factors is to control the particle ability of learning itself and its neighbors respectively. The maximum velocity serves as a constraint to control the global exploration ability of particle swarm. In Ref. [39], the authors give the relationship between inertia weight and maximum velocity. The conclusion is that the inertia weight is the direct factor of influencing on the performance of BPSO; while the maximum velocity is an indirect factor. In general, we can set the maximum velocity and the value of dynamic range of each variable, and then tune the ability of global exploration and the local exploitation by changing the inertia weight. However, they point out that selecting the inertia parameter and setting the maximum velocity may be problem-dependent. So we should adopt different strategies [22,26,29] according to different problems.

Now we discuss using PSO to determine fuzzy measures from data.

4.1. GPSO

In this subsection, we briefly review some existing work on this topic. In [72], the authors designed a special particle swarm algorithm. In that algorithm, they solved four main questions. Firstly, considering the inertia weight model [39] and the complex performance mentioned in references [6,25,38,39,41,71], they adopted a nonlinear decreasing strategy of inertia weight [6]. Secondly, based on the experimental verification of methods [7,14,20] for dealing with the constraints, they adopted the reorder algorithm in the reference [60] to guarantee the monotonicity of general fuzzy measures. Thirdly, they adopted the sigmoid function to improve the basic algorithm. And finally, according to the special question of determining fuzzy measures, they designed a special velocity mutation to increase the diversity of the whole swarm, which can make the algorithm jump out of the local extremum to great extent. Through experimental analysis and comparisons, they further demonstrated that the proposed algorithm is effective and efficient. This algorithm was named as GPSO. It used PSO to determine general fuzzy measures, i.e., to determine a set function without any other constraints except for the monotonicity. The differences between GPSO and BPSO are that (A) BPSO adopts a linear decreasing strategy while our GPSO adopts a nonlinear decreasing strategy of inertia weight, and (B) the sigmoid function and velocity mutation are incorporated to the GPSO.

Now we give explanations about some technical concepts in reference [10].

(1) The sigmoid function
The formulation of sigmoid function is

$$y = \frac{1}{1+e^{(-x)}}$$

After we update particle positions, some position values may be larger than 1 or smaller than 0. When this case occurs, the general method in BPSO is making the position values be 1 or 0, respectively. Unfortunately, this adjustment badly influences the search of problem space. So in GPSO, the authors adopt this function to transform the updated particle position values into [0,1] when the position values are larger than 1 or smaller than 0. This can enhance the ability of searching problem space.

(2) The velocity mutation
The reason of the problem that easily falls into local extremum in BPSO is that every particle velocity value tends to zero with the number of iteration increasing. So the authors design a special strategy (the velocity mutation) in GPSO. In details, randomly selecting one particle in every iteration, and then randomly selecting one velocity in this particle, and then reinitializing the velocity.

(3) The successful rate
To verify the performance of GPSO, the authors set some values (empirical values) as the thresholds. The successful rate is that how many times can achieve with running the program 100 times.

(4) The process of updating p and g
The values g and p are very important for finding the optimal value in the PSO algorithm. The procedure of BPSO can be described as following: Firstly, g and p which represent the fuzzy measure actually are all initialized to 0; Then, we

find the best fuzzy measure of fitness in the current swarm in accordance with the fitness after each iteration and let p equal to the best fuzzy measure; Finally, we compare the fitness of p with g's: if the fitness of p is larger than g's, let $g = p$. This situation indicates that there is a particle whose fitness is better than last iteration in the current swarm; otherwise, the value of g remains unchanged. All works depicted above have been appended in the modified paper.

4.2. λ-PSO

Noting the specificity of λ-fuzzy measures, our main ideas for determining them are listed below:

(1) As mentioned in Section 2, if we want to determine λ-fuzzy measures, we only need to determine the measure densities. When we know the measure densities, we can solve the equation (1) to obtain λ and then get the fuzzy measure values on other sets by λ-rule. So we only initialize each particle that represents a measure density. In addition, regarding the method of solving the Eq. (1), we adopt the Newton method [23], bisection algorithm [48] or GA [28].
(2) Because λ-fuzzy measures satisfy the constraints of monotonicity, we do not need to adopt the reorder algorithm to revise the particles.
(3) In the reference [51], the authors added a step of unbiasing transformation to increase the speed of their GA and designed a step to verify the bounding conditions of λ-fuzzy measures. In our algorithm, we do not need these two steps. The reasons are as follows: Firstly, the purpose of unbiasing transformation in [51] is to transform the expectation from 1/2 to $1/n$ and to give a variety to the genes contained in chromosomes in the initial population (but PSO does not have the concept of gene); Secondly, the purpose of verifying the bounding condition in [51] is to make the gene value still locate in [0,1] after crossing and mutation, while our algorithm adopts the sigmoid function to make the particle locate in [0,1]. Therefore, in the process of our experiments, the performance of our algorithm without these two steps is not weakened.
(4) In reference [47], the algorithm running may lead to negative values of measure densities, so the authors used the bounding conditions to verify in their algorithm. But in our algorithm, because we adopt sigmoid function to keep the particle positions in [0,1], we need not to check up the bounding conditions.

Based on GPSO and the above four considerations, we now present the special particle swarm algorithm to determine λ-fuzzy measures. We denote this algorithm by λ-PSO.

Algorithm 2. λ-PSO

Step 1: Initialize a group of random particles, including two parts. One part is to initialize the particle positions. Each particle position represents a measure density, denoted by $\mu_i = \{\mu_{ij}\}$. The other part is to initialize particle velocities. Each particle velocity represents the changed value of particle position in each iteration, denoted by $v_i = \{v_{ij}\}$, where $\mu_{ij} \in [0,1]$, $v_{ij} \in [-1,1]$, $i = 1, 2, \ldots, m$, $j = 1, 2, \ldots, n$, m represents the number of particles, n represents the number of attributes. Then we initialize $pbest = 0$ and $gbest = 0$, where $pbest$ is the location of the best solution which a particle has achieved so far, $gbest$ is the location of the best solution that the whole swarm has achieved so far.

Step 2: Use Newton's method to obtain the value of λ and get the values of λ-fuzzy measures on other sets by λ-rule. Compute the fitness of each particle, which is defined by $\frac{1}{1+e}$, where e is computed by the formula (3), then update $pbest$ and $gbest$. If the fitness of $gbest$ is larger than a given stopping criterion or the number of iterations is larger than a given threshold, then go to step 6; otherwise, go to step 3.

Step 3: Update the particle velocities according to the following formula:

$$v_{ij}(t+1) = w_i \cdot v_{ij}(t) + c_1 \cdot rand() \cdot (pbest_{ij}(t) - \mu_{ij}(t)) + c_2 \cdot rand() \cdot (gbest_j(t) - \mu_{ij}(t)) \tag{4}$$

w_i is defined by

$$w_i(t) = \left\{ \frac{(t_{\max} - t)^l}{(t_{\max})^l} \right\} \cdot (w_{initial} - w_{final}) + w_{final} \tag{5}$$

where t represents tth iteration, $rand()$ represents a random number in [0,1], $c_1 = c_2 = 2$, $t_{\max}$ denotes a given maximum of iterations, $w_{initial}$ is the inertia weight at the beginning, w_{final} is the inertia weight at the end, and l represents a given index number. If the updated velocities $v_{ij} > v_{\max}$, then let $v_{ij} = v_{\max}$; if $v_{ij} < -v_{\max}$, then $v_{ij} = -v_{\max}$, where $v_{\max} = 1$.

Step 4: Reinitialize v_{kp} such that $v_{kp} \in [-1,1]$ where v_{kp} represents the pth velocity of the kth particle randomly selected from the current swarm.

Step 5: Update particle positions according to the following Eq. (6):

$$\mu_{ij}(t+1) = \mu_{ij}(t) + v_{ij}(t+1) \tag{6}$$

where t represents the tth iteration. If particle positions $\mu_{ij} > 1$ or $\mu_{ij} < 0$, then $\mu_{ij} = \frac{1}{1+e^{(-\mu_{ij})}}$. Go to step 2.

Step 6: Output $gbest$ and its corresponding error e.

The λ-PSO has been implemented via VB and has been executed on a number of datasets successfully. Two of them are shown below.

Example 1 [51]. The dataset Table 1 is selected from [51]. Choquet integral is adopted. After running the program for about 2 s, the results are shown in Table 2 and the convergence rate is shown in Fig. 1.

Example 2 [51]. The dataset Table 3 is selected from [51]. Sugeno integral is adopted. After running the program for about 3 s, the results are shown in Table 4 and the convergence rate is shown in Fig. 2.

4.3. BPPSO

According to the specificity of belief measures, the following considerations are proposed to design our algorithm:

(1) As mentioned in Section 2, if we want to determine belief measures and plausibility measures, we only need to determine the basic probability assignments. When we know the basic probability assignment, we can use Definitions 2.4 and 2.5 to determine the induced belief measure and plausibility measure. It implies that we only initialize each particle which represents a basic probability assignment. In addition, if we know the belief measures, we can use the duality to obtain its corresponding plausibility measures. The reverse case is the same.
(2) Because belief measures and plausibility measures satisfy the constraints of monotonicity, we still do not need to adopt the reorder algorithm to revise the particles.
(3) According to Definition 2.3, we must make the particles satisfy the conditions of basic probability assignments. In our algorithm, we use the formulae $Sum_i = \sum_j \mu_{ij}$ and $\mu_{ij} = \frac{\mu_{ij}}{Sum_i}$ $i = 1, 2, \ldots,$ $m, j = 1, 2, 3, \ldots, 2^n - 1$, where m represents the number of particles, n represents the number of attributes.
(4) We add a special step in the algorithm to decrease computational time. The velocities of particles are initialized by the following formula: $v_{ij} \in \left[\frac{-1}{(2^n-1)}, \frac{1}{(2^n-1)}\right]$, $i = 1, 2, \ldots,$ $m, j = 1, 2, \ldots, 2^n - 1$.

In addition, let $v_{\max} = 0.02$ in this algorithm.

The reason to add this step is because the condition (1) of Definition 2.3 makes the value of basic probability assignment much smaller. If the particle velocities are much larger, the particles probably fly over the optimal position in the updating process. Here we avoid this by decreasing the velocity and $v_{\max} = 0.02$.

Based on GPSO and the above four considerations, we present the specific particle swarm algorithm to determine belief measures. We denote this algorithm by BPPSO.

Algorithm 3. BPPSO

Step 1: Initialize a group of random particles, including two parts. One part is to initialize particle positions. Each particle position represents a basic probability assignment, denoted by $m_i = \{m_{ij}\}$. The other part is to initialize particle velocities. Each particle velocity represents the changed value of particle position in each iteration, denoted by $v_i = \{v_{ij}\}$, where $m_{ij} \in [0, 1]$, $v_{ij} \in \left[\frac{-1}{(2^n-1)}, \frac{1}{(2^n-1)}\right]$, $i = 1, 2, \ldots,$ $m, j = 1, 2, \ldots, 2^n - 1$, m represents the number of particles, n represents the number of attributes. Then we initialize *pbest* = 0 and *gbest* = 0, where *pbest* is the location of the best solution which a particle has achieved so far, *gbest* is the location of the best solution that the whole swarm has achieved so far.

Table 1
Data of example 1.

i	f_{i1}	f_{i2}	f_{i3}	f_{i4}	E_i
1	0	0	0.3	1	0.2
2	0.3	0.8	0.5	0.6	0.5
3	0.7	0.9	1	0.7	0.8
4	0.1	0	0.4	0.3	0.15
5	0.5	0.4	0.5	0.6	0.5
6	1	0.3	0.8	0.8	0.7
7	0.3	1	0	1	0.5
8	0.9	0.6	0.5	0.7	0.7
9	0.6	0.8	0.2	0.4	0.5
10	0.4	1	0.5	0	0.4
11	0.8	0.4	1	0.3	0.6
12	1	0.8	0.7	0.5	0.75
13	0.2	0.6	1	0.7	0.5
14	0	0.3	0.2	0.9	0.2

Table 2
Results of example 1.

Set	Value	λ	e	Number of generation
$\{x_1\}$	0.29750	0.95702	0.0160273	18
$\{x_2\}$	0.18364			
$\{x_3\}$	0.13521			
$\{x_4\}$	0.15378			

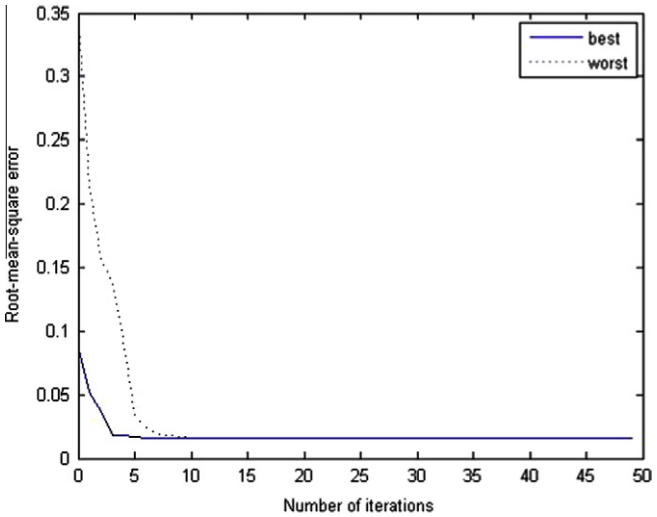

Fig. 1. The convergence rate of example 1.

Table 3
Data of example 2.

i	f_{i1}	f_{i2}	f_{i3}	f_{i4}	f_{i5}	E_i
1	0.8	0.3	0.1	0.1	0	0.131623
2	0.2	0.6	1	0.7	0.4	0.506664
3	0.3	0.4	0.2	1	0.1	0.354552
4	0.7	0.3	0.4	0.1	0.9	0.349929
5	1	0.5	0.9	0.9	1	0.893325
6	0.4	0	0.2	0.7	0.9	0.48364
7	0.9	0.8	0.9	1	0.3	0.70570
8	0.5	1	0.4	0.7	0.6	0.557034
9	0.7	0.9	0.8	0.2	0.7	0.412461

Table 4
Results of example 2.

Set	Value	λ	e	Number of generation
$\{x_1\}$	0.09004	2.38318	0.000089	31
$\{x_2\}$	0.03412			
$\{x_3\}$	0.08254			
$\{x_4\}$	0.29626			
$\{x_5\}$	0.10983			

Step 2: Use consideration (3) earlier proposed in this subsection to handle the particles such that the condition of basic probability assignments is satisfied. Then use Definition 2.4 to obtain the value of belief measure induced by m_i. Denote the belief measure by $Bel_i = \{bel_{ij}\}$. Compute the fitness of each particle, which is defined by $\frac{1}{1+e}$, where e is computed by the formula (3), then update *pbest* and *gbest*. If the fitness of *gbest* is larger than a given stopping criterion or the number of iteration is larger than a given threshold, then go to step 6; otherwise, go to step 3.

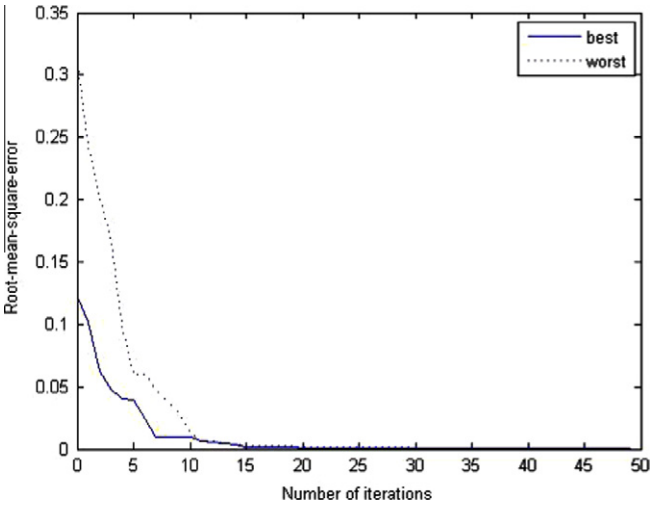

Fig. 2. The convergence rate of example 2.

Step 3: Update the particle velocities according to the following formula:

$$v_{ij}(t+1) = w_i \cdot v_{ij}(t) + c_1 \cdot rand() \cdot (pbest_{ij}(t) - m_{ij}(t)) + c_2 \cdot rand() \cdot (gbest_j(t) - m_{ij}(t))$$

where w_i is defined by $w_i(t) = \left\{\frac{(t_{\max}-t)^l}{(t_{\max})^l}\right\} \cdot (w_{initial} - w_{final}) + w_{final}$ and the settings of parameters are the same as λ-PSO. Only a difference is $v_{\max} = 0.02$.

Step 4: Reinitialize v_{kp} such that $v_{kp} \in \left[\frac{-1}{(2^n-1)}, \frac{1}{(2^n-1)}\right]$, where v_{kp} has the same meaning as the step (4) of Algorithm 2.

Step 5: Update particle positions according the formula below: $m_{ij}(t+1) = m_{ij}(t) + v_{ij}(t+1)$, where t represents tth iteration, and m_{ij} is the value of the particle position. If $m_{ij} > 1$ or $m_{ij} < 0$, then $m_{ij} = \frac{1}{1+e^{(-m_{ij})}}$. Go to step 2.

Step 6: Output *gbest* and its corresponding error e.

Example 3 [57]. The data is shown in Table 5 and the Choquet integral is adopted. After running the program about 2s, the results are shown in Table 6 and the convergence rate is shown in Fig. 3.

4.4. FEPSO

From sub Section 3.1, we know that, for solving the optimization problems, we do not need to compute the values of nonlinear integrals. The algorithms are similar to GPSO, λ-PSO and BPPSO. The main differences between them are that e is computed by formula (2) instead of (3) and the fitness function uses $\frac{1}{1+\sqrt{e^2/l}}$ instead of $\frac{1}{1+e}$, where l represents the cardinality of φ.

We denote the algorithm by FEPSO. Brief steps of FEPSO are listed below.

Algorithm 4. FEPSO

Step 1: Initialize a group of random particles, including two parts. One part is to initialize particle positions. The representation of each particle is based on the fuzzy measure to be identified. The other part is to initialize particle velocities. During the process of iteration, each particle velocity represents the changing value of particle position. Then initialize *pbest* = 0 and *gbest* = 0 , where *pbest* is the location of the best solution which a particle has achieved so far, *gbest* is the location of the best solution that the whole swarm has achieved so far.

Table 5
Data of example 3.

i	f_{i1}	f_{i2}	f_{i3}	E_i
1	0.9	0.5	0.1	0.5
2	0.2	0.3	0.8	0.35
3	0.4	1	0.6	0.6
4	0	0.3	0.9	0.23
5	0.7	0.2	0.4	0.42
6	0.8	0.7	1	0.8

Table 6
Results of example 3.

SET	m	Bel	e	Number of generation
ϕ	0	0	0.000748	68
$\{x_1\}$	0.33881	0.33881		
$\{x_2\}$	0.24750	0.24750		
$\{x_3\}$	0.19761	0.19761		
$\{x_1,x_2\}$	0.07499	0.66129		
$\{x_1,x_3\}$	0.05834	0.59475		
$\{x_2,x_3\}$	0.06225	0.50736		
X	0.02051	1		

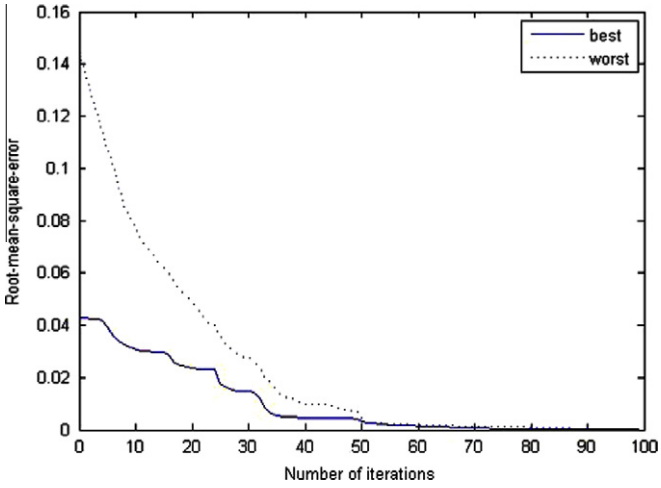

Fig. 3. The convergence rate of example 3.

Step 2: Compute the fitness of each particle, which is defined by $\frac{1}{1+\sqrt{e^2/l}}$, where e is computed by the formula (2), then update *pbest* and *gbest*. If the fitness of *gbest* is larger than a given stopping criterion or the number of iteration is larger than a given threshold, then go to step 6; otherwise, go to step 3.
Step 3: Update the particle velocities according to the formula (4).
Step 4: Conduct the mutation of velocity (Similar to step 4 in sub Sections 4.2 and 4.3).
Step 5: Update particle positions according to the formula (6).
Step 6: Output *gbest* and its corresponding error e.

We give two examples to illustrate FEPSO.

Example 4 [62]. Suppose that domain experts have specified a set function defined on the power set of X Table 7. It is easy to check the set function does not satisfy the properties of λ-fuzzy measures. The objective of this example is to revise the values of the set function such that it becomes a λ-fuzzy measure. After running the program for about 1s, a regular λ-fuzzy measure with $\lambda = -0.760844$ is obtained. The error $e = 0.019588$, while the number of generations is 13. The results are also shown in Table 7 and the convergence rate is shown in Fig. 4.

Table 7
Data and results of example 4.

Set	Value of v	Value of μ
ϕ	0	0
$\{x_1\}$	0.6	0.60875
$\{x_2\}$	0.5	0.49350
$\{x_3\}$	0.4	0.37678
$\{x_1,x_2\}$	0.9	0.87368
$\{x_1,x_3\}$	0.8	0.81102
$\{x_2,x_3\}$	0.7	0.72880
X	1	1

Example 5 [62]. Domain experts first give a number of data Table 8. Each data is considered as a value which a λ-fuzzy measure take on a certain subset of X. That is, experts specify partial values of a λ-fuzzy measure. The objective is to extend the partially specified fuzzy measure to the power set. After running the program about 2s, a regular λ-fuzzy measure with $\lambda = 2.390078$ is found. The error $e = 0.000506$, while the number of generation is 26. The results are shown in Table 8 and the convergence rate is shown in Fig. 5.

From the above discussion, we see that the λ-PSO, BPPSO and FEPSO proposed in this paper are based on GPSO. They are designed to determine λ-fuzzy measure, belief or plausible measure and to revise fuzzy measure respectively. The differences between λ-PSO, BPPSO, FEPSO and GPSO are (A) GPSO is a general method and has no specific results, (B) some specific constraints are introduced in GPSO to form λ-PS, BPPSO and FEPSO for determining different types of fuzzy measures, (C) due to the specific constrains introduced, λ-PSO, BPPSO and FEPSO (including the algorithm design and algorithm performance) have a constructive change respectively comparing with GPSO, and (D) they are not a simple revision/extension of GPSO.

5. Test and analyze the performance of our algorithms

In this section, we compare our algorithms with BPSO, GAs, and MGD respectively. Moreover, we verify whether our algorithms are robust by adding perturbations (noises) to the Type-II Data.

5.1. Comparisons with BPSO

The main differences between our algorithm and BPSO are described as follows:

(1) Our algorithm adopts nonlinear decreasing strategy of inertia weight instead of linear decreasing strategy in BPSO. The sigmoid function and velocity mutation are used in our algorithm.
(2) The specific structural characteristics of different types of fuzzy measures are incorporated into our algorithm for identification.

To test the performance of our algorithms, we first compare our algorithm with BPSO with respect to time complexity, success rate and number of iterations. We use the examples in Section 4. Let $w_{initial} = 0.2$, $w_{final} = -0.3$, $l = 1.1$, $t_{\max} = 2000$ in our algorithms and $w_{initial} = 0.9$, $w_{final} = 0.4$, $t_{\max} = 2000$ in BPSO, the population size be 150 and the stopping errors of examples be 0.016029, 0.001408, 0.000872991, 0.019588, 0.000564 respectively. The five errors are the results in papers [16,21,22]. We can regard them as empirical errors. In addition, we give examples 10–12, 16–17 for enhancing the comparison. The initial data of examples 10–12 are listed in Tables 20–22. The dataset of example 16 is constructed by the strategy 1 that is introduced in subsection C. For space consideration, we do not give the dataset of example 16. The dataset of example 17 is listed in Table 26. The stopping errors of these examples are 0.000698, 0.001, 0.0001, 0.0001 and 0.000001 and the parameters settings are same as above. The averaged results of running the program 100 times are shown in Tables 9,10,11,12, 27,28.

From Tables 9,10,11,12,27,28, we see the performances of our algorithms are much better than BPSO about successful rate and number of iterations. One reason is that adopting the sigmoid function makes our algorithm sufficiently search the problem space during the learning process. Another reason is that adding velocity mutation increases the diversity of swarm. This enhances the ability of jumping out of local minimum. In addition, by analyzing our algorithm, we see our algorithms do not

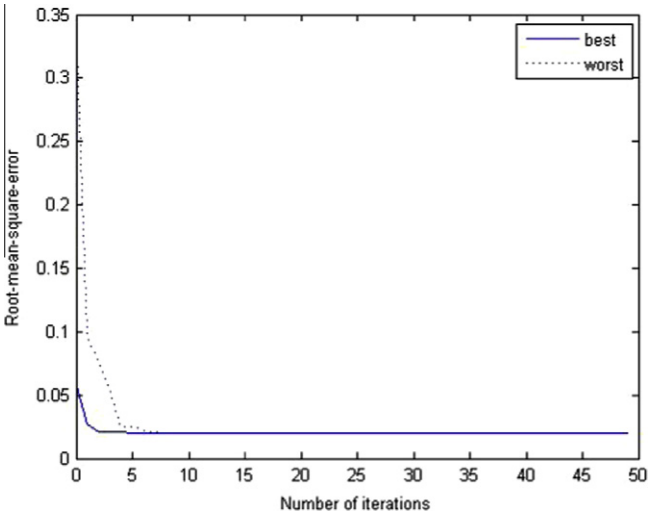

Fig. 4. The convergence rate of example 4.

Table 8
Data and results of example 5.

Set	Value of v	Value of μ
ϕ	0	0
$\{x_1\}$	0.090820	0.091203
$\{x_2\}$		0.034667
$\{x_3\}$		0.081278
$\{x_4\}$		0.296424
$\{x_5\}$	0.108398	0.108686
$\{x_1,x_2\}$		0.133427
$\{x_1,x_3\}$		0.190198
$\{x_1,x_4\}$		0.452242
$\{x_1,x_5\}$		0.223581
$\{x_2,x_3\}$	0.121899	0.122679
$\{x_2,x_4\}$		0.355652
$\{x_2,x_5\}$		0.152359
$\{x_3,x_4\}$	0.435820	0.435285
$\{x_3,x_5\}$		0.211077
$\{x_4,x_5\}$		0.482112
$\{x_1,x_2,x_3\}$		0.240624
$\{x_1,x_2,x_4\}$		0.524381
$\{x_1,x_2,x_5\}$		0.276773
$\{x_1,x_3,x_4\}$		0.621373
$\{x_1,x_3,x_5\}$		0.348292
$\{x_1,x_4,x_5\}$	0.678980	0.678407
$\{x_2,x_3,x_4\}$		0.506019
$\{x_2,x_3,x_5\}$		0.263234
$\{x_2,x_4,x_5\}$		0.556725
$\{x_3,x_4,x_5\}$		0.657044
$\{x_1,x_2,x_3,x_4\}$		0.707525
$\{x_1,x_2,x_3,x_5\}$		0.411817
$\{x_1,x_2,x_4,x_5\}$	0.768992	0.769285
$\{x_1,x_3,x_4,x_5\}$		0.891472
$\{x_2,x_3,x_4,x_5\}$		0.746152
$\{x_1,x_2,x_3,x_4,x_5\}$	1	1

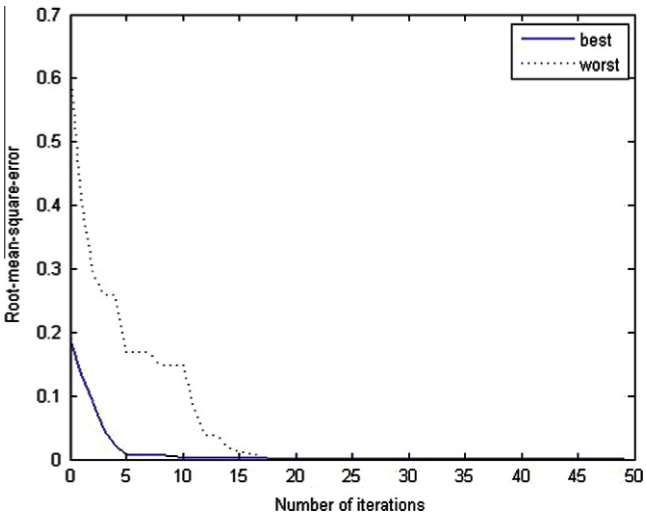

Fig. 5. The convergence rate of example 5.

increase time complexity compared with BPSO. What is more, the results of our experiments prove that the revisions based on the special fuzzy measures are very reasonable.

5.2. Comparisons with GAs

To compare our algorithms with the GAs [51,57,62], we test the same examples as in Section 5.1. The parameters settings are same as Section 5.1. For a relatively fair comparison, we use the same computational environment and the same stopping criterion to test GAs. The averaged results of running the program 100 times are shown in Table 13.

Table 9
Data of example 10.

i	f_{i1}	f_{i2}	f_{i3}	f_{i4}	f_{i5}	E_i
1	1	1	0.9	0.7	0.5	0.70229
2	0.3	0.1	1	0.9	0.6	0.61636
3	0.5	0.7	0.3	0.5	0.9	0.505893
4	1	0.5	0.4	0.1	0.5	0.296415
5	0.8	0.6	0.8	0.8	0.7	0.751446
6	0.4	0	0.2	0.7	0.9	0.481258
7	0.9	0.8	0.9	1	0.3	0.744601
8	0.5	1	1	0.5	0.1	0.443916
9	0.7	0.9	0.8	0.2	0.7	0.421968

Table 10
Data of example 11.

i	f_{i1}	f_{i2}	f_{i3}	E_i
1	.3824274	.2995678	7.930499E-02	.252605
2	.4745142	.2888865	.111349	.2917283
3	.3783823	.5772824	.2461612	.3825428
4	.239525	.9938543	.7308662	.5542063
5	.2542514	.6840957	.9257212	.5199865
6	.2008448	.8443157	.1794821	.3520296
7	.9084037	.9872181	.4851956	.7842782
8	.1945256	.1451641	2.766943E-02	.1219726
9	.3801534	.3227016	.7606362	.4319926
10	.1812532	.6538228	.7672724	.4433112

Table 11
Data of example 12.

Set	Bel
ϕ	0
$\{x_1\}$	0.3388
$\{x_2\}$	
$\{x_3\}$	
$\{x_1,x_2\}$	0.6613
$\{x_1,x_3\}$	
$\{x_2,x_3\}$	0.5074
X	

Table 12
The comparisons of basic algorithms and improved algorithms.

		Successful rate (%)	Number of iteration
Example 1	BPSO	100	388
	λ-PSO	100	14
Example 2	BPSO	63	274
	λ-PSO	100	33
Example 3	BPSO	73	81
	BPPSO	100	47
Example 4	BPSO	100	333
	FEPSO	100	11
Example 5	BPSO	88	401
	FEPSO	100	20
Example 10	BPSO	13	458
	λ-PSO	100	23
Example 11	BPSO	99	79
	BPPSO	100	46
Example 12	BPSO	100	63
	FEPSO	100	56

From Table 13, we see our algorithms have better performance than GAs. For the success rate, our algorithms are very stable; while GAs have good performance but do not reach 100%, because GAs have the possibility of precocity. On number of iterations, our algorithms are faster than GAs. It may result from the particle swarm algorithm which has a more adaptive

ability itself. Each particle has memory and the updating process is guided by *pbest* and *gbest*, which makes the particle swarm algorithm show a single direction searching mechanism. The factors that influence time complexity are the population size N, the number of attributes n, the number of cases m and the number of iterations t. The time complexity between our algorithms and GAs is mainly determined by number of attributes, and is exponentially increasing with n. From Table 13, we see this detail. Compared with other examples, the running time about examples 2 and 5 is a little longer. It means the time-complexity of running the program is fast increasing with the number of attributes. For problems with same number of attributes, the main factor influencing the time complexity is the population size. Because the difference between them is in the process of updating the population, we only need to compare the time complexity in updating the population. An analysis on the complexity of the population shows that the time complexity of our algorithm is $O(N)$ and the GAs are $O(N \cdot \log_2 N)$. What is more, the convergent speed of our algorithms is faster than GAs. The experimental results testify this result of time complexity analysis.

5.3. Comparisons with MGD

In this subsection, we compare GPSO with MGD [16] on different training datasets. We adopt two strategies to construct our Type-II Data. The procedures are described as follows:

Strategy 1:

(1) Given an original fuzzy measure shown in Table 14.
(2) Construct values of information sources by a generator which creates random numbers independently and uniformly distributed in $[0,1]$.
(3) Calculate the fuzzy integrals as output.

Strategy 2:

(1) Given an original fuzzy measure shown in Table 14.
(2) Construct values of information sources by random selection from set $\{0, 0.5, 1\}$.
(3) Calculate the fuzzy integrals as output.

Five experiments are given to illustrate the comparison. Three of them are examples 6–7, 13 following strategy 1 and the other is example 8, 14 following strategy 2. The size of training data is 20, 100, 200, 81 and 729, respectively. To save apace, we only list the dataset of example 6 in Table 15. In addition, to verify the stability of MGD, we test 100 datasets constructed by strategy 1 and regard it as example 15. The Choquet integral is used in this subsection.

Now we test our GPSO and MGD on examples 1, 6, 7, 8, 13, 14, 15. The parameters settings of GPSO are same as in subsection V.A. The stopping errors of examples are 0.00875734, 0.0001, 0.000001, 0.0000001, 0.000001, 0.0000001 and 0.0000001 respectively. The experimental results are shown in Tables 16,17,18,19, 23, 24, 25.

Regarding the processing time, we can conclude from Tables 16,17,18,19,23, 24 that MGD is better than our GPSO definitely. One obvious reason is that the time complexity of MGD is much lower than algorithms based on population including GA and GPSO. Another reason is, in most cases, the convergent speed of MGD is faster than GPSO. However, we see that the number of generations of GPSO is smaller than MGD from Table 16 and MGD falls into local minimum from Table 17. Compared with GPSO, the stability of MGD is not better. Table 25 demonstrates this point. This shows the performance of MGD is severely influenced by the training data. Easy-falling to the local minimum seriously affects the performance of MGD. Moreover, the difference of learning accuracy between MGD and GPSO is very little. Experimental results show that, in comparison with GPSO which uses a random search strategy, MGD finds the optimal fuzzy measure along the direction of gradient descent and keeps the mean square error minimal. This can be considered as one reason that the MGD gives better solutions in accuracy and training time. Another reason is that, under the condition MGD dose not fall into the local minimum, MGD has the smaller iteration-times and lower complexity.

Table 13
The comparisons of genetic algorithms and improved algorithms.

		Successful rate (%)	Number of iteration	Time (s)
Example 1	GA [18]	97	179	About 20
	λ-PSO	100	15	About 2
Example 2	GA [18]	96	258	About30
	λ-PSO	100	30	About 4
Example 3	GA [17]	99	122	About 10
	BPPSO	100	45	About1.5
Example 4	GA [13]	96	77	About 8
	FEPSO	100	13	About 1
Example 5	GA [13]	97	125	About 30
	FEPSO	100	17	About 3

Table 14
The original fuzzy measure of example 6–8.

	Original		Original
$\mu(\{x_1\})$	0.1	$\mu(\{x_2,x_3\})$	0.4211
$\mu(\{x_2\})$	0.2105	$\mu(\{x_2,x_4\})$	0.8070
$\mu(\{x_3\})$	0.2353	$\mu(\{x_3,x_4\})$	0.8235
$\mu(\{x_4\})$	0.6667	$\mu(\{x_1,x_2,x_3\})$	0.5
$\mu(\{x_1,x_2\})$	0.3	$\mu(\{x_1,x_2,x_4\})$	0.8667
$\mu(\{x_1,x_3\})$	0.3235	$\mu(\{x_1,x_3,x_4\})$	0.8824
$\mu(\{x_1,x_4\})$	0.7333	$\mu(\{x_2,x_3,x_4\})$	0.9474

Table 15
Data of example 6.

i	f_{i1}	f_{i2}	f_{i3}	f_{i4}	E_i
1	.4320337	.3491741	.1289113	.5241206	.4418743
2	.3384928	.1609553	.4279886	.6268888	.524032
3	.2957675	.2891313	4.346E-02	.7804725	.5846908
4	.3038577	.2025436	.4441691	.7192926	.5907549
5	.3627636	.69793	.4268516	.505666	.5278026
6	3.64E-03	.7752904	.7259288	.6084342	.6367602
7	.9609181	.9034663	.3414009	.762018	.7538807
8	.2345876	.3480371	.2732678	.4975758	.4324786
9	.5590727	.8434235	.2559503	.2839491	.4225496
10	.4655318	.6552047	.5550275	.1211381	.3515279
11	.2655494	.6261235	1.324E-02	.383044	.3769789
12	.2736396	7.069E-02	.6795302	.8530226	.6998073
13	.8637039	.8316599	.9277919	.639396	.7604112
14	.5045838	.6434411	.2268692	.4765849	.4813345
15	.4618585	.7716171	.8423412	.8957479	.8494977
16	.2043695	.6850294	.9774705	.834568	.8168723
17	.5288545	.1804157	.2257321	.6209414	.5032343
18	.1697344	.2577761	.5248094	.7237096	.605919
19	.1270091	.3859521	.1402815	.4707687	.3945077
20	.9941539	.4239982	.4000445	.9407473	.8052169

Table 16
The results of GPSO and MGD on example 1.

	MGD	GPSO
$\mu(\{x_1\})$	.307266	.2658922
$\mu(\{x_2\})$	.1825316	.1626612
$\mu(\{x_3\})$	.1082779	6.206E−02
$\mu(\{x_4\})$	9.865E-02	.094211
$\mu(\{x_1,x_2\})$	.5392901	.5531115
$\mu(\{x_1,x_3\})$	.524068	.5245267
$\mu(\{x_1,x_4\})$	.5351469	.6794262
$\mu(\{x_2,x_3\})$	.448338	.4685422
$\mu(\{x_2,x_4\})$	.3817421	.3868518
$\mu(\{x_3,x_4\})$	.4106263	.4270666
$\mu(\{x_1,x_2,x_3\})$	.6674463	.69666
$\mu(\{x_1,x_2,x_4\})$	.7761548	.7730081
$\mu(\{x_1,x_3,x_4\})$	.6828253	.6912512
$\mu(\{x_2,x_3,x_4\})$	.5635936	.596729
Processing time	0.3 s	2 s
e	8.75E−03	8.64E−03
Number of generation	45	13

5.4. Verification of the robustness of our algorithms

Practically Type-II Data may include some noise. In this subsection we test whether our algorithms are sensitive to the noise of Type-II Data, in other words, whether they are robust. It is expected that, for a robust algorithm, the error between actual outputs and expected outputs is not notably increased when data includes bigger noise.

For the purpose of verifying robustness of our algorithm, we adopt two procedures used in references [12,63] to add perturbations in given data. The procedures are the following:

Table 17
The results of GPSO and MGD on example 6.

	MGD	GPSO
$\mu(\{x_1\})$	.2250576	9.92E-02
$\mu(\{x_2\})$	.2230763	.210425
$\mu(\{x_3\})$	.2447411	.2350132
$\mu(\{x_4\})$	.671806	.6666449
$\mu(\{x_1,x_2\})$	.2860792	.3000486
$\mu(\{x_1,x_3\})$	.4803794	.3227769
$\mu(\{x_1,x_4\})$	.7239075	.7333602
$\mu(\{x_2,x_3\})$	.4497355	.4206176
$\mu(\{x_2,x_4\})$	.8026575	.8070102
$\mu(\{x_3,x_4\})$	.8225247	.8234757
$\mu(\{x_1,x_2,x_3\})$	.4845417	.5001692
$\mu(\{x_1,x_2,x_4\})$	.8575634	.8667863
$\mu(\{x_1,x_3,x_4\})$	.879716	.8825088
$\mu(\{x_2,x_3,x_4\})$	.9429958	.947484
Processing time	0.4 s	32 s
e	1.367406E−03	1.000852E−05

Table 18
The results of GPSO and MGD on example 7.

	MGD	GPSO
$\mu(\{x_1\})$	.1000003	9.9999E-02
$\mu(\{x_2\})$	.2105009	.2104997
$\mu(\{x_3\})$	.2353008	.2352984
$\mu(\{x_4\})$	.6667019	.6667014
$\mu(\{x_1,x_2\})$	.3000003	.3000033
$\mu(\{x_1,x_3\})$	.3235041	.323504
$\mu(\{x_1,x_4\})$	.7333014	.7332984
$\mu(\{x_2,x_3\})$	.4210998	.4210973
$\mu(\{x_2,x_4\})$	.8069983	.8070008
$\mu(\{x_3,x_4\})$	.8234893	.8235021
$\mu(\{x_1,x_2,x_3\})$	.4999995	.5000058
$\mu(\{x_1,x_2,x_4\})$	.8666996	.8667004
$\mu(\{x_1,x_3,x_4\})$	.8823969	.8823989
$\mu(\{x_2,x_3,x_4\})$	.9474015	.9474007
Processing time	0.8	1 min
e	9.319149E−07	9.833265E−07
Number of generation	47	76

Table 19
The results of GA, GD, GPSO and MGD on example 8.

	GA [20]	GD [10]	MGD	GPSO
$\mu(\{x_1\})$	0.1018	0.1114	0.0998	0.1000014
$\mu(\{x_2\})$	0.2118	0.2767	0.2114	0.2105008
$\mu(\{x_3\})$	0.2353	0.2743	0.2353	0.2352987
$\mu(\{x_4\})$	0.6652	0.6169	0.6674	0.6667012
$\mu(\{x_1,x_2\})$	0.3056	0.2731	0.3004	0.2999988
$\mu(\{x_1,x_3\})$	0.3283	0.3083	0.3237	0.3235011
$\mu(\{x_1,x_4\})$	0.7266	0.6910	0.7333	0.7332995
$\mu(\{x_2,x_3\})$	0.4220	0.2636	0.4209	0.4211024
$\mu(\{x_2,x_4\})$	0.7974	0.7444	0.8067	0.8069981
$\mu(\{x_3,x_4\})$	0.8133	0.6169	0.8232	0.8235011
$\mu(\{x_1,x_2,x_3\})$	0.4999	0.5752	0.5000	0.4999982
$\mu(\{x_1,x_2,x_4\})$	0.8663	0.8924	0.8665	0.8667006
$\mu(\{x_1,x_3,x_4\})$	0.8824	0.9169	0.8823	0.8823968
$\mu(\{x_2,x_3,x_4\})$	0.9540	1.0000	0.9475	0.9474013
Processing time	27 min	1.7 s	0.7 s	45 s
e	3.5e−3	2.6e−3	3.13e−8	9.847529E−07

(A) Add random perturbation with diverse strengths into (3) of strategy 1 as output. The random perturbation is generated by a random variable uniformly distributed in interval $[-0.5\rho, 0.5\rho]$, where ρ represents the diverse strengths.
(B) Add Gaussian noise with increasing variance σ^2 into (3) of strategy 2 as output.

Table 20
The original fuzzy measure and results of GPSO.

	Original	$\rho = 0$	$\rho = 0.001$	$\rho = 0.01$
T		70	65	67
e		0.000409	0.000434	0.002705
$\mu(\phi)$	0	0	0	0
$\mu(\{x_1\})$	0.2	0.199233	0.199067	0.198263
$\mu(\{x_2\})$	0.4	0.399636	0.39993	0.401359
$\mu(\{x_3\})$	0.1	0.098701	0.099429	0.097345
$\mu(\{x_4\})$	0.5	0.500164	0.499956	0.499412
$\mu(\{x_1,x_2\})$	0.5	0.500581	0.501740	0.500166
$\mu(\{x_1,x_3\})$	0.4	0.402037	0.401545	0.399139
$\mu(\{x_1,x_4\})$	0.6	0.599817	0.600127	0.599176
$\mu(\{x_2,x_3\})$	0.4	0.400466	0.400123	0.401389
$\mu(\{x_2,x_4\})$	0.7	0.699366	0.700209	0.703093
$\mu(\{x_3,x_4\})$	0.7	0.700860	0.700761	0.69979
$\mu(\{x_1,x_2,x_3\})$	0.6	0.599452	0.599671	0.600771
$\mu(\{x_1,x_2,x_4\})$	0.8	0.801328	0.799024	0.796924
$\mu(\{x_1,x_3,x_4\})$	0.9	0.899365	0.899677	0.900801
$\mu(\{x_2,x_3,x_4\})$	0.8	0.801246	0.800352	0.797892
$\mu(X)$	1	1	1	1

For space consideration, we do not list the dataset itself. The size of data sets is 100 and 81, respectively. The number of information sources is 4. Let ρ be 0, 0.0001 and 0.001 respectively. Let σ^2 be 0, 0.00096, 0.00125, 0.00625 and 0.0125, respectively. The parameter settings of our algorithms are the same as in Section 5.A. The Choquet integral is still used in this subsection.

Two original fuzzy measures are first given in Tables 20 and 21 for determining the general fuzzy measure and λ-fuzzy measure respectively. We observe the difference between both fuzzy measures with and without adding random perturbation. The running results of GPSO, λ-PSO are also listed in Tables 20 and 21. The running time of the programs ranges from 5 s to 20 s.

Another original fuzzy measure is given in Table 14. We observe the output error change with the added increasing Gaussian noise. The running results are listed in Table 22. The running time of GPSO is about 50 s.

From Tables 20–22, we see the values of all these resulting fuzzy measures are all close to the values of original fuzzy measures and the errors are acceptable. However, when there exits bigger noise, the error is notably increased. In other words, comparing with the GAs in reference [12,63], our algorithms shown in this paper have a good recoverability but are less robust. However, our algorithm is more robust than the algorithms in [37,16], which can be seen from Table 22. More numerical experiments show that, on average, the robustness of our proposed methods is a little weaker than the GAs but there is no essential difference. The robustness of GAs depends strongly on their mutation probability and crossover mechanism, and sometimes the robustness of PSO is a little better than the robustness of GAs. Another mentioned phenomenon is that GAs has the weak sensitivity to the extreme data (bigger perturbations) and our algorithms are a little sensitive to the extreme data. One possible reason is that the crossover and mutation operations in GAs effectively resist the impact of extreme data. It implies the robustness of our PSO-algorithms for determining specific types of fuzzy measures may be improved by incorporating the crossover or mutation operation of GAs into our algorithms.

The essential reason hidden behind these results remains to be studied further. Practically, to determine fuzzy measures or general set functions, if there does not exits extreme data or exits smaller noise, we suggest using particle swarm algorithms; if there exits bigger noise, we suggest using GAs.

5.5. The discussion of parameters of our algorithms

In our algorithms, there are not many parameters that need to be tuned. Only the following several parameters need to be taken care of: maximum velocity Vmax that controls the particle position (independent variable) in its range; inertia weight w that balances the global exploration and local exploitation; cognition learning rate C1 and social learning rate C2 that control the particle ability of learning itself and its neighbors, respectively.

In general, a recommended choice for learning factors is integer 2 as mentioned in [25], since it on average makes the weights for social and cognition parts to be 1. Under this condition, the particles statistically contract swarm to the current global best position until another particle takes over from which time all the particles statistically contract to the new global best position. In our algorithms, we set C1 = 2 and C2 = 2.

As mentioned in Section 4, the inertia weight is the direct factor of influencing on the performance of PSO; while the maximum velocity is an indirect factor. In general, we can set the maximum velocity and the value of dynamic range of each variable, and then tune the ability of global exploration and the local exploitation by changing the inertia weight. In our algorithms, we set Vmax = 1 in GPSO, λ-PSO and FEPSO, while set Vmax = 0.02 in BPPSO. The reason for this is because there is a constraint of the sum of a basic probability assignment values to be 1 for belief and plausible measure, which leads to some

Table 21
The original fuzzy measure and results of λ-PSO.

	Original	$\rho = 0$	$\rho = 0.001$	$\rho = 0.01$
T		17	22	21
e		8.92e−5	2.85e−4	2.9e−3
λ	−0.43805	−0.43764	−0.43756	−0.44002
$\mu(\{x_1\})$	0.2	0.200025	0.199781	0.200386
$\mu(\{x_2\})$	0.4	0.399822	0.399914	0.400478
$\mu(\{x_3\})$	0.1	0.099913	0.100189	0.099798
$\mu(\{x_4\})$	0.5	0.499979	0.499826	0.500543

Table 22
The results of four algorithms with Gaussian noise.

σ^2\Algorithm	Mori and Murofushi	Grabisch	GPSO	GA[22]
0.0	0.0000	1.4E−7	6.1792E−14	5.13779E−5
0.00096	0.00087	0.00083	7.318E−4	0.00012567
0.00125	0.0117	0.0108	9.4088E−4	0.000152595
0.00625	0.0605	0.0530	0.0046	0.000511731
0.01250	0.1211	0.1054	0.0094	0.00100916

Table 23
The results of GPSO and MGD on example 13.

	MGD	GPSO
$\mu(\{x_1\})$	.1000001	9.9999E−02
$\mu(\{x_2\})$	.2105010	.2105011
$\mu(\{x_3\})$	.2351913	.2353014
$\mu(\{x_4\})$	.6670005	.6667103
$\mu(\{x_1,x_2\})$	.3000154	.3000005
$\mu(\{x_1,x_3\})$	.3232821	.3235046
$\mu(\{x_1,x_4\})$	.7332942	.7333041
$\mu(\{x_2,x_3\})$	.4211123	.4211023
$\mu(\{x_2,x_4\})$	.8070128	.8070075
$\mu(\{x_3,x_4\})$	.8235020	.8235018
$\mu(\{x_1,x_2,x_3\})$	.5000010	.5000102
$\mu(\{x_1,x_2,x_4\})$	.8667019	.8666805
$\mu(\{x_1,x_3,x_4\})$	.8824023	.8824017
$\mu(\{x_2,x_3,x_4\})$	.9473951	.9474011
Processing time	1.7 s	102 s
e	9.287915E−07	9.752643E−07
Number of generation	64	142

Table 24
The results of GPSO and MGD on example 14.

	MGD	GPSO
$\mu(\{x_1\})$	0.0999012	0.1000012
$\mu(\{x_2\})$	0.2113978	0.2105013
$\mu(\{x_3\})$	0.2354001	0.2353016
$\mu(\{x_4\})$	0.6674011	0.6667009
$\mu(\{x_1,x_2\})$	0.3003967	0.3000008
$\mu(\{x_1,x_3\})$	0.3237102	0.3234993
$\mu(\{x_1,x_4\})$	0.7332964	0.7333007
$\mu(\{x_2,x_3\})$	0.4210000	0.4211006
$\mu(\{x_2,x_4\})$	0.8066675	0.8070004
$\mu(\{x_3,x_4\})$	0.8233015	0.8235000
$\mu(\{x_1,x_2,x_3\})$	0.4999979	0.5000005
$\mu(\{x_1,x_2,x_4\})$	0.8664878	0.8667001
$\mu(\{x_1,x_3,x_4\})$	0.8822984	0.8824124
$\mu(\{x_2,x_3,x_4\})$	0.9474013	0.9474117
Processing time	5.4 s	312 s
e	2.9512845E−8	9.9652483E−7
Number of generation	86	138

Table 25
The results of GPSO and MGD on example 15.

		Successful rate (%)	Number of iteration
Example 15	MGD	87	68
	GPSO	100	139

Table 26
Data of example 17.

Set	Value of v
ϕ	0
$\{x_1\}$	0.09
$\{x_2\}$	
$\{x_3\}$	
$\{x_4\}$	
$\{x_5\}$	0.1
$\{x_1,x_2\}$	
$\{x_1,x_3\}$	
$\{x_1,x_4\}$	
$\{x_1,x_5\}$	
$\{x_2,x_3\}$	0.122
$\{x_2,x_4\}$	
$\{x_2,x_5\}$	
$\{x_3,x_4\}$	0.436
$\{x_3,x_5\}$	
$\{x_4,x_5\}$	
$\{x_1,x_2,x_3\}$	
$\{x_1,x_2,x_4\}$	
$\{x_1,x_2,x_5\}$	
$\{x_1,x_3,x_4\}$	
$\{x_1,x_3,x_5\}$	
$\{x_1,x_4,x_5\}$	0.679
$\{x_2,x_3,x_4\}$	
$\{x_2,x_3,x_5\}$	
$\{x_2,x_4,x_5\}$	
$\{x_3,x_4,x_5\}$	
$\{x_1,x_2,x_3,x_4\}$	
$\{x_1,x_2,x_3,x_5\}$	
$\{x_1,x_2,x_4,x_5\}$	0.769
$\{x_1,x_3,x_4,x_5\}$	
$\{x_2,x_3,x_4,x_5\}$	
$\{x_1,x_2,x_3,x_4,x_5\}$	1

Table 27
The results on example 16.

	BPSO	GPSO	λ-PSO	BPPSO
successful rate (%)	67	100	100	100
number of iteration	325	74	89	67

Table 28
The results on example 17.

	BPSO	EFPSO
Successful rate (%)	100	100
Number of iteration	365	13

smaller measure values, while for general fuzzy measure, λ-fuzzy measure and revising fuzzy measure there is not this constraint, so smaller Vmax for BPPSO and larger Vmax for GPSO, λ-PSO and FEPSO avoid the particles flying over the best position and enhancing the ability of global exploration, respectively. In addition, the inertia weight often adopts a linear decreasing strategy from 0.9 to 0.4 with time. In fact, selecting the inertial weight is related to the adopted strategy and is problem-dependent. In determining fuzzy measures, considering the inertia weight model and the complex performance, we adopt a nonlinear strategy. Under the condition, we suggest selecting $w_{initial}$ in $[0.1,0.3]$ and w_{final} in $[-0.4,-0.2]$.

Table 29
The brief summarizations to the excellences and disadvantages of GPSO, λ-PSO, BPPSO and EFPSO.

	The type of fuzzy measure	Excellence	Disadvantage
GPSO	General fuzzy measure based on the type-II data	(1) Apply sigmoid function to restrict measure values beyond [0,1] (faster convergent speed) (2) Reinitialize the velocity of particle randomly in every iteration (stronger stability)	Not for large scale problem
λ-PSO	λ-fuzzy measure based on the type-II data	Introduce the special strategy to deal with λ-rule (stronger stability)	Only for λ-fuzzy measure
BPPSO	Belief and plausible fuzzy measure based on the type-II data	Make the particles stratify the conditions of basic probability assignments (stronger stability)	Slower convergent speed and only for belief and plausible fuzzy measure
EFPSO	Extension and fitting of fuzzy measure based on the Type-I Data	Introduce the special strategy to satisfy the type of fuzzy measure wanted to determine (faster convergent speed)	Higher time-complexity

5.6. The analysis of performance of our algorithms

BPSO is the basic PSO algorithm which firstly is used to solve the problem of determining the fuzzy measure. The experimental results demonstrate it is feasible. Based on the different types of fuzzy measures, we improved BPSO and proposed four algorithms: GPSO, λ-PSO, BPPSO and FEPSO. Now, we briefly summarize their characteristics including the type of fuzzy measure, excellence and disadvantage as Table 29.

6. Conclusions and future research

The main goals of this paper are to apply the PSO to determine some special types of fuzzy measures from data and to prove that the determination is feasible and effective. Based on GPSO and the structural characteristics of the special types of fuzzy measures, we design λ-PSO, BPPSO and FEPSO to learn λ-fuzzy measures, belief measures and to revise fuzzy measures respectively. By the analysis and comparison, we verify that the determination is feasible and the designed algorithms have merits of easy coding, lower time complexity, faster convergent speed and stronger stability than GAs in past studies [51,57,62]. In addition, compared with MGD, our algorithms are more robust.

Our scheduled further development in this research topic includes two aspects. Firstly, inspired by the paper [12], we try to design a general particle swarm algorithm with self-adaptive parameters for determining most special types of fuzzy measures. Secondly, by investigating some fundamental issues of PSO and incorporating some operation mechanism in GAs into PSO, we try to further improve the robustness of the PSO for fuzzy measure determination.

Acknowledgments

This research is supported by the National Natural Science Foundation of China (60903088, 60903089), by the Natural Science Foundation of Hebei Province (F2010000323, F2008000635), and by the Key Project Foundation of Applied Fundamental Research of Hebei Province (08963522D).

References

[1] S.H. Alavi, J. Jassbi, P.J.A. Serra, R.A. Ribeiro, Defining fuzzy measures: a comparative study with genetic and gradient descent algorithms, Intelligent Systems at the Service of Mankind, vol. 3, Springer, 2009.
[2] G. Beliakov, Fitting fuzzy measures by linear programming. Programming library fmtools, in: Proceedings of IEEE International Conference on Fuzzy Systems, 2008, pp. 862–867.
[3] B. Biggio, G. Fumera, F. Roli, Multiple classifier systems for robust classifier design in adversarial environments, International Journal of Machine Learning and Cybernetics 1 (1–4) (2010) 27–41.
[4] J. Bock, J. Hettenhausen, Discrete particle swarm optimisation for ontology alignment, Information Sciences, in press. doi:10.1016/j.ins.2010.08.013.
[5] G. Büyüközkan, D. Ruan, Choquet integral based aggregation approach to software development risk, Information Sciences 180 (3) (2010) 441–451.
[6] A. Chatterjee, P. Siarry, Nonlinear inertia weight variation for dynamic adaptation in particle swarm optimization, Computer & Operations Research 33 (2006) 859–871.
[7] M. Chen, Y. Liu, Q. Luo, A novel hybrid algorithm with marriage of particle swarm optimization and extremal optimization, Optimization Community e-print, 2007.
[8] T. Chen, Static and dynamic analyzes of importance assessing for evaluation criteria and applications on transportation, Ph.D. dissertation, Inst. Traffic and Transportation, National Chiao Tung University, Hsinchu, Taiwan, R.O.C, 1998.
[9] T. Chen, J.C. Wang, Identification of λ-fuzzy measure using sampling design and genetic algorithms, Fuzzy Sets and Systems 123 (2001) 321–334.
[10] T. Chen, J.C. Wang, G.H. Tzeng, Identification of general fuzzy measures by genetic algorithms based on partial information, IEEE Transactions on Systems, Man, And Cybernetics 30 (4) (2000) 517–528.
[11] S.B. Cho, J.H. Kim, Combining multiple neural networks by fuzzy integral for robust classification, IEEE Transaction on Systems, Man, and Cybernetics 25 (2) (1995) 380–384.
[12] E.F. Combarro, P. Miranda, Identification of fuzzy measures from sample data with genetic algorithms, Computer & Operations Research 33 (2006) 3046–3066.
[13] S.Y. Daniel, X.Z. Wang, E.C.C. Tsang, Handling interaction in fuzzy production rule reasoning, IEEE Transaction on Systems, Man, and Cybernetics – Part B: Cybernetics 34 (5) (2004) 1979–1987.

[14] Y. Dong, J. Tang, Baodong Xu, Dingwei Wang, An application of particle swarm optimization to nonlinear programming, Computers and Mathematics with Application 49 (2005) 1655–1668.
[15] M. Friedman, M. Ma, A. Kandel, On typical values and fuzzy integrals, IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics 27 (4) (1997) 703–705.
[16] M. Grabisch, A new algorithm for identifying fuzzy measures and its application to pattern recognition, in: Processings of FUZZ-IEEE/IFES'95, 1, Yokohama, Japan, 1995, pp. 145–150.
[17] M. Grabisch, Nicolas J. Nicolas, Classification by fuzzy integral: performance and tests, Fuzzy Sets and Systems 65 (2–3) (1994) 255–271.
[18] M. Grabisch, The application of fuzzy integrals in multicriteria decision making, European Journal of Operational Research 89 (1996) 445–456.
[19] Q.H. Hu, W. Pan, S. An, P.J. Ma, J.M. Wei, An efficient gene selection technique for cancer recognition based on neighborhood mutual information, International Journal of Machine Learning and Cybernetics 1 (1–4) (2010) 63–74.
[20] X. Hu, R. Eberhart, Solving constrained nonlinear optimization problem with particle swarm optimization, in: Proceedings of the Sixth World Multi-conference on Systemic, Cybernetics and Informatics, 2002.
[21] K. Ishii, M. Sugeno, A model of human evaluation process using fuzzy measure, International Journal of Man–Machine Studies 22 (1985) 19–38.
[22] C.F. Juang, A hybrid of genetic algorithm and particle swarm optimization for recurrent network design, IEEE Transactions on Systems Man and Cybernetics – Part B: Cybernetics 34 (2) (2004) 997–1006.
[23] J.M. Keller, J. Osborn, Training the fuzzy integral, International Journal of Approximate Reasoning 15 (1996) 1–24.
[24] J.M. Keller, H. Qiu, H. Tahani, Fuzzy integral and image segmentation, in: Proceedings of NAFIPS'86, New Orleans, 1986, pp. 324–338.
[25] J. Kennedy, R.C. Eberhart, Particle swarm optimization, in: Proceedings of the IEEE International Conference on Neural Networks, 1995, pp. 1942–1948.
[26] R.A. Krohling, S.C.L. Dos, Coevolutionary particle swarm optimization using Gaussian distribution for solving constrained optimization problems, IEEE Transaction on Systems, Man, and Cybernetics – Part B: Cybernetics 36 (6) (2006) 1407–1416.
[27] K.M. Lee, H. Leekwang, Genetic algorithms for fuzzy measure identification, in: Proceedings of the 3rd Internet Conference on Fuzzy Logic, Neural Nets and Soft Computing (IIZUKA'94), 1994, pp. 461–463.
[28] K.M. Lee, H. Leekwang, Identification of λ-fuzzy measure by genetic algorithms, Fuzzy Sets and Systems 75 (1995) 301–309.
[29] W. Leong, G.G. Yen, PSO-based multiobjective optimization with dynamic population size and adaptive local archives, IEEE Transaction on Systems, Man, and Cybernetics – Part B: Cybernetics 38 (6) (2008) 1270–1293.
[30] K.S. Leung, M.L. Wong, W. Lam, Z.Y. Wang, K. Xu, Learning nonlinear multiregression networks based on evolutionary computation, IEEE Transaction on Systems, Man, and Cybernetics – Part B: Cybernetics 32 (5) (2002) 630–644.
[31] H. Liu, Y. Jheng, W. Lin, G. Chen, A novel fuzzy measure and its Choquet integral regression model, in: Proceedings of the Sixth International Conference on Machine Learning and Cybernetics, Hong Kong, 2007, pp. 1394–1398.
[32] Z.Q. Liu, L.T. Bruton, J.C. Bezdek, J.M. Keller, S. Dance, N.R. Bartley, C. Zhang, Dynamic image sequence analysis using fuzzy measures, IEEE Transaction on Systems, Man, and Cybernetics – Part B: Cybernetics 31 (4) (2001) 557–572.
[33] A.M. Magdi, W. Xiao, Q-measures: an efficient extension of the Sugeno-measure, IEEE Transactions on Fuzzy Systems 11 (2003) 419–426.
[34] A. Mendez-Vazquez, P. Gader, Maximum A Posteriori EM MCE Logistic LASSO for learning fuzzy measures, in: Proceedings of IEEE International Conference on Fuzzy Systems, 2008, pp. 2007–2013.
[35] O. Mendoza, P. Melin, G. Licea, A hybrid approach for image recognition combining type-2 fuzzy logic, modular neural networks and the Sugeno integral, Information Sciences 179 (13) (2009) 2078–2101.
[36] P. Miranda, M. Grabisch, P-symmetric fuzzy measures, in: Proceedings of the ninth International Conference of Information Processing and Management of Uncertainty in Knowledge-Based Systems, Annecy, France, 2002, pp. 545–552.
[37] T. Mori, T. Murofushi, An analysis of evaluation model using fuzzy measure and the Choquet integral, in: Proceedings of the 5th Fuzzy Systems Symposium, Kobe, Japan, June 2–3 1989 (in japanese).
[38] Y. Shi, R.C. Eberhart, Fuzzy adaptive particle swarm optimization, in: Proceedings of Congress on Evolutionary Computation Seoul, Korea, Piscataway, NJ, IEEE Service Center, 2001, pp. 101–106.
[39] Y. Shi, R.C. Eberhart, Parameter selection in particle swarm optimization, Lecture Notes in Computer Science 1447 (1998) 591–600.
[40] Y. Shi, H.C. Liu, L. Gao, G.H. Zhang, Cellular particle swarm optimization, Information Sciences 181 (20) (2011) 4457–4490.
[41] Y.Shi, R.C. Eberhart, A modified particle swarm optimizer, in: Proceedings of the 1998 IEEE International Conference on Evolutionary Computation, 1998, pp. 69–73.
[42] S. Sundar, A. Singh, A swarm intelligence approach to the quadratic minimum spanning tree problem, Information Sciences 180 (17) (2010) 3182–3191.
[43] H. Tahani, J.M. Keller, Information fusion in computer vision using the fuzzy integral, IEEE Transaction on Systems, Man, and Cybernetics 20 (3) (1990) 733–741.
[44] D.L. Tong, R. Mintram, Genetic algorithm-neural network (GANN): a study of neural network activation functions and depth of genetic algorithm search applied to feature selection, International Journal of Machine Learning and Cybernetics 1 (1–4) (2010) 75–87.
[45] V. Torra, Y. Narukawa, Modeling Decisions: Information Fusion and Aggregation Operators (Cognitive Technologies), Springer, 2007.
[46] J. Wang, Z.Y. Wang, Detecting constructions of nonlinear integral systems from input-output data: an application of neural networks, In: Proceedings of the NAFIPS'96., Berkeley, 1996, pp. 559–563.
[47] J. Wang, Z.Y. Wang, Using neural networks to determine Sugeno measures by statistics, Neural Networks 10 (1) (1997) 183–195.
[48] J.C. Wang, T.Y. Chen, Bisection algorithms for solving λ–fuzzy measures, in: JCIS-2006 Proceedings of Advances in Intelligent Systems Research, October 2006.
[49] L.J. Wang, An improved multiple fuzzy NNC system based on mutual information and fuzzy integral, International Journal of Machine Learning and Cybernetics 1 (2) (2011) 25–36.
[50] S.M. Wang, J. Watada, A hybrid modified PSO approach to VaR-based facility location problems with variable capacity in fuzzy random uncertainty, Information Sciences, in press. doi:10.1016/j.ins.2010.02.014.
[51] W. Wang, Z.Y. Wang, G.J. Klir, Genetic algorithms for determining fuzzy measures from data, Journal of Intelligent and Fuzzy Systems 6 (2) (1998) 171–183.
[52] X.Z. Wang, S.X. Lu, J.H. Zhai, Fast fuzzy multi-category SVM based on support vector domain description, International Journal of Pattern Recognition and Artificial Intelligences 22 (1) (2008) 109–120.
[53] X.Z. Wang, J.H. Zhai, S.X. Lu, Induction of multiple fuzzy decision trees based on rough set technique, Information Sciences 178 (16) (2008) 3188–3202.
[54] X.Z. Wang, S.F. Zhang, J.H. Zhai, A nonlinear integral defined on partition and its application to decision trees, Soft Computing 11 (4) (2007) 317–321.
[55] Z.Y. Wang, G.J. Klir, Fuzzy measure theory, Plenum Press (A division of Plenum Publishing Corporation), New York, 1992.
[56] Z.Y. Wang, K.S. Leung, J. Wang, Determining nonnegative monotone set functions based on Sugeno's integral: an application of genetic algorithms, Fuzzy Sets and Systems 112 (2000) 155–164.
[57] Z.Y. Wang, K.S. Leung, J. Wang, Genetic algorithms used for determining belief measures and plausibility measures, in: Proceedings of NAFIPS'97, Syracuse, 1997, pp. 195–198.
[58] Z.Y. Wang, K.S. Leung, J. Wang, Genetic algorithms used for determining non-additive set functions in information fusion, in: Proceedings of IFSA'97, 1999, pp. 518–521.
[59] Z.Y. Wang, K.S. Leung, M.L. Wong, J. Fang, A new type of nonlinear integrals and computational algorithm, Fuzzy Sets and Systems 112 (2000) 223–231.
[60] Z.Y. Wang, K.S. Leung, M.L. Wong, J. Fang, K. Xu, Nonlinear nonnegative multiregressions based on Choquet integrals, International Journal of Approximate Reasoning 25 (2000) 71–87.

[61] Z.Y. Wang, J. Wang, Using genetic algorithm for extension and fitting of belief measures and plausibility measures, in: Proceedings of NAFIPS'96, Berkeley, 1996, pp. 348–350.
[62] Z.Y. Wang, J. Wang, Using genetic algorithms for λ-fuzzy measure fitting and extension, in: Proceedings of FUZZ/IEEE'96, New Orleans, 1996, pp. 1871–1874.
[63] Z.Y. Wang, K. Xu, J. Wang, G.J. Klir, Using genetic algorithms to determine nonnegative monotone set functions for information fusion in environments with random perturbation, International Journal of Intelligence Systems 14 (1999) 949–962.
[64] Z.Y. Wang, K. Xu, P.A. Heng, K.S. Leung, Indeterminate integrals with respect to nonadditive measures, Fuzzy Sets and Systems 138 (2003) 485–495.
[65] K. Xu, Z.Y. Wang, P.A. Heng, K.S. Leung, Classification by nonlinear integral projections, IEEE Transactions on Fuzzy Systems 11 (2) (2003) 187–201.
[66] Z. Xu, Choquet integrals of weighted intuitionistic fuzzy information, Information Sciences 180 (5) (2010) 726–736.
[67] R. Yager, On ordered weighted averaging aggregation operators in multicriteria decision making, IEEE Transactions on Systems, Man, and Cybernetics 18 (1988) 183–190.
[68] R. Yager, Uncertainty representation using fuzzy measures, IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics 32 (1) (2002) 13–20.
[69] R. Yang, Z.Y. Wang, P.A. Heng, K.S. Leung, Classification of heterogeneous fuzzy data by Choquet integral with fuzzy-valued integrand, IEEE Transactions on Fuzzy Systems 15 (5) (2007) 931–942.
[70] O.Y. Yao, R. Mesiar, H. Agahi, An inequality related to Minkowski type for Sugeno integrals, Information Sciences 180 (14) (2010) 2793–2801.
[71] L. Zhang, H. Yu, D. Chen, S. Hu, Analysis and improvement of particle swarm optimization algorithm, Information and Control 33 (5) (2004) 513–517 (in Chinese).
[72] H.Y. Zhao, X.Z. Wang, K. Wei, Using a particle swarm algorithm to determine fuzzy measure from data, Fuzzy Systems and Mathematics 22 (A) (2008) 357–362 (in Chinese).
[73] J. Zhu, X.P. Li, W.M. Shen, Effective genetic algorithm for resource-constrained project scheduling with limited preemptions, International Journal of Machine Learning and Cybernetics, in press. doi:10.1007/s13042-011-0014-3.