Hindawi Publishing Corporation
Advances in Fuzzy Systems
Volume 2013, Article ID 158969, 17 pages
http://dx.doi.org/10.1155/2013/158969

Research Article

A Proposal to Speed up the Computation of the Centroid of an Interval Type-2 Fuzzy Set

Carlos E. Celemin and Miguel A. Melgarejo

Laboratorio de Automática e Inteligencia Computacional, Universidad Distrital Francisco Jose de Caldas, Carrera 8 No. 40-62, Piso 7, Bogota, Colombia

Correspondence should be addressed to Miguel A. Melgarejo; migbet@gmail.com

Received 28 August 2012; Accepted 21 October 2012

Academic Editor: M. Onder Efe

This paper presents two new algorithms that speed up the centroid computation of an interval type-2 fuzzy set. The algorithms include precomputation of the main operations and initialization based on the concept of uncertainty bounds. Simulations over different kinds of footprints of uncertainty reveal that the new algorithms achieve computation time reductions with respect to the Enhanced-Karnik algorithm, ranging from 40 to 70%. The results suggest that the initialization used in the new algorithms effectively reduces the number of iterations to compute the extreme points of the interval centroid while precomputation reduces the computational cost of each iteration.

1. Introduction

Type-2 fuzzy logic systems (T2-FLS) theory and its applications have grown in recent years [1–3]. One of the main problems related to the implementation of these systems is type reduction, which computes the generalized centroid of a type-2 fuzzy set (T2-FS) [4–6]. This operation becomes computationally simpler when performed over a particular class of T2-FS, namely, interval type-2 fuzzy set (IT2-FS) [5, 7]. Basically, the centroid of an IT2-FS is an interval [3, 8]. Therefore, computing this centroid can be considered as an optimization problem that finds the extreme points that define the interval [6].

Fast computation of type reduction for IT2-FSs is an attractive problem, which is critical since type-reduction procedures for more general type-2 fuzzy sets (T2-FSs) make use of interval type-2 fuzzy computations [4, 9–11]. Up-to-date, several iterative approaches to computing type reduction for IT2-FSs have been proposed [5, 12–18]. The Karnik-Mendel (KM) algorithms are the most popular procedures used in interval type-2 fuzzy logic systems (IT2-FLS) [17]. It has been demonstrated that the KM algorithms converge monotonically and superexponentially fast. These properties are highly desirable in iterative algorithms [13]. However, KM procedures converge in several iterations and demand a considerable amount of arithmetic and memory look-up operations [7, 8, 12, 19], for real IT2-FLS implementations.

In the case of IT2 fuzzy controllers, the computational cost of type reduction is an important subject [20, 21]. The overall complexity of the controller largely depends on the type-reduction and defuzzification stages. Thus developing strategies for reducing the computational burden and necessary resources for implementing these two stages is highly convenient. In addition, there are other applications of IT2-FLS in which the complexity of the hardware and software platforms should be reduced in order to guarantee the fastest possible execution of fuzzy inferences [22, 23].

Noniterative type reduction of IT2-FS can be achieved by means of uncertainty bounds [7]; however, this method is approximate and involves relatively complex computations. Other methods for computing the centroid of an IT2-FS have been proposed, for example, geometric type-reduction [4], genetically optimized type-reducers [8], interval-analysis-based type-reduction [24], and sampling defuzzification and collapsing defuzzification [12]. Although these alternative methods exhibit interesting properties, they are not as popular as the KM algorithm.

An enhanced version of the KM algorithm (EKM) was presented in [16]. This version includes a better initialization and some computational improvements. Experimental evidence shows that the EKM algorithm saves about two iterations, which means a reduction in computation time of more than 39% with respect to the original algorithm. By the time the EKM algorithm was presented, and the iterative algorithm with stop condition (IASC) was also introduced by Melgarejo et al. [14, 15, 19]. This algorithm deals with the problem of computing type-reduction for an IT2-FS by using some properties of the centroid function [13, 19]. Experimental evidence showed that IASC is faster than the EKM algorithm in some cases.

Recently, IASC has been enhanced by Wu and Nie [18] leading to a new version called EIASC, which proved to be the fastest algorithm with respect to the IASC and EKM algorithms for several type-2 fuzzy logic applications. The improvement introduced in EIASC mainly deals with modifying the iterative procedure for computing the right point of the interval centroid. Although IASC and EIASC outperform the EKM algorithm, the initialization of the IASC-type algorithms is somewhat trivial and does not provide an appropriate starting point, which can limit the convergence speed of these algorithms when calculating the points of the type-reduced set. An extensive and interesting comparison of several type-reduction methods is provided by Wu in [25]. This study confirmed the convenience of EIASC over the EKMA for reducing the computational cost of an IT2-FLS. The experiments that were limited to interpreted language implementations also showed that EIASC and the enhanced opposite direction searching algorithm (EODS) exhibited similar performance over different applications; however, the EODS outperformed EIASC in low-resolution cases [26].

Regarding the aforementioned problem, this work considers a different initialization perspective for iterative type-reduction algorithms based on the concept of inner-bound set for a type-reduced set [13]. This kind of initialization has not been tried in type-reduction computing until now. As it will be analyzed later, the inner-bound initialization reduces the distance that the algorithms need to cover in order to find the optimal points of the interval centroid. In addition, bearing in mind the computational burden of iterative algorithms, this paper also focuses on proposing an alternative strategy to speed up their computation based on the concept of precomputing. Precomputation in algorithmic design is a powerful concept that reduces the necessary arithmetic operations. Current literature does not report the use of precomputation in the type-reduction stage in IT2-FLSs.

Using inner bound sets initialization and precomputation, Both the IASC and KM algorithms have been restated, leading to faster algorithms hereafter refered to IASC2 and KMA2. Our experiments considering two different implementation perspectives (i.e., interpreted language and compiled language) show that IASC2 and KMA2 outperform EKMA and EIASC. In fact, timing results suggest that IASC2 may be the best option for implementing IT2-FLSs in dedicated hardware, whereas EKMA may support the computation of large-scale simulations of IT2-FLSs over general purpose processors.

The paper is organized as follows. Section 2 provides some background about the centroid of an interval type-2 fuzzy set. IASC2 and KMA2 are presented in Section 3. Section 4 is devoted to a computational comparative study of iterative type-reduction methods which considers two types of implementation: compiled-language-based and interpreted-language-based. Finally, we draw conclusions in Section 5.

2. Background

The purpose of this section is to provide some basic information about the centroid of interval type-2 fuzzy sets. The reader who is not familiar with this theory is invited to consult [3, 5, 13] among others.

2.1. Type-2 Fuzzy Sets. According to Mendel [5] a type-2 fuzzy set, denoted $\widetilde{A}$, is characterized by a type-2 membership function $\mu_{\widetilde{A}}(x,u)$ where $x \in X$ and $0 \le \mu_{\widetilde{A}}(x,u) \le 1$:

$$\widetilde{A} = \left\{ \left((x,u), \mu_{\widetilde{A}}(x,u) \right) \mid \forall u \in J_x \subseteq [0,1] \right\}. \tag{1}$$

2.2. Interval Type-2 Fuzzy Sets. An IT2-FS is a particular case of a T2-FS whose values in $\mu_{\widetilde{A}}(x,u)$ are equal to one [27]. An IT2-FS is fully described by its footprint of uncertainty (FOU) [3]. Note that an IT2-FS set can be understood as a collection of type-1 fuzzy sets (i.e., traditional fuzzy sets). The membership functions of these sets are contained within the FOU; thus, they are called embedded fuzzy sets [5, 13].

2.3. Centroid of an IT2-FS. Let $\widetilde{A}$ be an IT2-FS, and its centroid $c(\widetilde{A})$ is an interval given by [5]:

$$\mathrm{c}\left(\widetilde{A}\right) = \frac{1}{\left[y_l\left(\widetilde{A}\right), y_r\left(\widetilde{A}\right) \right]} = \frac{1}{\left[y_l, y_r \right]}, \tag{2}$$

where y_L and y_R are the solutions to the following optimization problems:

$$\begin{aligned} y_L &= \min_{\forall f_i \in \left[\underline{f_i}, \overline{f_i} \right]} \frac{\sum_{i=1}^{N} x_i f_i}{\sum_{i=1}^{N} f_i}, \\ y_R &= \max_{\forall f_i \in \left[\underline{f_i}, \overline{f_i} \right]} \frac{\sum_{i=1}^{N} x_i f_i}{\sum_{i=1}^{N} f_i}, \end{aligned} \tag{3}$$

where the universe of discourse X as well as the upper and lower membership functions have been discretized in N points.

A simpler way to understand the centroid of an IT2-FS is to think about it as a set of centroids that are computed from all the embedded fuzzy sets [3, 5]. Clearly, in order to characterize an interval set, it is only necessary to find its minimum and maximum. The Karnik-Mendel iterative algorithms [5, 16, 17] are the most popular procedures for computing these two points.

2.4. Computing the Centroid of an IT2-FS. It has been demonstrated that y_L and y_R can be computed in terms of the upper and lower membership functions of the footprint of uncertainty (FOU) of $\widetilde{A}$ [3, 5] as follows:

$$y_L = \min_{\forall f_i \in [\underline{f_i}, \overline{f_i}]} \frac{\sum_{i=1}^{N} x_i f_i}{\sum_{i=1}^{N} f_i} = \frac{\sum_{i=1}^{L} x_i \overline{f_i} + \sum_{i=L+1}^{N} x_i \underline{f_i}}{\sum_{i=1}^{L} \overline{f_i} + \sum_{i=L+1}^{N} \underline{f_i}},$$
$$y_R = \max_{\forall f_i \in [\underline{f_i}, \overline{f_i}]} \frac{\sum_{i=1}^{N} x_i f_i}{\sum_{i=1}^{N} f_i} = \frac{\sum_{i=1}^{R} x_i \underline{f_i} + \sum_{i=R+1}^{N} x_i \overline{f_i}}{\sum_{i=1}^{R} \underline{f_i} + \sum_{i=R+1}^{N} \overline{f_i}}, \quad (4)$$

where $\underline{f_i}$ and $\overline{f_i}$ are the values of the lower and upper membership functions, respectively, considering that the universe of discourse X has been discretized into N points x_i. In addition L and R are two switch points that satisfy the following:

$$x_L \leq y_L \leq x_{L+1},$$
$$x_R \leq y_R \leq x_{R+1}. \quad (5)$$

2.5. Uncertainty Bounds of the Type-Reduced Set. The type-reduced set of an interval type-2 fuzzy given in (2) can be approximated by means of two interval sets called inner- and outer-bound sets [7]. The end y_L and y_R points of the type-reduced set of an interval type-2 fuzzy set are bounded from below and above by

$$\underline{y_l} \leq y_L \leq \overline{y_l}, \quad (6)$$

$$\underline{y_r} \leq y_R \leq \overline{y_r}, \quad (7)$$

where $[\overline{y_l}, \underline{y_r}]$ and $[\underline{y_l}, \overline{y_r}]$ are the inner- and outer-bound sets, respectively. For the purposes of this paper only the inner bound set is considered, so it is computed as follows:

$$y_A = \frac{\sum_{i=1}^{N} x_i \overline{f_i}}{\sum_{i=1}^{N} \overline{f_i}},$$
$$y_B = \frac{\sum_{i=1}^{N} x_i \underline{f_i}}{\sum_{i=1}^{N} \underline{f_i}}, \quad (8)$$

$$\overline{y_l} = \min(y_A, y_B),$$
$$\underline{y_r} = \max(y_A, y_B). \quad (9)$$

2.6. Properties of the Centroid Function. The definition and corresponding study of the continuous centroid function are provided by Mendel and Liu in [13]. Theoretical studies derived from (6) can be legitimated for the discrete centroid function [12]. Regarding this fact, Mendel and Liu demonstrated several interesting properties of the continuous centroid function, which are applicable also to the discrete one. Some of these properties can be summarized saying that the centroid function decreases or has flat spots to the left of its minimum, and it increases or has flat spots to the right of this value. Additionally, this function has a global minimum when $k = L$. The right-centroid function y_r has similar properties since this function has a global maximum when $k = R$. Thus, it increases to the left of the maximum and decreases to the right.

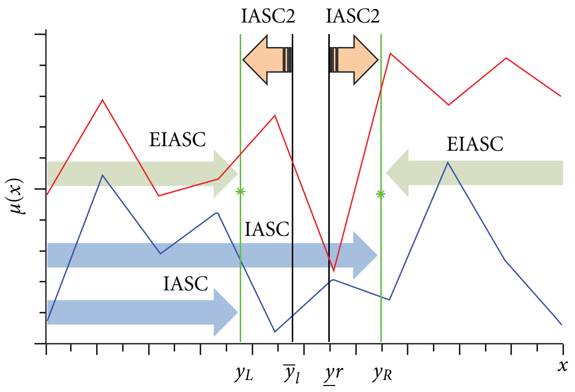


FIGURE 1: Uniform random FOU: $\overline{f_i}$ (red line) and $\underline{f_i}$ (blue line). The arrows indicates the computation of minimum and maximum. Orange arrows correspond to IASC2, green to EIASC and blue to IASC.

3. Modified Iterative Algorithms for Computing the Centroid of an Interval Type-2 Fuzzy Set

3.1. IASC2. The IASC-2 algorithm is presented in Table 1. The algorithm consists of four stages: sorting, precomputation, initialization, and recursion. Sorting and precomputation are the same for computing y_L and y_R. Precomputation stores the partial results of computing (8); only two divisions are required. Those results will be used along the other stages of the algorithm.

The initialization of IACS2 is characterized by its dependence from the inner-bound set. This feature introduces a big difference with previous algorithms since they use fixed initialization points (e.g., EKMA, IASC, and EIASC); however, regarding this particularity, IASC2 is similar to the KMA, because both algorithms include a FOU-dependent initialization.

The IASC2 works with the switch points starting inside the centroid, iterating similarly as IASC type algorithms but from the $y_{\min}$ toward left side while the minimum y_L is reached, it is in the opposite sense to IASC and EIASC. The procedure for computing the maximum y_R begins in $y_{\max}$ and goes to the right side until it is found, and the search direction is the same of IASC but opposite to the proposed in EIASC. The advantage of this initialization for computing the centroid is that always the switch points start close to the y_L and y_R; therefore, less iterations are used.

Figure 1 depicts a uniform random FOU with its y_L, y_R, $y_{\min}$, or $\overline{y_l}$ and the $y_{\max}$ or $\underline{y_r}$, and the arrows show the direction of the search of each procedure from the initial switch point to the minimum or maximum. Also, in this example, it is possible to see the space scanned by every

Table 1: IASC2 algorithm flow.

IASC2	
The minimum extreme point y_L of the centroid of an interval type-2 fuzzy set over x_i with upper membership function $\overline{f_i}$ and lower membership function $\underline{f_i}$ can be computed using the following procedure:	The maximum extreme point y_R of the centroid of an interval type-2 fuzzy set over x_i with upper membership function $\overline{f_i}$ and lower membership function $\underline{f_i}$ can be computed by the following procedure:
(1) Sort x_i $(i = 1, 2, \dots, N)$ in ascending order and call the sorted x_i by the same name, but now $x_1 \le x_2 \le \cdots \le x_N$. Match the weights f_i with their respective x_i and renumber them so that their index corresponds to the renumbered x_i.	
(2) Precomputation:	
$V[n] = \sum_{i=1}^{n} \overline{f_i}$ (15)	
$H[n] = \sum_{i=1}^{n} \underline{f_i}$ (16)	
$T[n] = \sum_{i=1}^{n} x_i \overline{f_i}$ (17)	
$Z[n] = \sum_{i=1}^{n} x_i \underline{f_i}$ (18)	
with $n = 1, 2, \dots N$.	
$y_A = \frac{T[N]}{V[N]}$ (19)	
$y_B = \frac{Z[N]}{H[N]}$ (20)	
(3) Initialization:	Initialization:
$y_{\min} = \min(y_A, y_B)$ (21)	$y_{\max} = \max(y_A, y_B)$ (32)
Find $L_i (1 \le L_i \le N-1)$ so that	Find $R_i (1 \le R_i \le N-1)$ so that
$x_{L_i} \le y_{\min} < x_{L_i+1}$	$x_{R_i} \le y_{\max} < x_{R_i+1}$
$D_l(L_i) = T[L_i] + (Z[N] - Z[L_i])$ (22)	$E_R(R_i) = Z[R_i] + (T[N] - T[R_i])$ (33)
$P_l(L_i) = V[L_i] + (Y[N] - Y[L_i])$ (23)	$Q_R(R_i) = Y[R_i] + (X[N] - X[R_i])$ (34)
$y_{\min} = \frac{D_{L_i}}{P_{L_i}}$ (24)	$y_{\max} = \frac{E_R(R_i)}{Q_R(R_i)}$ (35)
$D = D_l(L_i)$ (25)	$E = E_{R_i}$ (36)
$P = P_l(L_i)$ (26)	$Q = Q_{R_i}$ (37)
$k = L_i.$ (27)	$k = R_i + 1.$ (38)
(4) Recursion:	Recursion:
$\alpha_k = \overline{f_k} - \underline{f_k}$ (28)	$\alpha_k = \overline{f_k} - \underline{f_k}$ (39)
$D = D - x_k \alpha_k$ (29)	$E = E - x_k \alpha_k$ (40)
$P = P - \alpha_k$ (30)	$Q = Q - \alpha_k$ (41)
$y_l = \frac{D}{P}.$ (31)	$y_r = \frac{E}{Q}.$ (42)
(5) If $y_l \le y_{\min}$, then $y_{\min} = y_l$, $k = k - 1$ and go to step 4 else $y_L = y_{\min}$ and stop.	If $y_R \ge y_{\max}$, then $y_{\max} = c_R$ and $k = k + 1$ go to step 4 else $y_R = y_{\max}$ and stop.

algorithm, suggesting that this space is proportional to the amount of iterations required to converge. Note that IASC2 has the shortest arrows, which mean few iterations.

In Table 1 (21), $y_{\min}$ corresponds to the minimum of the inner-bound set; thus according to (6) $y_{\min} \ge y_L$. Therefore Table 1 (21) provides an initialization point that is always greater than or equal to the minimum extreme point y_L; In addition, it is possible to find an integer L_i so that $x_{L_i} \le y_{\min} < x_{L_i+1}$. It is easy to show that Table 1 (22)–(24) is computing the left centroid function with $k = L_i$, which fixes the starting point of recursion from the values obtained in the precomputation stage.

Recursion Table 1 (28)–(31) computes the left centroid function with initial point in $k = L_i$, and each decrement in k leads to a decrement in $y_l(k)$. By doing some algebraic operations, it is easy to show that $y_l(k) = (\sum_{i=1}^{k} x_i \overline{f_i} + \sum_{i=k+1}^{N} x_i \underline{f_i}) / (\sum_{i=1}^{k} \overline{f_i} + \sum_{i=k+1}^{N} \underline{f_i}) = (D_{L_i} - \sum_{i=k+1}^{L_i} x_i(\overline{f_i} - \underline{f_i})) / (P_{L_i} - \sum_{i=k+1}^{L_i} (\overline{f_i} - \underline{f_i}))$. If sums are computed recursively, it leads to $y_l(k) = (D_{k-1} - x_k(\overline{f_k} - \underline{f_k})) / (P_{k-1} - (\overline{f_k} - \underline{f_k}))$.

Since $L \le L_i$ given that $y_l \le y_{\min}$ in Table 1 (21), iterations should decrease k from L_i. Thus, a decrement of k induces a decrement in f_k from $\overline{f_k}$ to $\underline{f_k}$. In [5], the following was demonstrated.

Condition 1. If $x_k < y(f_1 \cdots f_N)$, then $y(f_1 \cdots f_N)$ increases when f_k decreases.

Table 2: KMA2 algorithm flow.

KMA2	
The KM algorithm for finding the minimum extreme point y_L of the centroid of an interval type-2 fuzzy set over x_i with upper membership function $\overline{f}_i$ and lower membership function $\underline{f}_i$ can be restated as follows:	The KM algorithm for finding the maximum extreme point y_R of the centroid of an interval type-2 fuzzy set over xi with upper membership function $\overline{f}_i$ and lower membership function $\underline{f}_i$ can be reexpressed as follows:
(1) Sort x_i $(i = 1, 2, \ldots, N)$ in ascending order and call the sorted x_i by the same name, but now $x_1 \leq x_2 \leq \cdots \leq x_N$. Match the weights f_i with their respective x_i and renumber them so that their index corresponds to the renumbered x_i.	
(2) Precomputation: $V[n] = \sum_{i=1}^{n} \overline{f}_i$ (43) $H[n] = \sum_{i=1}^{n} \underline{f}_i$ (44) $T[n] = \sum_{i=1}^{n} x_i \overline{f}_i$ (45) $Z[n] = \sum_{i=1}^{n} x_i \underline{f}_i$ (46) with $n = 1, 2, \ldots N$. $y_A = \frac{T[N]}{V[N]}$ (47) $y_B = \frac{Z[N]}{H[N]}$. (48)	
(3) Set $y_{\min} = \min(y_A, y_B)$ (49) Find $k (1 \leq k \leq N-1)$ so that $x_k \leq y_{\min} < x_{k+1}$ $D_l(k) = T[k] + (Z[N] - Z[k])$ (50) $P_l(k) = V[k] + (Y[N] - Y[k])$ (51) $y = \frac{D_l(k)}{P_l(k)}$. (52)	Set $y_{\max} = \max(y_A, y_B)$ (56) Find $k (1 \leq k \leq N-1)$ so that $x_k \leq y_{\max} < x_{k+1}$ $E_R(k) = Z[k] + (T[N] - T[k])$ (57) $Q_R(k) = Y[k] + (X[N] - X[k])$ (58) $y = \frac{E_R(k)}{Q_R(k)}$. (59)
(4) Find $k' \in [1, N-1]$ such that $x_{k'} \leq y < x_{k'+1}$.	Find $k' \in [1, N-1]$ such that $x_{k'} \leq y < x_{k'+1}$.
(5) If $k' = k$, stop and set $y_L = y$, else continue.	If $k' = k$, stop and set $y_R = y$, else continue.
(6) Set $D_l(k') = T[k'] + (Z[N] - Z[k'])$ (53) $P_l(k') = V[k'] + (Y[N] - Y[k'])$ (54) $y' = \frac{D_l(k')}{P_l(k')}$. (55)	Set $E_R(k') = Z[k'] + (T[N] - T[k'])$ (60) $Q_R(k') = Y[k'] + (X[N] - X[k'])$ (61) $y' = \frac{E_R(k')}{Q_R(k')}$. (62)
(7) Set $y = y'$, $k = k'$ and go to step 4.	Set $y = y'$, $k = k'$ and go to step 4.

Condition 2. If $x_k > y(f_1 \cdots f_N)$, then $y(f_1 \cdots f_N)$ increases when f_k increases.

Note that k is greater than L which implies that $x_k > y_L$; thus, it is guaranteed that $x_k > y_l(k)$ satisfying Condition 1. Therefore, it can be concluded that a decrement in f_k will induce a decrement in $y_l(k)$.

Finally, the stop condition is stated from the properties of the centroid function [13]. They indicate that the left-centroid function has a global minimum in $k = L$, so it is found when the functions start to increase.

The procedure to compute y_L is only analyzed in this section. The full demonstration of this procedure is provided in the appendix for the readers who are interested in following it. The computation of y_R obeys similar conceptual principles, and it can be understood by using the ideas suggested above or those provided in the appendix.

3.2. KMA2. The KM2 algorithm is presented in Table 2. This algorithm uses the same conceptual principles of KMA for finding y_L and y_R [5]. The algorithm includes precomputation, initialization, and searching loop. Precomputation and initialization of KMA2 are the same for IASC2, having always the initial switch points near to the extreme points y_L and y_R. The searching loop corresponds to steps 4–7. The core of the loop is Table 2 (53)–(55) and (A.2)–(A.4), which can be easily demonstrated by making L_i equal to k or k'. These equations calculate $y_l(k)$ or $y_r(k)$ with few sums and subtractions taking advantage of pre-computed values. Finally, the stop condition is the same for the EKMA algorithm, which was stated in [16].

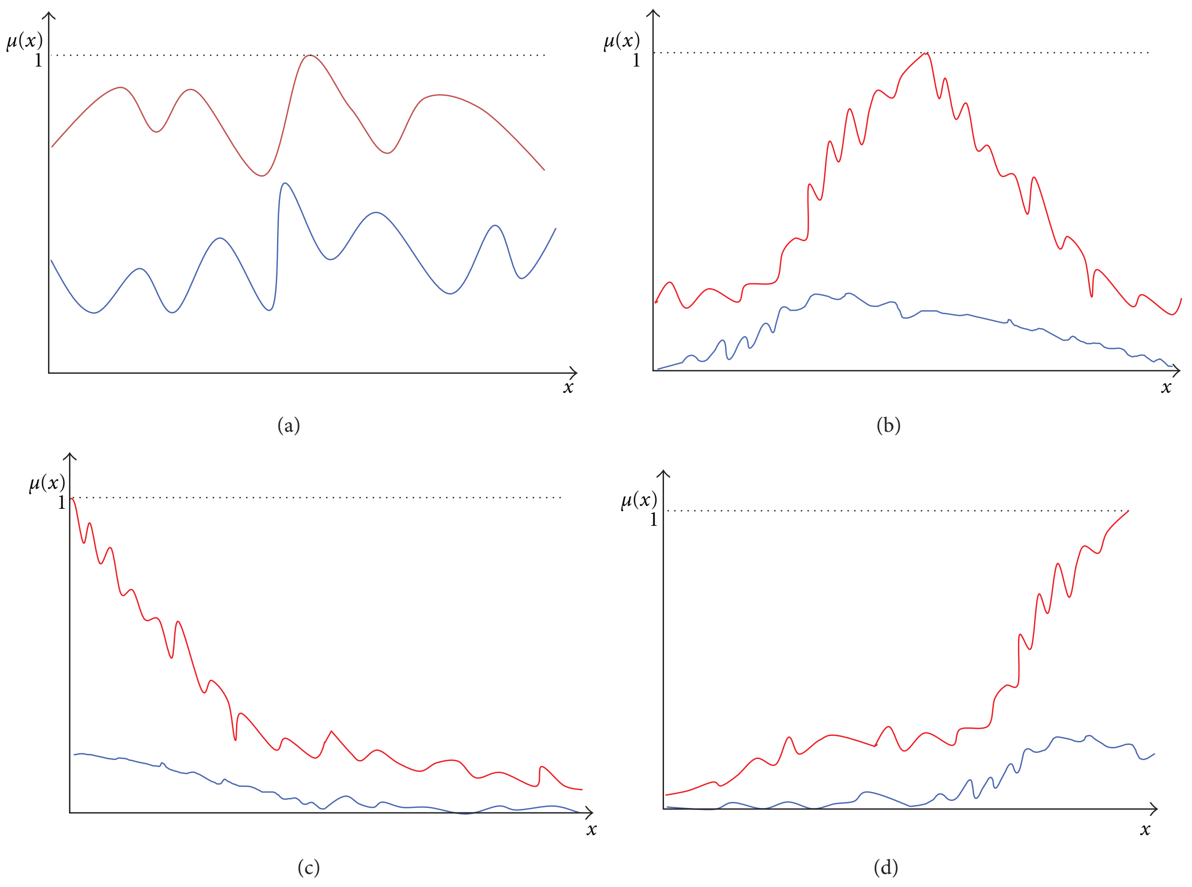


Figure 2: FOUs cases used in the experiments. (a) Uniform random FOU. (b) Centered uniform random FOU. (c) Right-sided uniform random FOU. (d) Left-sided uniform random FOU.

4. Computational Comparison with Existing Iterative Algorithms: Implementation Based on Compiled Language

4.1. Simulation Setup. The purpose of these experiments is to measure the average computing time that a type-reduction algorithm takes to compute the interval centroid of a particular kind of FOU when the algorithm is implemented as a program in a compiled language like C. In addition, complementary variables like standard deviation of the computing time and the number of iterations required by each algorithm to converge are also registered. The algorithms considered in these experiments are IASC [14], EIASC [18], and EKMA [16], which are confronted against IASC2 and KMA2. For the sake of comparison, the EKM algorithm is regarded as a reference for all the experiments.

Four FOU cases were selected to evaluate the performance of the algorithms. All cases corresponded to random FOUs modified by different envelopes. The first one considered a constant envelope that simulated the output of a fuzzy inference engine in which most of the rules are fired at a similar level. In the second one, a centered envelope was used to simulate that rules with consequents in the middle region of the universe of discourse are fired. The last two cases were proposed to simulate firing rules whose consequents are in the outer regions of the universe of discourse. For the sake of clarity, several examples of the FOUs considered in this work are depicted in Figure 2.

The following experimental procedure was applied to characterize the performance of the algorithms over each FOU case described above: we increased N from 0 to 100 with step size of 10, and then N was increased from 100 to 1000 with step size of 100. For each N, 2000 Monte Carlo simulations were used to compute y_L and y_R. In order to overcome the problem of measuring very small times, 100.000 simulations were computed in each Monte Carlo trial; thus, the time of a type-reduction computation corresponded to the measured time divided by 100.000.

The algorithms were implemented as C programs and compiled as SCILAB functions. The experiments were carried out over SCILAB 5.3.1 numeric software running over a platform equipped with an AMD Athlon(tm) 64 × 2 Dual Core Processor 4000 + 2.10 GHz, 2 GB RAM, and Windows XP operating system.

4.1.1. Results: Computation Time. The results of computation time for uniform random FOUs are presented in Figure 3. IASC2 exhibited the best performance of all algorithms for N between 10 and 100. For N greater than 100, KMA2 was the fastest algorithm followed by IASC2. The percentage

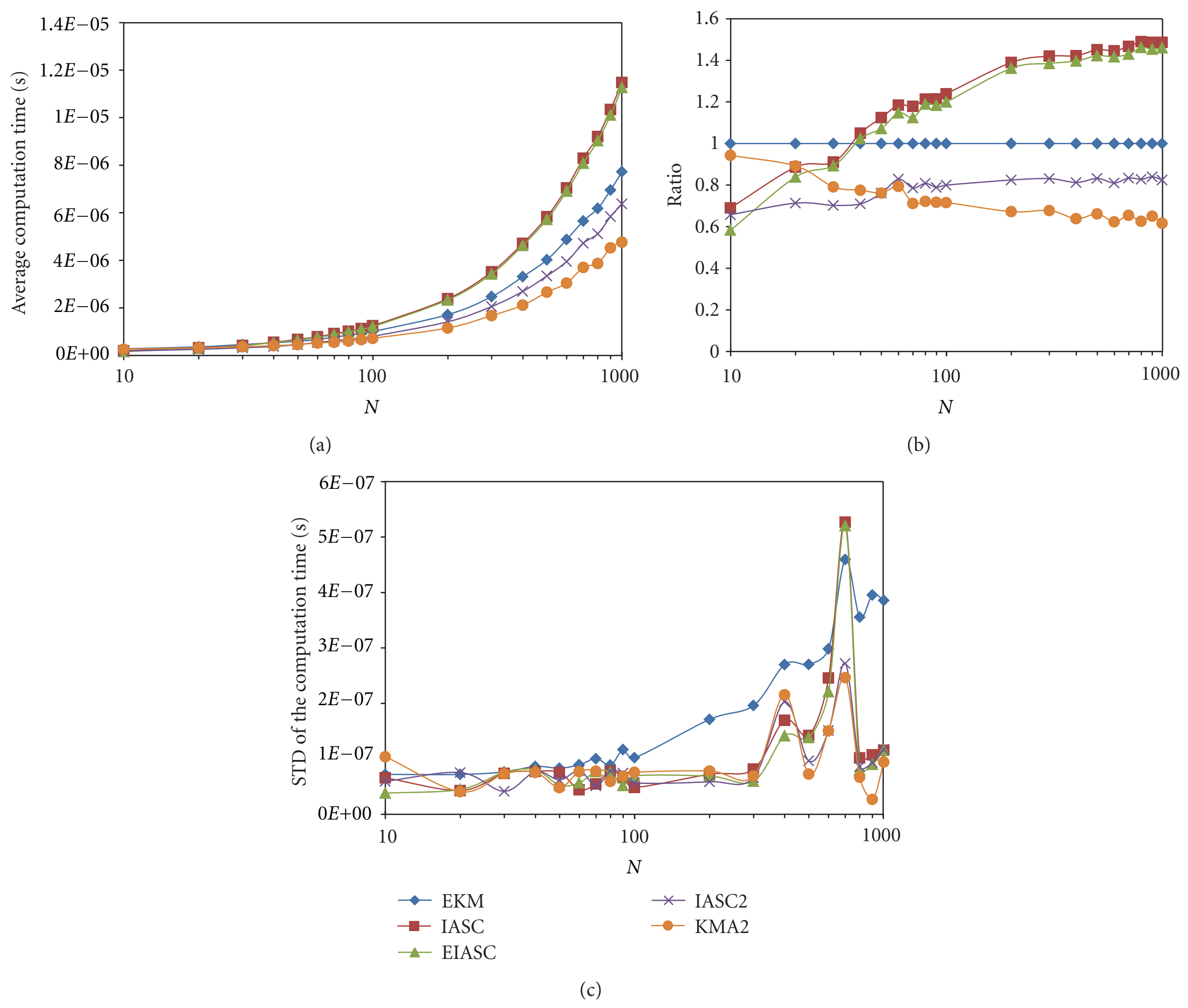


FIGURE 3: Uniform random FOU results (compiled language implementation of the algorithms). (a) Average computation time, (b) ratio with respect to the EKMA, and (c) standard deviation of the computation time.

of computation time reduction (PCTR) with respect to the EKMA (i.e., PCTR $= (t_o - t_{\text{ekm}})/t_{\text{ekm}}$ where t_o and t_e are the average computation times of a particular algorithm and EKMA, resp.) of IASC2 was about 30% for $N = 10$, while in the case of KMA2, the largest PCTR was about 40% for $N = 1000$. Note that IASC and EIASC exhibited poorer performance with respect to EKMA when N grows beyond 100 points; however EIASC was the fastest algorithm for $N = 10$ points. The standard deviation of the computation time for IASC2 and KMA2 was always smaller than that of the EKMA.

Figure 4 presents the results for centered random FOUs. IASC2 exhibited the largest PCTR for N smaller than 100 points, about 40%. When $N$was greater than 100, KMA2 was the fastest algorithm. This algorithm provided a maximum PCTR of about 50% for $N = 1000$. The standard deviation of the computation time for all algorithms was often smaller than that of EKMA.

In the case of left-sided random FOUs (Figure 5), IASC was the fastest algorithm for N smaller than 100 points. This algorithm provided a PCTR that was between 70% and 50%. Here again KMA2 was the fastest algorithm for N greater than 100 points with a PCTR that was about 45% for this region. In this case, EIASC exhibited the worst performance and, in general, the standard deviation of the computation time was similar for all the algorithms.

The results concerning right-sided random FOUs are presented in Figure 6. IASC2 was the algorithm that provided the largest PCTR for N smaller than 100 points. This percentage was between 55% and 60%. KMA2 accounted for the largest PCTR for N greater than 100 points. The reduction achieved by this algorithm was between 50% and 60%. IASC exhibited the worst performance in this case. The standard deviation of the computation time was similar in all algorithms for N smaller than 100; however, a big difference with respect to EKMA was observed for N greater than 100 points.

4.1.2. *Results: Iterations.*

The amount of iterations required by each algorithm to find y_R was recorded in all the experiments. In Figures 7–10, results for the mean of the

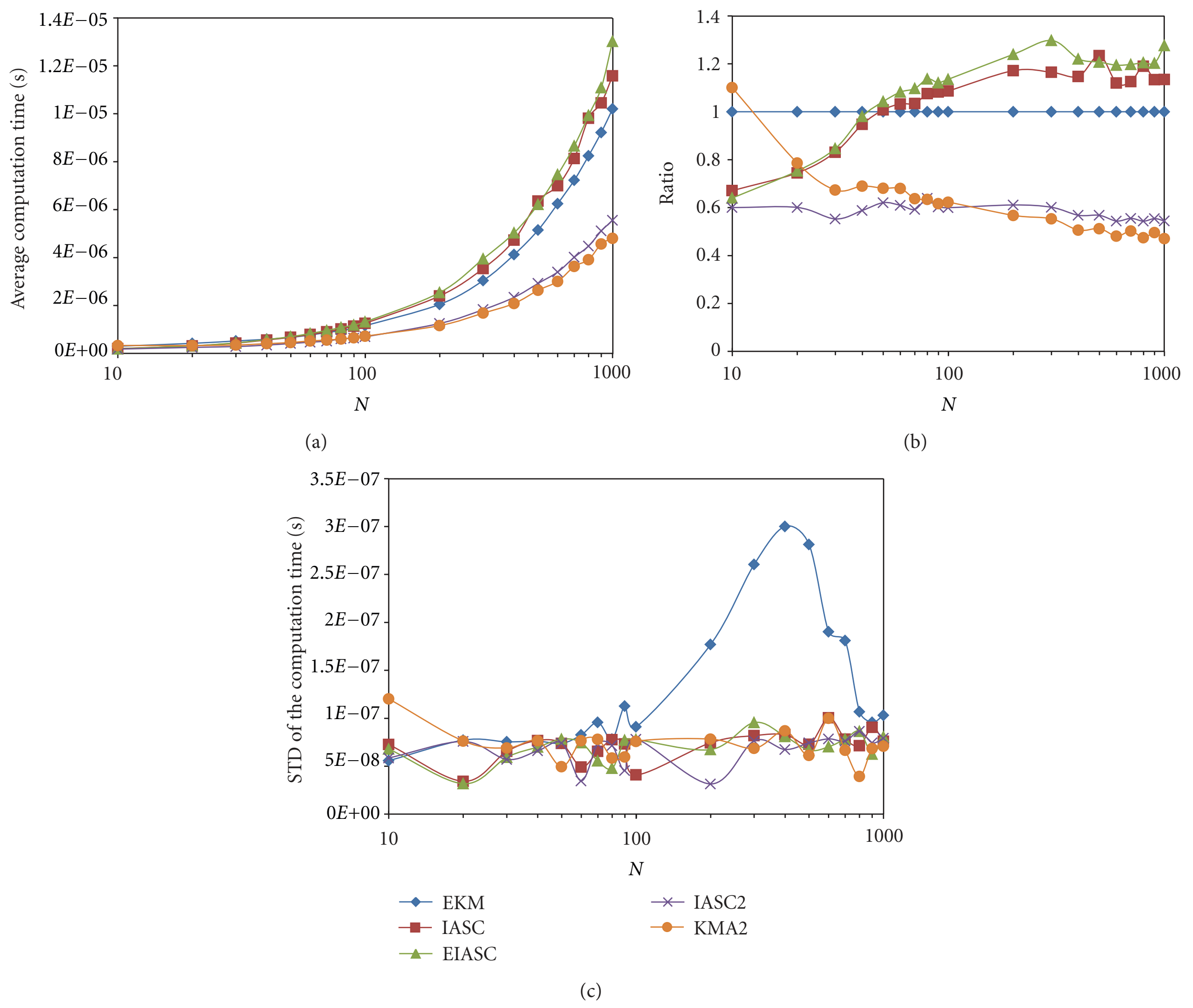


Figure 4: Centered random FOU results (compiled language implementation of the algorithms). (a) Average computation time, (b) ratio with respect to the EKMA, and (c) standard deviation of the computation time.

number of iterations regarding each FOU case are presented. A comparison of the IASC-type algorithms and also of the KM-type algorithms are provided for discussion.

In the case of a uniform random FOU (Figure 7), IASC2 reduced considerably the number of iterations when compared to the IASC and EIASC algorithms. The percentage reduction was about 75% for $N = 1000$ with respect to EIASC. On the other hand, no improvement was observed for KMA2 over the EKMA in this case; however, the computation time was smaller for KMA2 as presented in previous subsection.

In the case of a centered random FOU, the IASC2 provided a significant reduction in the number of iterations compared to the IASC and EIASC algorithms, as shown in Figure 8. The percentage reduction was about 88% for $N = 1000$ with respect to EIASC. Regarding the KM-type algorithms, the KMA2 reduced the number of iterations with respect to the EKMA for N smaller than 50 points. However, the performance of KMA2 is quite similar to that of EKMA for N greater than 100 points.

Regarding the results from the left-sided random FOU case (Figure 9), an interesting reduction in the number of iterations was achieved by IASC2 compared to the IASC and EIASC algorithms. The reduction percentage is around 90% for $N = 1000$ with respect to EIASC. KMA2 displayed a reduction percentage ranging from 30% for $N = 1000$ to 83% for $N = 10$ with respect to EKMA.

The IASC2 algorithm outperformed the other two IASC-type algorithms for a right-sided random FOU (Figure 10), as observed in the previous cases. Considering the KM-type algorithms, KMA2 provided the largest reduction in the number of iterations with respect to EKMA for $N = 10$ points. The reduction percentage was about 80% in this case. KMA2 also outperformed EKMA for greater resolutions.

4.2. Discussion. We believe that the four experiments reported in this section provide enough data to claim that IASC2 and KMA2 are faster than existing iterative algorithms like EKMA, IASC, and EIASC as long as the algorithms are

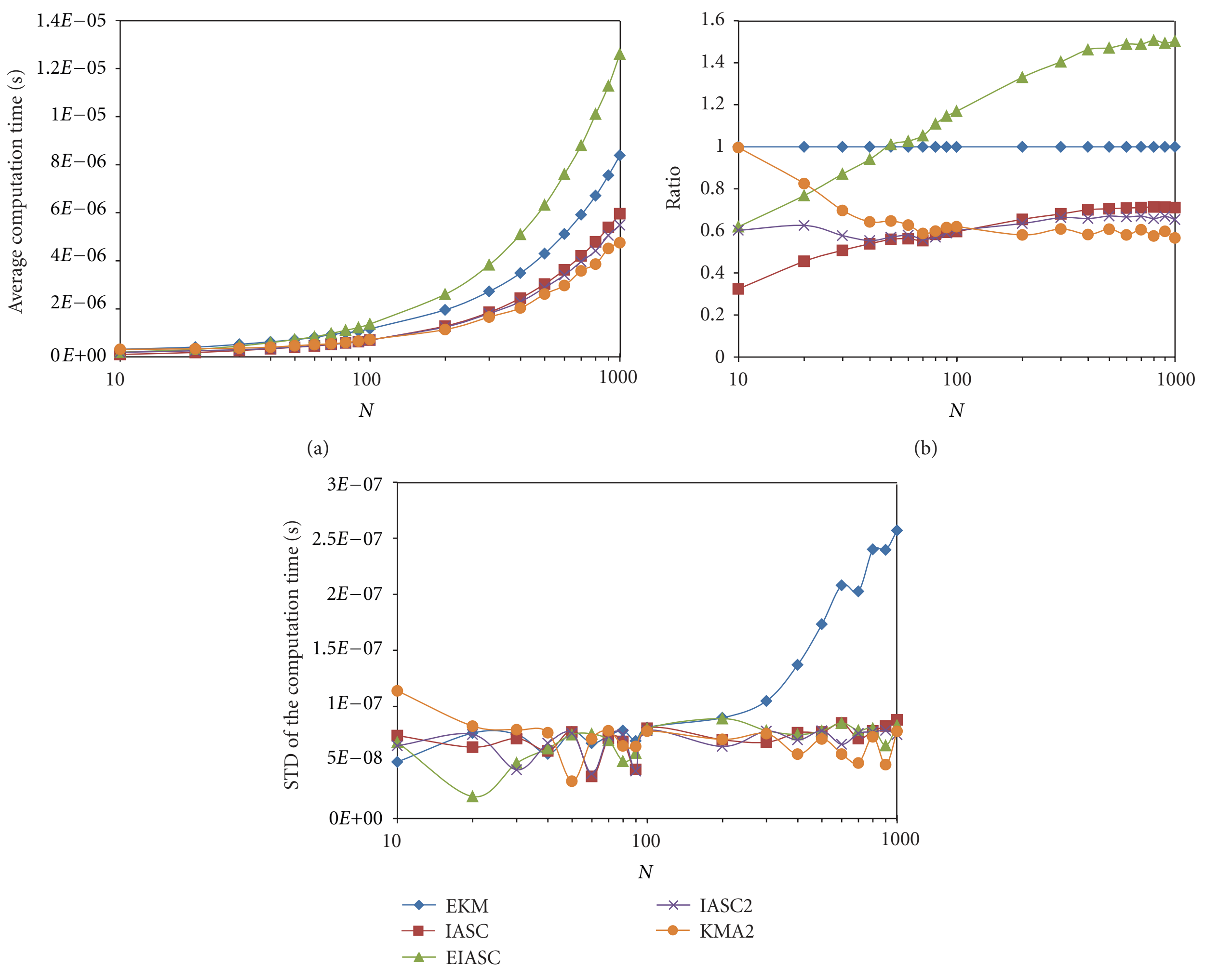


Figure 5: Left-sided random FOU results (compiled language implementation of the algorithms). (a) Average computation time, (b) ratio with respect to the EKMA, and (c) standard deviation of the computation time.

coded as compiled language programs. The experiments are not only limited to one type of FOU, instead, typical FOU cases that are expected to be obtained at the aggregated output of an IT2-FLS were used. IASC2 may be regarded as the best solution for computing type-reduction in an IT2-FLS when the discretization of the output universe of discourse is smaller than 100 points. On the other hand, KMA2 might be the fastest solution when discretization is bigger than 100 points. Thus we believe that IASC2 should be used in real-time applications while KMA2 should be reserved for intensive applications.

Initialization based on the uncertainty bounds proved to be more effective than previous initialization schemes. This is evident from the reduction achieved in the number of iterations when finding y_R (similar results were obtained in the case of y_L). We argue that the inner-bound set is an initial point that adapts itself to the shape of the FOU. This is clearly different from the fixed initial points that pertain to EIASC and EKMA, where the uncertainty involved in the IT2-FS is not considered. In addition, a simple analysis of the uncertainty bounds in [7] shows that the inner-bound set largely contributes to the computation of the outer-bound set, which is used to estimate the interval centroid.

5. Computational Comparison with Existing Iterative Algorithms: Implementation Based on Interpreted Language

5.1. Simulation Setup. The same four FOU cases of the previous section were used to characterize the algorithms implemented as interpreted programs. Since an interpreted implementation of an algorithm is slower than a compiled implementation, the experimental setup was modified. In this case, 2000 Monte Carlo trials were performed for each FOU case; however only 100 simulations were computed for each trial, so the time for executing a type-reduction is the measured time divided by 100.

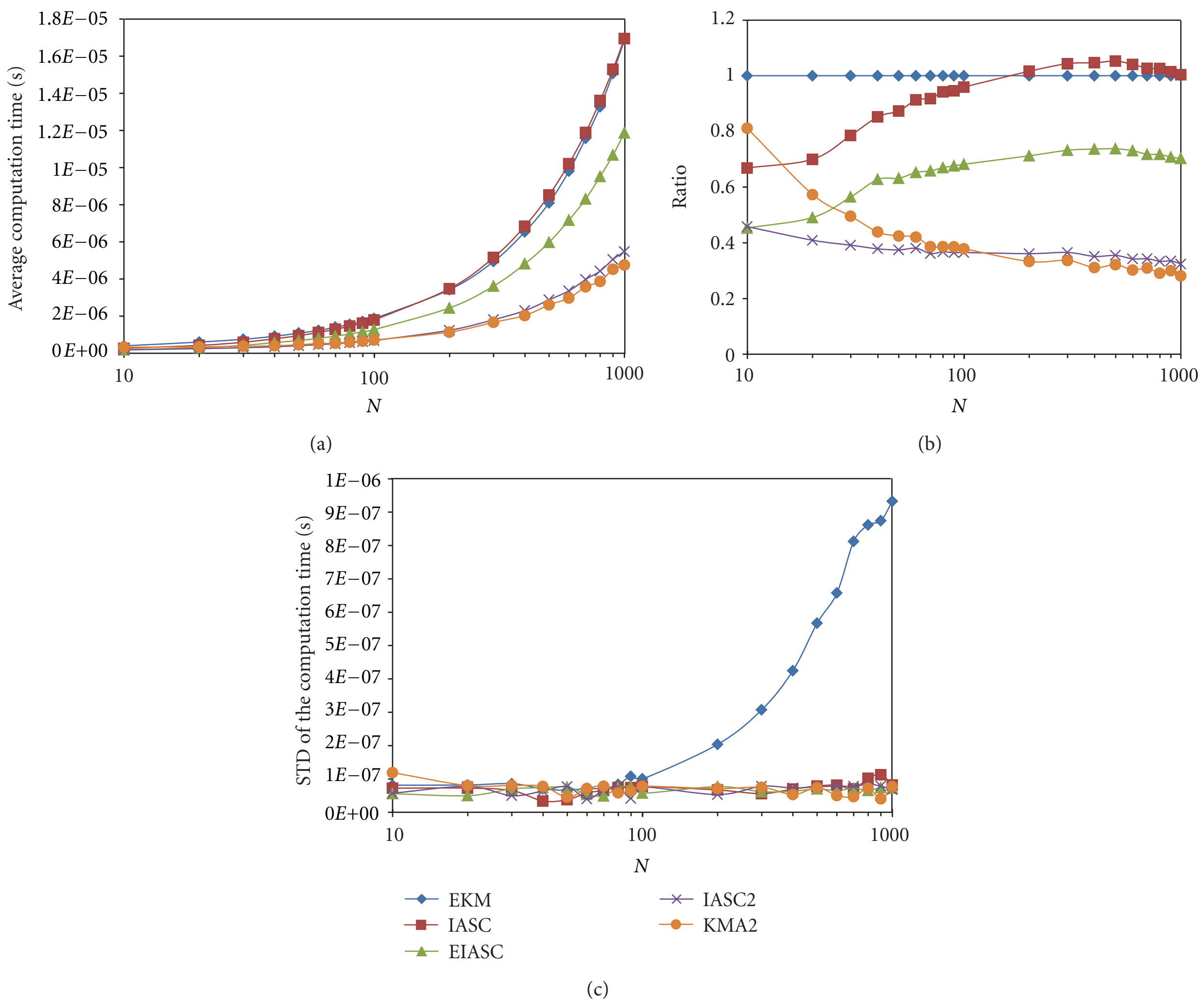


Figure 6: Right-sided random FOU results (compiled language implementation of the algorithms). (a) Average computation time, (b) ratio with respect to the EKMA, and (c) standard deviation of the computation time.

The algorithms were implemented as SCILAB programs. The experiments were carried out over SCILAB 5.3.1 numeric software running over a platform equipped with an AMD Athlon(tm) 64 × 2 Dual Core Processor 4000 + 2.10 GHz, 2 GB RAM, and Windows XP operating system.

5.2. Results: Execution Time. The results of computation time for uniform random FOUs are presented in Figure 11. KMA2 achieved the best performance among all algorithms with a PCTR of around 60%, and the IASC2 PCTR was close to 50%. The standard deviation of the computation time for the IASC2 and KMA2 was always smaller than that of EKMA. Thus, considering an implementation using an interpreted language, KMA2 should be the most appropriated solution for type-reduction in an IT2-FLS.

Figure 12 is similar to Figure 13. These figures present the results for centered random FOUs and left-sided random FOUs, respectively. KMA2 was the fastest algorithm, followed by IASC2, EIASC, IASC, and EKMA as the slowest. The IASC2 PCTR was almost similar to the KMA2 PCTR for N greater than 20, which was about 60%. When $N = 10$, the IASC2 PCTR was about 46%. The standard deviation of computation time was similar for all algorithms for N smaller than 300 although it was significantly different with respect to the EKMA for N greater than 300 points.

The results with right-sided random FOUs in Figure 14 show that KMA2 was the fastest algorithm in all the cases, followed by IASC2 with a similar performance. The PCTR for N greater than 30 was about 76% with KMA2, while, with IASC2, it was about 74% over the EKMA computation time. For the other cases the reduction was around 70% and 64%, respectively. The standard deviation of the computation time for all algorithms was often smaller than that of EKMA.

6. Conclusions

Two new algorithms for computing the centroid of an IT2-FS have been presented, namely, IASC2 and KMA2. IASC2 follows the conceptual line of algorithms like IASC [14]

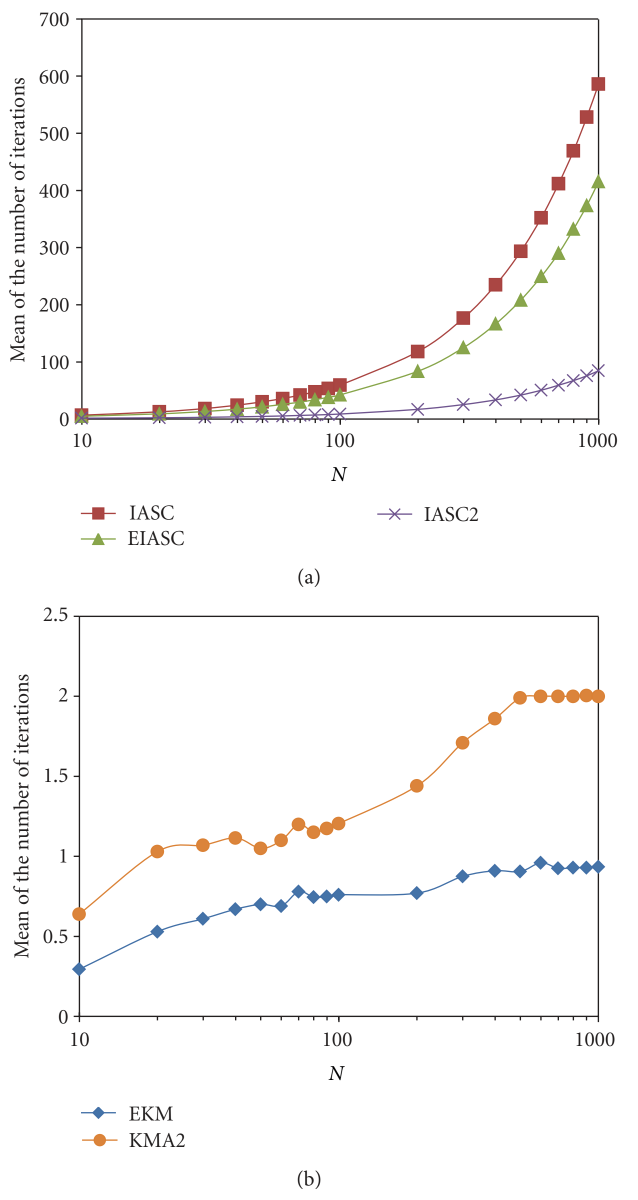


FIGURE 7: Uniform random FOU, iterations, (a) IASC-type algorithms and (b) KM-type algorithms.

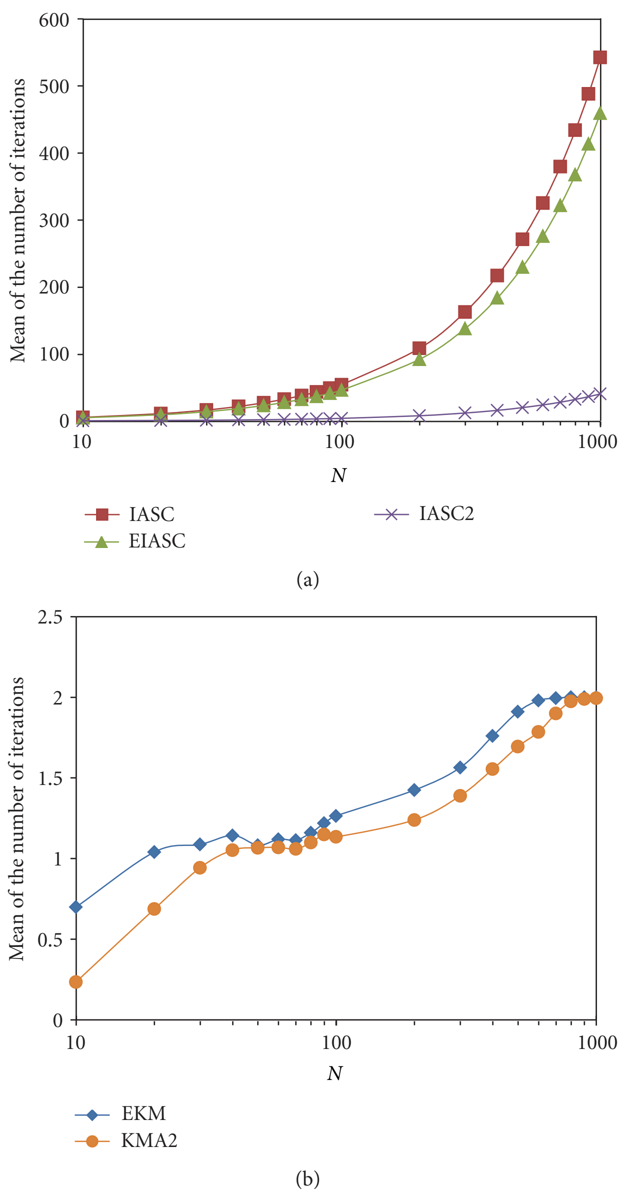
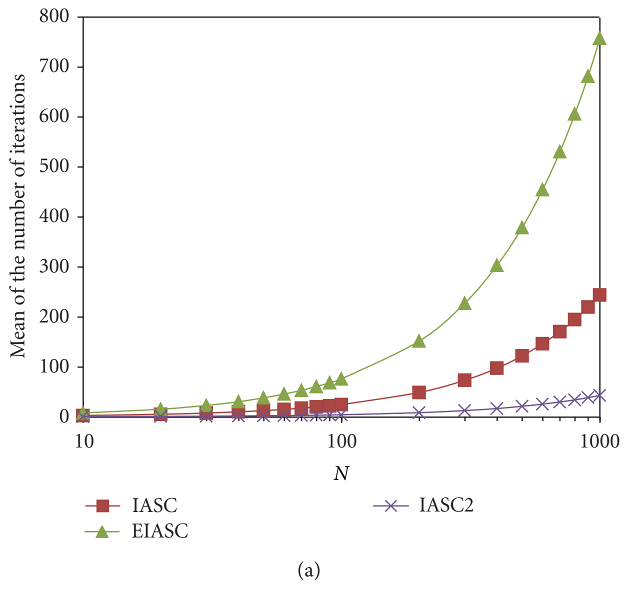


FIGURE 8: Centered random FOU, iterations: (a) IASC-type algorithms and (b) KM-type algorithms.

and EIASC [18]. KMA2 is inspired in the KM and EKM algorithms [16]. The new algorithms include an initialization that is based on the concept of inner-bound set [7], which reduces the number of iterations to find the centroid of an IT2-FS. Moreover, precomputation is considered in order to reduce the computational burden of each iteration.

These features have led to motivating results over different kinds of FOUs. The new algorithms achieved interesting computation time reductions with respect to the EKM algorithm. In the case of a compiled-language-based implementation, IASC2 exhibited the best reduction for resolutions smaller than 100 points between 40% and 70%. On the other hand, KMA2 exhibited the best performance for resolutions greater than 100 points, resulting in a reduction ranging from 45% to 60%. In the case of an interpreted-language-based implementation, KMA2 was the fastest algorithm, exhibiting a computation time reduction of around 60%.

The implementation of IT2-FLSs can take advantage of the algorithms presented in this work. Since IASC2 reported the best results for low resolutions, we consider that this algorithm should be applied in real-time applications of IT2-FLSs. Conversely, KMA2 can be considered for intensive applications that demand high resolutions. The following step in this paper will be focused to compare the new algorithms IASC2 and KMA2 with the EODS algorithm, since it demonstrated to outperform EIASC in low-resolution applications [25].



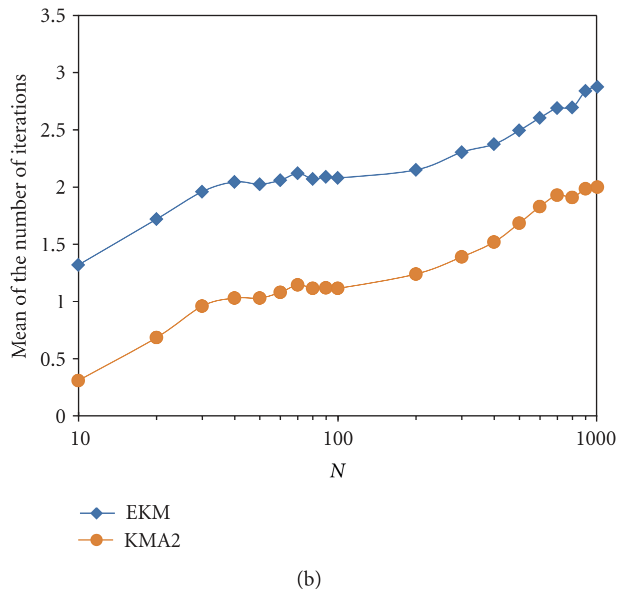
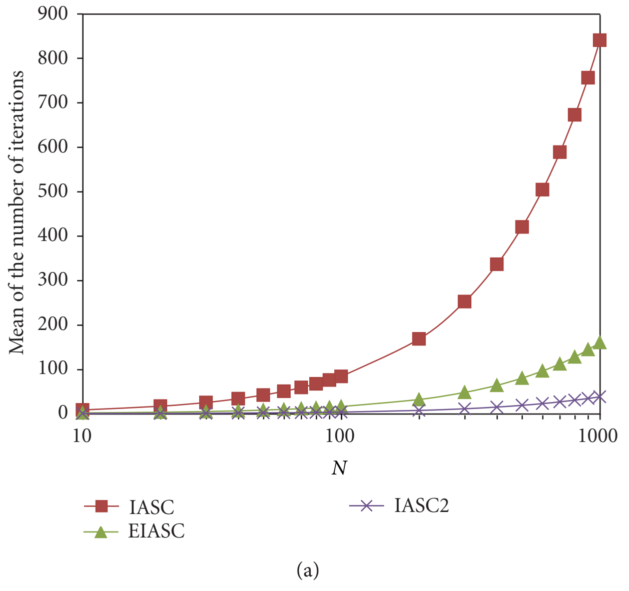


Figure 9: Left-sided random FOU, iterations, (a) IASC-type algorithms and (b) KM-type algorithms.



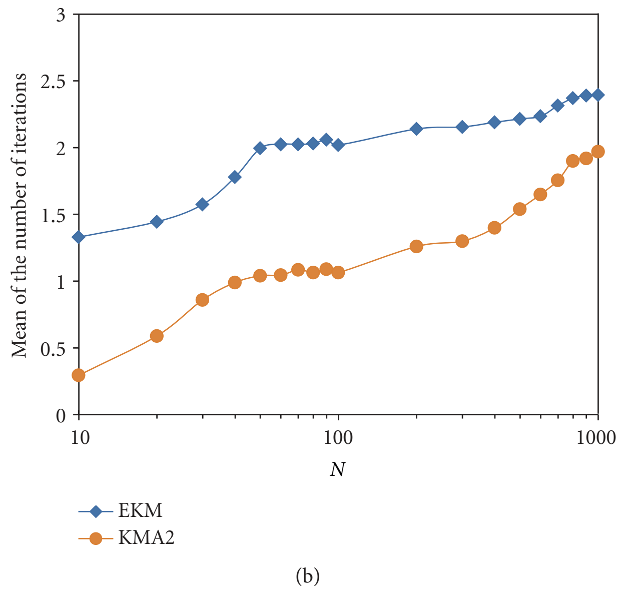


Figure 10: Right-sided random FOU, iterations, (a) IASC-type algorithms and (b) KM-type algorithms.

Appendix

The proof of the algorithm IASC-2 for computing y_L is divided in four parts.

(a) It will be stated that Table 1 (21) fixes a valid initialization point.

(b) It will be demonstrated that Table 1 (22)–(24) computes the left centroid function with $k = L_i$.

(c) It will be demonstrated that recursion of step 4 computes the left centroid function with initial point in $k = L_i$, and each decrement in k leads to a decrement in $y_l(k)$.

(d) The validity of the stop condition will be stated.

Proof. (a) From Table 1 (15)–(21), it follows that

$$y_A = \frac{T[N]}{V[N]} = \frac{\sum_{i=1}^{N} x_i \overline{f_i}}{\sum_{i=1}^{N} \overline{f_i}}, \tag{A.1}$$

$$y_B = \frac{Z[N]}{H[N]} = \frac{\sum_{i=1}^{N} x_i \underline{f_i}}{\sum_{i=1}^{N} \underline{f_i}}, \tag{A.2}$$

$$y_{\min} = \min\left(\frac{\sum_{i=1}^{N} x_i \overline{f_i}}{\sum_{i=1}^{N} \overline{f_i}}, \frac{\sum_{i=1}^{N} x_i \underline{f_i}}{\sum_{i=1}^{N} \underline{f_i}}\right), \tag{A.3}$$

$$y_{\min} = \overline{y_l}. \tag{A.4}$$

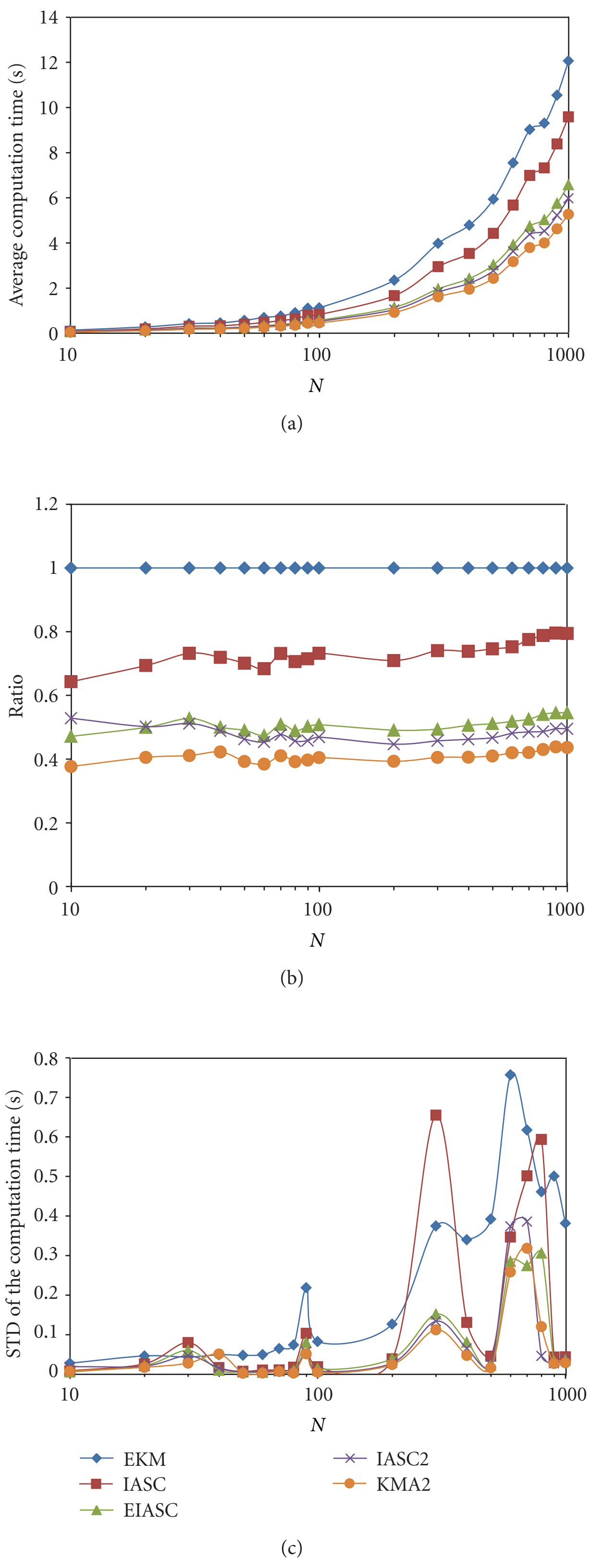


Figure 11: Uniform random FOU results (interpreted language implementation of the algorithms). (a) Average computation time, (b) ratio with respect to the EKMA, and (c) standard deviation of computation time.

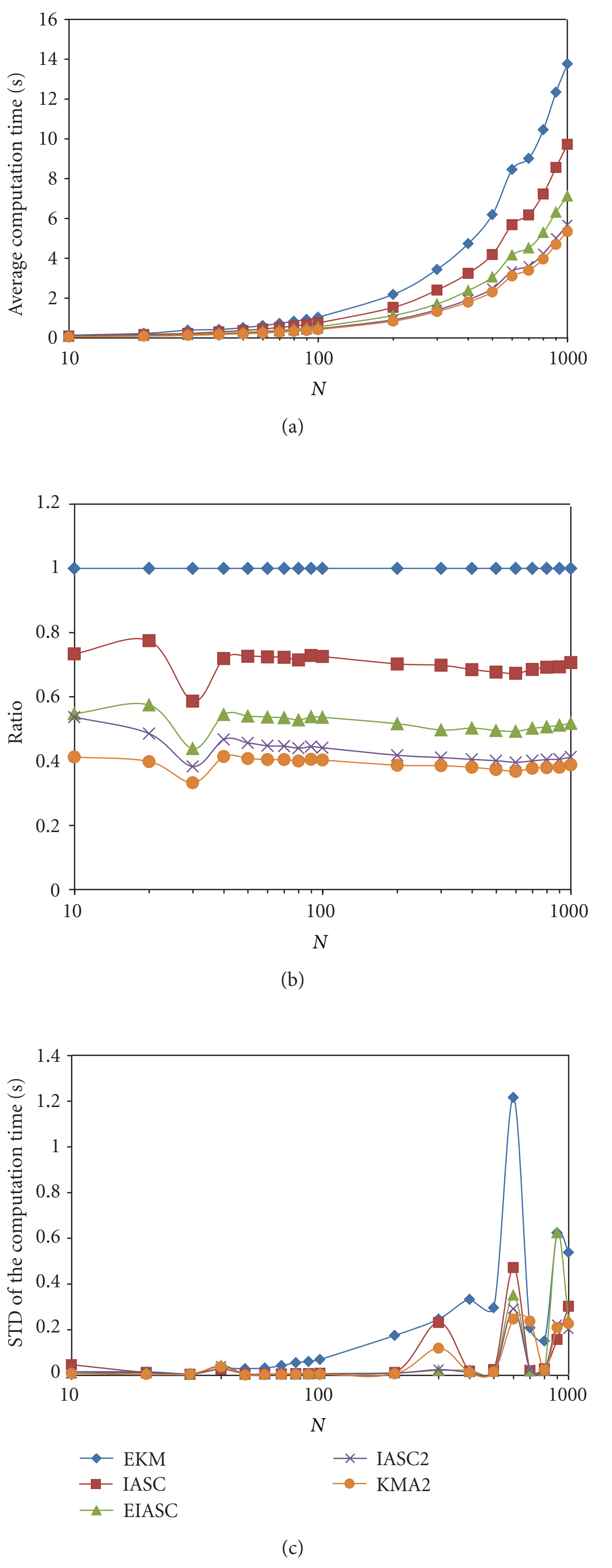


Figure 12: Centered random FOU results (interpreted language implementation of the algorithms). (a) Average computation time, (b) ratio with respect to the EKMA, and (c) standard deviation of computation time.

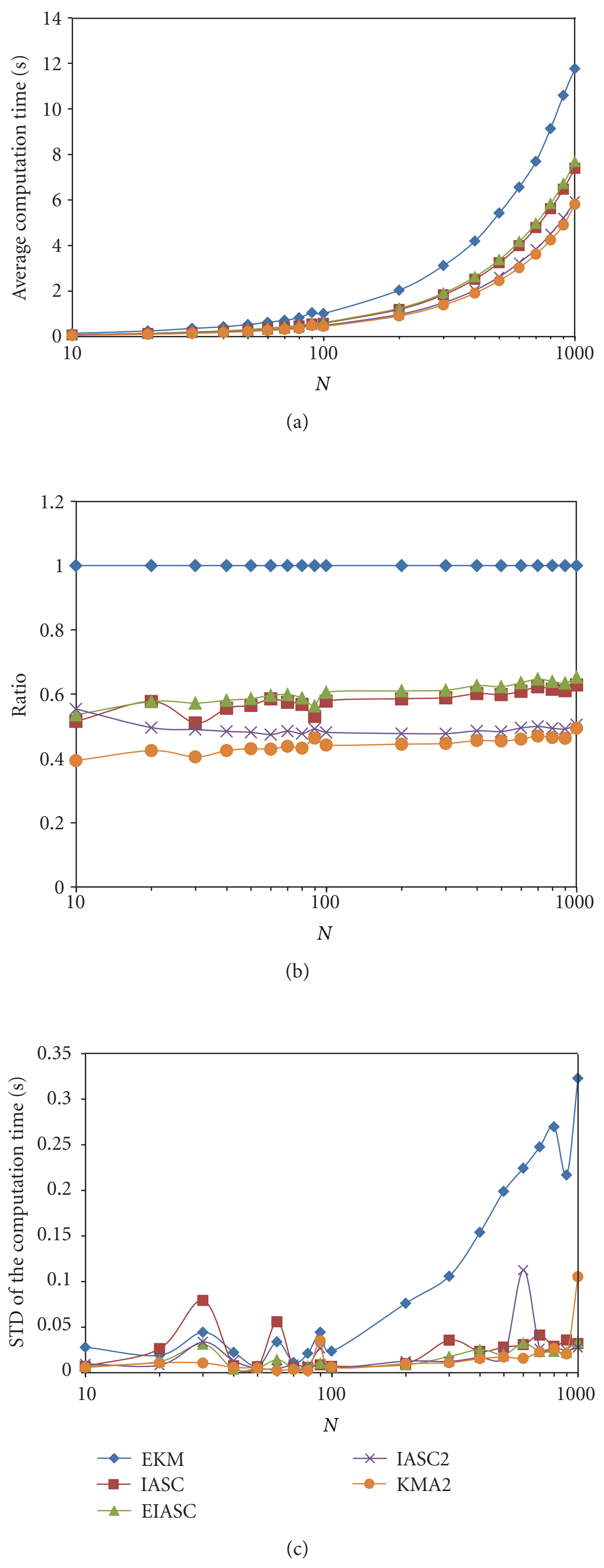


Figure 13: Left-sided random FOU results (interpreted language implementation of the algorithms). (a) Average computation time, (b) ratio with respect to the EKMA, and (c) standard deviation of computation time.

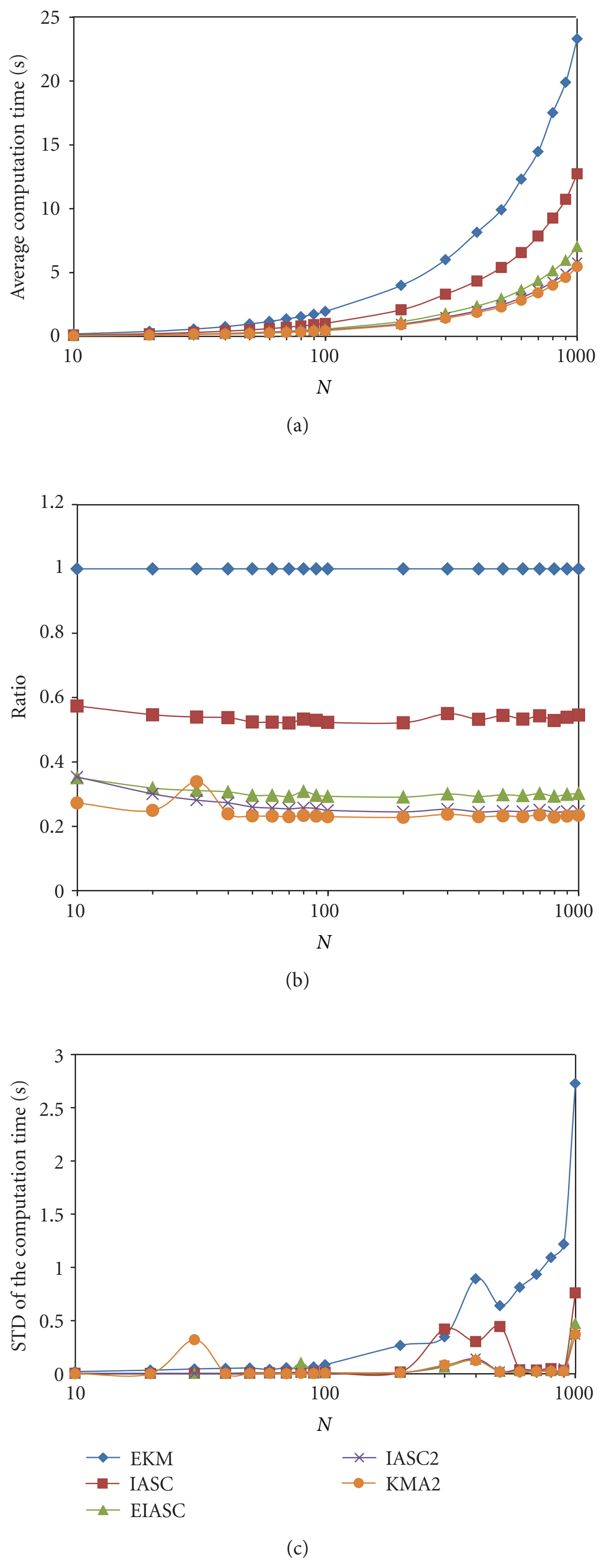


Figure 14: Right-sided random FOU results (interpreted language implementation of the algorithms). (a) Average computation time, (b) ratio with respect to the EKMA, and (c) standard deviation of computation time.

It is clear that $y_{\min}$ in Table 1 (21) corresponds to the minimum of the inner-bound set [7], so according to (6) $y_{\min} \geq y_L$. Thus Table 1 (21) provides an initialization point that is always greater than or equal to the minimum extreme point y_L. In addition, It is possible to find an integer L_i so that $x_{L_i} \leq y_{\min} < x_{L_i+1}$.

(b) From Table 1 (22), it follows that

$$\begin{aligned} D_l(L_i) &= T[L_i] + (Z[N] - Z[L_i]), \\ D_{L_i} = D_l(L_i) &= \sum_{i=1}^{L_i} x_i \overline{f_i} + \left(\sum_{i=1}^{N} x_i \underline{f_i} - \sum_{i=1}^{L_i} x_i \underline{f_i} \right), \\ D_{L_i} = D_l(L_i) &= \sum_{i=1}^{L_i} x_i \overline{f_i} + \sum_{i=L_i+1}^{N} x_i \underline{f_i}. \end{aligned} \tag{A.5}$$

From Table 1 (23), it follows that

$$\begin{aligned} P_l(L_i) &= V[L_i] + (Y[N] - Y[L_i]), \\ P_{L_i} = P_l(L_i) &= \sum_{i=1}^{L_i} \overline{f_i} + \left(\sum_{i=1}^{N} \underline{f_i} - \sum_{i=1}^{L_i} \underline{f_i} \right), \\ P_{L_i} = P_l(L_i) &= \sum_{i=1}^{L_i} \overline{f_i} + \sum_{i=L_i+1}^{N} \underline{f_i}. \end{aligned} \tag{A.6}$$

Therefore from Table 1 (24)

$$\begin{aligned} y_{\min} &= \frac{D_{L_i}}{P_{L_i}} = \frac{\sum_{i=1}^{L_i} x_i \overline{f_i} + \left(\sum_{i=1}^{N} x_i \underline{f_i} - \sum_{i=1}^{L_i} x_i \underline{f_i} \right)}{\sum_{i=1}^{L_i} \overline{f_i} + \left(\sum_{i=1}^{N} \underline{f_i} - \sum_{i=1}^{L_i} \underline{f_i} \right)}, \\ y_{\min} &= \frac{\sum_{i=1}^{L_i} x_i \overline{f_i} + \sum_{i=1}^{L_i} x_i \underline{f_i} + \sum_{i=L_i+1}^{N} x_i \underline{f_i} - \sum_{i=1}^{L_i} x_i \underline{f_i}}{\sum_{i=1}^{L_i} \overline{f_i} + \sum_{i=1}^{L_i} \underline{f_i} + \sum_{i=L_i+1}^{N} \underline{f_i} - \sum_{i=1}^{L_i} \underline{f_i}}, \\ y_{\min} &= \frac{\sum_{i=1}^{L_i} x_i \overline{f_i} + \sum_{i=L_i+1}^{N} x_i \underline{f_i}}{\sum_{i=1}^{L_i} \overline{f_i} + \sum_{i=L_i+1}^{N} \underline{f_i}}. \end{aligned} \tag{A.7}$$

(c) First we show that recursion computes the left centroid function with initial point L_i and decreasing k. Considering

$$y_l(k) = \frac{\sum_{i=1}^{k} x_i \overline{f_i} + \sum_{i=k+1}^{N} x_i \underline{f_i}}{\sum_{i=1}^{k} \overline{f_i} + \sum_{i=k+1}^{N} \underline{f_i}}. \tag{A.8}$$

Expanding it without altering the proportion

$$\begin{aligned} y_l(k) &= \left(\sum_{i=1}^{k} x_i \overline{f_i} + \sum_{i=k+1}^{N} x_i \underline{f_i} + \sum_{i=k+1}^{L_i} x_i \overline{f_i} - \sum_{i=k+1}^{L_i} x_i \overline{f_i} + \sum_{i=k+1}^{L_i} x_i \underline{f_i} - \sum_{i=k+1}^{L_i} x_i \underline{f_i} \right) \\ &\quad \times \left(\sum_{i=1}^{k} \overline{f_i} + \sum_{i=k+1}^{N} \underline{f_i} + \sum_{i=k+1}^{L_i} \overline{f_i} - \sum_{i=k+1}^{L_i} \overline{f_{ii}} + \sum_{i=k+1}^{L_i} \underline{f_i} - \sum_{i=k+1}^{L_i} \underline{f_i} \right)^{-1}, \\ y_l(k) &= \frac{\sum_{i=1}^{L_i} x_i \overline{f_i} + \sum_{i=L_i+1}^{N} x_i \underline{f_i} + \sum_{i=k+1}^{L_i} x_i \underline{f_i} - \sum_{i=k+1}^{L_i} x_i \overline{f_i}}{\sum_{i=1}^{L_i} \overline{f_i} + \sum_{i=L_i+1}^{N} \underline{f_i} + \sum_{i=k+1}^{L_i} \underline{f_i} - \sum_{i=k+1}^{L_i} \overline{f_i}}, \\ y_l(k) &= \frac{\sum_{i=1}^{L_i} x_i \overline{f_i} + \sum_{i=L_i+1}^{N} x_i \underline{f_i} - \sum_{i=k+1}^{L_i} x_i \left(\overline{f_i} - \underline{f_i} \right)}{\sum_{i=1}^{L_i} \overline{f_i} + \sum_{i=L_i+1}^{N} \underline{f_i} - \sum_{i=k+1}^{L_i} \left(\overline{f_i} - \underline{f_i} \right)}, \\ y_l(k) &= \frac{D_{L_i} - \sum_{i=k+1}^{L_i} x_i \left(\overline{f_i} - \underline{f_i} \right)}{P_{L_i} - \sum_{i=k+1}^{L_i} \left(\overline{f_i} - \underline{f_i} \right)}, \end{aligned} \tag{A.9}$$

$$y_l(k) = \frac{D_{L_i} - \sum_{i=k+1}^{L_i} x_i \left(\overline{f_i} - \underline{f_i} \right)}{P_{L_i} - \sum_{i=k+1}^{L_i} \left(\overline{f_i} - \underline{f_i} \right)}. \tag{A.10}$$

It can be observed that (A.10) is a proportion composed by initial values D_{L_i} and P_{L_i} and two terms that can be computed by recursion $D(k)$ and $P(k)$, so

$$\begin{aligned} y_l(k) &= \frac{D_{L_i} - D(k)}{P_{L_i} - P(k)}, \\ D(k) &= \sum_{i=k+1}^{L_i} x_i \left(\overline{f_i} - \underline{f_i} \right), \\ P(k) &= \sum_{i=k+1}^{L_i} \left(\overline{f_i} - \underline{f_i} \right). \end{aligned} \tag{A.11}$$

Sums are computed by recursion as

$$D_{k-1} = D_k - x_k \left(\overline{f_k} - \underline{f_k} \right), \tag{A.12}$$

$$P_{k-1} = P_k - \left(\overline{f_k} - \underline{f_k} \right), \tag{A.13}$$

$$y_l(k-1) = \frac{D_{k-1}}{P_{k-1}} = \frac{D_k - x_k \left(\overline{f_k} - \underline{f_k} \right)}{P_k - \left(\overline{f_k} - \underline{f_k} \right)}. \tag{A.14}$$

Since $L \leq L_i$ given that $y_l \leq y_{\min}$ in Table 1 (21), iterations decrease k from L_i according to (A.10).

It has been demonstrated that (A.14) is the same (A.8) computed recursively from initial point L_i. So, by performing a decrement of k in step 5, a decrement is introduced in f_i; that is,

$$f \;= (f_1, \dots, f_N) = \left(\overline{f_1}, \dots \overline{f_{k-1}}, \overline{f_k}, \underline{f_{k+1}}, \dots \underline{f_N}\right),$$
$$f = (f_1, \dots, f_N) = \left(\overline{f_1}, \dots \overline{f_{k-1}}, \underline{f_k}, \underline{f_{k+1}}, \dots \underline{f_N}\right). \quad \text{(A.15)}$$

In [5], it is demonstrated that

$$\text{if } x_k < y(f_1, \dots, f_N), \text{then } y(f_1, \dots, f_N) \text{ increases when } f_k \text{ decreases.} \quad \text{(A.16)}$$

$$\text{If } x_k > y(f_1, \dots, f_N), \text{then } y(f_1, \dots, f_N) \text{ increases when } f_k \text{ increases.} \quad \text{(A.17)}$$

Note that k is greater than L which implies that $x_k > y_L$; thus from the centroid function properties, it is guaranteed that $x_k > y_l(k)$, satisfying (A.17). Therefore, it can be concluded that a decrement in f_k as (A.15) will induce a decrement in $y_l(k)$.

(d) The stop condition is stated from the properties of the centroid function. Since the centroid function has a global minimum in L, once recursion has reached $k = L$, $y_l(k)$ will increase in the next decrement of k because (A.17) is no longer valid. So by detecting the increment in $y_l(k)$, it can be said that the minimum has been found in the previous iteration. □

Conflict of Interests

The authors hereby declare that they do not have any direct financial relation with the commercial identities mentioned in this paper.

References

[1] J. M. Mendel, "Advances in type-2 fuzzy sets and systems," *Information Sciences*, vol. 177, no. 1, pp. 84–110, 2007.

[2] R. John and S. Coupland, "Type-2 fuzzy logic: a historical view," *IEEE Computational Intelligence Magazine*, vol. 2, no. 1, pp. 57–62, 2007.

[3] J. M. Mendel, "Type-2 fuzzy sets and systems: an overview," *IEEE Computational Intelligence Magazine*, vol. 2, no. 1, pp. 20–29, 2007.

[4] S. Coupland and R. John, "Geometric type-1 and type-2 fuzzy logic systems," *IEEE Transactions on Fuzzy Systems*, vol. 15, no. 1, pp. 3–15, 2007.

[5] J. Mendel, *Uncertain Rule-Based Fuzzy Logic Systems: Introduction and New Directions*, Prentice Hall, Upper Saddle River, NJ, USA, 2001.

[6] J. M. Mendel, "On a 50% savings in the computation of the centroid of a symmetrical interval type-2 fuzzy set," *Information Sciences*, vol. 172, no. 3-4, pp. 417–430, 2005.

[7] H. Wu and J. M. Mendel, "Uncertainty bounds and their use in the design of interval type-2 fuzzy logic systems," *IEEE Transactions on Fuzzy Systems*, vol. 10, no. 5, pp. 622–639, 2002.

[8] D. Wu and W. W. Tan, "Computationally efficient type-reduction strategies for a type-2 fuzzy logic controller," in *Proceedings of the IEEE International Conference on Fuzzy Systems*, pp. 353–358, May 2005.

[9] C. Y. Yeh, W. H. R. Jeng, and S. J. Lee, "An enhanced type-reduction algorithm for type-2 fuzzy sets," *IEEE Transactions on Fuzzy Systems*, vol. 19, no. 2, pp. 227–240, 2011.

[10] C. Wagner and H. Hagras, "Toward general type-2 fuzzy logic systems based on zSlices," *IEEE Transactions on Fuzzy Systems*, vol. 18, no. 4, pp. 637–660, 2010.

[11] J. M. Mendel, F. Liu, and D. Zhai, "α-Plane representation for type-2 fuzzy sets: theory and applications," *IEEE Transactions on Fuzzy Systems*, vol. 17, no. 5, pp. 1189–1207, 2009.

[12] S. Coupland and R. John, "An investigation into alternative methods for the defuzzification of an interval type-2 fuzzy set," in *Proceedings of the IEEE International Conference on Fuzzy Systems*, pp. 1425–1432, July 2006.

[13] J. M. Mendel and F. Liu, "Super-exponential convergence of the Karnik-Mendel algorithms for computing the centroid of an interval type-2 fuzzy set," *IEEE Transactions on Fuzzy Systems*, vol. 15, no. 2, pp. 309–320, 2007.

[14] K. Duran, H. Bernal, and M. Melgarejo, "Improved iterative algorithm for computing the generalized centroid of an interval type-2 fuzzy set," in *Proceedings of the Annual Meeting of the North American Fuzzy Information Processing Society (NAFIPS '08)*, New York, NY, USA, May 2008.

[15] K. Duran, H. Bernal, and M. Melgarejo, "A comparative study between two algorithms for computing the generalized centroid of an interval Type-2 Fuzzy Set," in *Proceedings of the IEEE International Conference on Fuzzy Systems*, pp. 954–959, Hong Kong, China, June 2008.

[16] D. Wu and J. M. Mendel, "Enhanced Karnik-Mendel algorithms," *IEEE Transactions on Fuzzy Systems*, vol. 17, no. 4, pp. 923–934, 2009.

[17] X. Liu, J. M. Mendel, and D. Wu, "Study on enhanced Karnik-Mendel algorithms: Initialization explanations and computation improvements," *Information Sciences*, vol. 184, no. 1, pp. 75–91, 2012.

[18] D. Wu and M. Nie, "Comparison and practical implementation of type-reduction algorithms for type-2 fuzzy sets and systems," in *Proceedings of the IEEE International Conference on Fuzzy Systems*, pp. 2131–2138, June 2011.

[19] M. Melgarejo, "A fast recursive method to compute the generalized centroid of an interval type-2 fuzzy set," in *Proceedings of the Annual Meeting of the North American Fuzzy Information Processing Society (NAFIPS '07)*, pp. 190–194, June 2007.

[20] R. Sepúlveda, O. Montiel, O. Castillo, and P. Melin, "Embedding a high speed interval type-2 fuzzy controller for a real plant into an FPGA," *Applied Soft Computing*, vol. 12, no. 3, pp. 988–998, 2012.

[21] R. Sepúlveda, O. Montiel, O. Castillo, and P. Melin, "Modelling and simulation of the defuzzification stage of a type-2 fuzzy controller using VHDL code," *Control and Intelligent Systems*, vol. 39, no. 1, pp. 33–40, 2011.

[22] O. Montiel, R. Sepúlveda, Y. Maldonado, and O. Castillo, "Design and simulation of the type-2 fuzzification stage: using active membership functions," *Studies in Computational Intelligence*, vol. 257, pp. 273–293, 2009.

[23] Y. Maldonado, O. Castillo, and P. Melin, "Optimization of membership functions for an incremental fuzzy PD control based on genetic algorithms," *Studies in Computational Intelligence*, vol. 318, pp. 195–211, 2010.

[24] C. Li, J. Yi, and D. Zhao, "A novel type-reduction method for interval type-2 fuzzy logic systems," in *Proceedings of the 5th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD '08)*, pp. 157–161, October 2008.

[25] D. Wu, "Approaches for reducing the computational cost of logic systems: overview and comparisons," *IEEE Transactions on Fuzzy Systems*, vol. 21, no. 1, pp. 80–90, 2013.

[26] H. Hu, Y. Wang, and Y. Cai, "Advantages of the enhanced opposite direction searching algorithm for computing the centroid of an interval type-2 fuzzy set," *Asian Journal of Control*, vol. 14, no. 6, pp. 1–9, 2012.

[27] J. M. Mendel, R. I. John, and F. Liu, "Interval type-2 fuzzy logic systems made simple," *IEEE Transactions on Fuzzy Systems*, vol. 14, no. 6, pp. 808–821, 2006.