2011 IEEE International Conference on Fuzzy Systems
June 27-30, 2011, Taipei, Taiwan

Comparison and Practical Implementation of Type-Reduction Algorithms for Type-2 Fuzzy Sets and Systems

Dongrui u
Industrial Artificial Intelligence Laboratory
E lobal Research, Niskayuna, N , USA
E-mail: wud ge.com

Maowen Nie
Department of Electrical and Computer Engineering
National University of Singapore, Singapore
E-mail: g0 00223 nus.edu.sg

***Abstract*—Type-reduction algorithms are very important for type-2 fuzzy sets and systems. The earliest one, and also the most popular one, is the Karnik-Mendel Algorithm, which is iterative and computationally intensive. In the last a few years researchers have proposed several other more efficient type-reduction algorithms. In this paper we also propose a new algorithm which improves over the latest results. Experiments show that it is the most efficient one to use in practice. Particularly, when the number of elements in type-reduction is smaller than 100, which is true in most practical type-reduction computations, our proposed algorithm can save over 50% computational cost over the Karnik-Mendel Algorithms. We also give the Matlab implementation of our most efficient algorithm in the Appendix. It includes preprocessing steps to eliminate numerical problems, and also improved testing criteria to prevent possible infinite loops. This program will be very helpful in promoting the popularity of type-2 fuzzy sets and systems.**

***Keywords*—Interval type-2 fuzzy sets, interval type-2 fuzzy logic systems, type-reduction, Karnik-Mendel Algorithm, Enhanced Karnik-Mendel Algorithm, computational cost**

I. Introduction

As introduced in 1 , the following optimization problem is frequently met in but not limited to) fuzzy logic theory:

Given

$$x_n \in X_n \equiv [\underline{x}_n, \overline{x}_n], \qquad n = 1, 2, \ldots, N \qquad 1)$$

$$w_n \in W_n \equiv [\underline{w}_n, \overline{w}_n], \qquad n = 1, 2, \ldots, N \qquad 2)$$

where

$$\underline{x}_n \leqslant \overline{x}_n, \qquad n = 1, 2, \ldots, N \qquad 3)$$

$$\underline{w}_n \leqslant \overline{w}_n, \qquad n = 1, 2, \ldots, N \qquad 4)$$

compute

$$Y = \frac{\sum_{n=1}^{N} X_n W_n}{\sum_{n=1}^{N} W_n} \equiv [y_l, y_r], \qquad)$$

where

$$y_l = \min_{\substack{\forall x_n \in [\underline{x}_n, \overline{x}_n] \\ \forall w_n \in [\underline{w}_n, \overline{w}_n]}} \frac{\sum_{n=1}^{N} x_n w_n}{\sum_{n=1}^{N} w_n} \qquad)$$

$$y_r = \max_{\substack{\forall x_n \in [\underline{x}_n, \overline{x}_n] \\ \forall w_n \in [\underline{w}_n, \overline{w}_n]}} \frac{\sum_{n=1}^{N} x_n w_n}{\sum_{n=1}^{N} w_n} \qquad —)$$

For instance, in computing the centroid of an interval type-2 fuzzy sets –IT2 FS) $\tilde{A}$ –2—4–, $\underline{x}_n = \overline{x}_n = x_n$ represent discretizations of the primary variable x, and interval $[\underline{w}_n, \overline{w}_n]$ is the membership grade of x_n. Y is the centroid of $\tilde{A}$. In center-of-sets type-reduction of IT2 fuzzy logic systems –FLSs) –4–, X_n represents the centroid of the consequent IT2 FS of the n^{th} rule, W_n is the firing level of that rule, and Y is the type-reduced set. In computing the linguistic weighted average ——, X_n and W_n are α-cuts of the upper membership functions –or lower membership functions) of the inputs signals and the weights, both of which are IT2 FSs, and Y is the corresponding α-cut on the upper membership function –or lower membership function) of the linguistic weighted average, which is also an IT2 FS.

Equation —) is usually solved by the iterative Karnik-Mendel algorithms –KMA) –2–, –4–, which is computationally intensive. In the last a few years several more efficient algorithms[1] have been proposed, e.g., the enhanced Karnik-Mendel algorithm –EKMA) –1–, —, a fast recursive method for computing the generalized centroid of IT2 FSs –1—, an iterative algorithm with stop condition[2] –IASC) for computing the generalized centroid[3] –1—, and, an enhanced algorithm for computing the generalized centroid of general type-2 FSs –19–. For simplicity these algorithms are all referred to as type-reduction algorithms in this paper.

In this paper we focus on more efficient type-reduction algorithms. Our contributions are: 1) – e propose an enhanced version of IASC and compare it with several other type-reduction algorithms– and, 2) – e give an Matlab implementation of our most efficient type-reduction algorithm, which can be used freely by other researchers.

The rest of this paper is organized as follows: Section II introduces KMA. Section III introduces four more efficient type-reduction algorithms, including EKMA, EKMA with new

[1]There are also many approximations to KMA –9—1—. As pointed out in –1—, these approximations are fundamentally different from the original KMA, and hence they are not considered in this paper.

[2]This algorithm is called IASCO in –1—.

[3]Though the two algorithms in –1—, –1— were initially proposed for computing the generalized centroid of IT2 FSs, they can also be used in type-reduction of IT2 FLSs.

978-1-4244-7317-5/11/$26.00 ©2011 IEEE

initialization –EKMANI), IASC, and enhanced IASC –EIASC). Section I– compares the performances of these five algorithms under three different scenarios. Finally, Section – draws conclusions. Matlab implementation of EIASC is given in the Appendix.

II. KARNIK-MENDEL AL– ORITHMS –KMA)

KMA consists of two parts, one for computing y_l in —) and the other for computing y_r in —). Define

$$f_l(k) = \frac{\sum_{n=1}^{k} \underline{x}_n \overline{w}_n + \sum_{n=k+1}^{N} \underline{x}_n \underline{w}_n}{\sum_{n=1}^{k} \overline{w}_n + \sum_{n=k+1}^{N} \underline{w}_n} \quad —)$$

$$f_r(k) = \frac{\sum_{n=1}^{k} \overline{x}_n \underline{w}_n + \sum_{n=k+1}^{N} \overline{x}_n \overline{w}_n}{\sum_{n=1}^{k} \underline{w}_n + \sum_{n=k+1}^{N} \overline{w}_n} \quad –9)$$

where $\{\underline{x}_n\}$ and $\{\overline{x}_n\}$ have been sorted in ascending order, respectively. Furthermore, in this paper it is assumed that $\{\underline{x}_n\}$ –$\{\overline{x}_n\}$) has no duplicate elements, which can be easily achieved by combining the weights for duplicate elements. Then, y_l in —) and y_r in —) can also be re-expressed as –4–:

$$y_l = \min_{k=1,\ldots,N-1} f_l(k) \equiv f_l(l) \quad –10)$$

$$= \frac{\sum_{n=1}^{l} \underline{x}_n \overline{w}_n + \sum_{n=l+1}^{N} \underline{x}_n \underline{w}_n}{\sum_{n=1}^{l} \overline{w}_n + \sum_{n=l+1}^{N} \underline{w}_n} \quad –11)$$

$$y_r = \max_{k=1,\ldots,N-1} f_r(k) \equiv f_r(r) \quad –12)$$

$$= \frac{\sum_{n=1}^{r} \overline{x}_n \underline{w}_n + \sum_{n=r+1}^{N} \overline{x}_n \overline{w}_n}{\sum_{n=1}^{r} \underline{w}_n + \sum_{n=r+1}^{N} \overline{w}_n} \quad –13)$$

where l and r are switch points satisfying

$$\underline{x}_l \leqslant y_l < \underline{x}_{l+1} \quad –14)$$

$$\overline{x}_r < y_r \leqslant \overline{x}_{r+1}. \quad –1–)$$

Note that in –4– these two inequalities are

$$\underline{x}_l \leqslant y_l \leqslant \underline{x}_{l+1} \quad –1–)$$

$$\overline{x}_r \leqslant y_r \leqslant \overline{x}_{r+1} \quad –1–)$$

because there $\{\underline{x}_n\}$ and $\{\overline{x}_n\}$ can have duplicate elements. – hen duplicate elements are combined, it follows that $\underline{x}_l < \underline{x}_{l+1}$ for $\forall l \in [1, N-1]$ and $\overline{x}_r < \overline{x}_{r+1}$ for $\forall r \in [1, N-1]$, and hence the two equalities in –1–) or –1–) cannot be satisfied simultaneously.

KMA for Computing y_l:

1) Sort $\underline{x}_n$ –$n = 1, 2, \ldots, N$) in increasing order and call the sorted $\underline{x}_n$ by the same name, but now $\underline{x}_1 < \underline{x}_2 < \cdots < \underline{x}_N$. Match the weights W_n with their respective $\underline{x}_n$ and renumber them so that their index corresponds to the renumbered $\underline{x}_n$.
2) Initialize w_n by setting
$$w_n = \frac{\underline{w}_n + \overline{w}_n}{2}, \qquad n = 1, 2, \ldots, N \quad –1–)$$
and then compute
$$y = \frac{\sum_{n=1}^{N} \underline{x}_n w_n}{\sum_{n=1}^{N} w_n}. \quad –19)$$
3) Find switch point l –$1 \leqslant l \leqslant N-1$) such that
$$\underline{x}_l < y \leqslant \underline{x}_{l+1} \quad –20)$$
4) Set
$$w_n = \begin{cases} \overline{w}_n, & n \leqslant l \\ \underline{w}_n, & n > l \end{cases} \quad –21)$$
and compute
$$y' = \frac{\sum_{n=1}^{N} \underline{x}_n w_n}{\sum_{n=1}^{N} w_n} \quad –22)$$
–) If $y' = y$, stop and set $y_l = y$–otherwise, set $y = y'$ and go to Step 3.

KMA for Computing y_r:

1) Sort $\overline{x}_n$ –$n = 1, 2, \ldots, N$) in increasing order and call the sorted $\overline{x}_n$ by the same name, but now $\overline{x}_1 < \overline{x}_2 < \cdots < \overline{x}_N$. Match the weights W_n with their respective $\overline{x}_n$ and renumber them so that their index corresponds to the renumbered $\overline{x}_n$.
2) Initialize w_n by setting
$$w_n = \frac{\underline{w}_n + \overline{w}_n}{2}, \qquad n = 1, 2, \ldots, N \quad –23)$$
and then compute
$$y = \frac{\sum_{n=1}^{N} \overline{x}_n w_n}{\sum_{n=1}^{N} w_n}. \quad –24)$$
3) Find switch point r –$1 \leqslant r \leqslant N-1$) such that
$$\overline{x}_r < y \leqslant \overline{x}_{r+1} \quad –2–)$$
4) Set
$$w_n = \begin{cases} \underline{w}_n, & n \leqslant r \\ \overline{w}_n, & n > r \end{cases} \quad –2–)$$
and compute
$$y' = \frac{\sum_{n=1}^{N} \overline{x}_n w_n}{\sum_{n=1}^{N} w_n} \quad –2–)$$
–) If $y' = y$, stop and set $y_r = y$–otherwise, set $y = y'$ and go to Step –3).

The main idea of KMA is to find the switch points for y_l and y_r. Take y_l as an example. y_l is the minimum of Y. Since $\underline{x}_n$ increases from the left to the right along the horizontal axis of Fig. 1–a), we should choose a large w_n for $\underline{x}_n$ on the left and a small w_n for $\underline{x}_n$ on the right. KMA for y_l finds the switch point l. For $n \leqslant l$, $\overline{w}_n$ is used to calculate y_l–for $n > l$, $\underline{w}_n$ is used. This ensures y_l is the minimum.

Note that the above KMA works well in most practical cases–however, sometimes due to roundoff error y' can never be equal to y and hence KMA runs into infinite loops. Special attentions need to be paid to these cases, as we have done in the Matlab program in the Appendix.

III. MORE EFFICIENT T– PE-REDUCTION AL– ORITHMS

Three existing more efficient type-reduction algorithms are introduced in this section, and a new algorithm is also proposed.

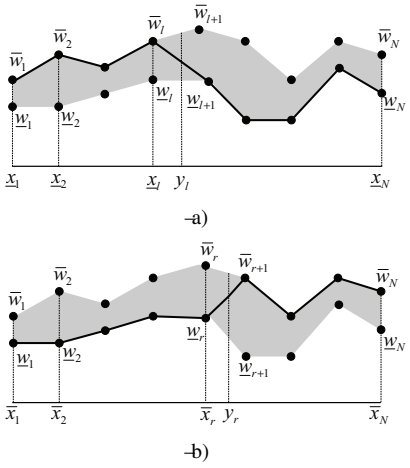


Fig. 1. Illustration of the switch points in computing $_l$ and $_r$. –a) Computing $_l$: switch from $\overline{w}_l$ to $\underline{w}_{l+1}$–b) Computing $_r$: switch from $\underline{w}_r$ to $\overline{w}_{r+1}$.

A. Enhanced Karnik-Mendel Algorithms (EKMA)

EKMA –1– has three improvements over KMA. First, a better initialization is used to reduce the number of iterations. Then, the termination condition of the iterations is changed to remove one unnecessary iteration. Finally, a subtle computing technique is used to reduce the computational cost of each iteration.

EKMA for computing y_l:

1) The same as Step –1) of KMA for computing y_l.
2) Set $l = [N/2.4]$ –the nearest integer to $N/2.4$), and compute

$$a = \sum_{n=1}^{l} \underline{x}_n \overline{w}_n + \sum_{n=l+1}^{N} \underline{x}_n \underline{w}_n \qquad \text{–2–)}$$

$$b = \sum_{n=1}^{l} \overline{w}_n + \sum_{n=l+1}^{N} \underline{w}_n \qquad \text{–29)}$$

$$y = a/b \qquad \text{–30)}$$

3) Find $l' \in [1, N-1]$ such that

$$\underline{x}_{l'} < y \leqslant \underline{x}_{l'+1} \qquad \text{–31)}$$

4) If $l' = l$, stop–otherwise, continue.

–) Compute $s = \text{sign}(l' - l)$, and

$$a' = a + s \sum_{n=\min(l,l')+1}^{\max(l,l')} \underline{x}_n (\overline{w}_n - \underline{w}_n) \qquad \text{–32)}$$

$$b' = b + s \sum_{n=\min(l,l')+1}^{\max(l,l')} (\overline{w}_n - \underline{w}_n) \qquad \text{–33)}$$

$$y' = a'/b' \qquad \text{–34)}$$

–) Set $y = y'$, $a = a'$, $b = b'$ and $l = l'$. – o to Step 3.

EKMA for computing y_r:

1) The same as Step –1) of KMA for computing y_r.
2) Set $r = [N/1.7]$ –the nearest integer to $N/1.7$), and compute

$$a = \sum_{n=1}^{r} \overline{x}_n \underline{w}_n + \sum_{n=r+1}^{N} \overline{x}_n \overline{w}_n \qquad \text{–3–)}$$

$$b = \sum_{n=1}^{r} \underline{w}_n + \sum_{n=r+1}^{N} \overline{w}_n \qquad \text{–3–)}$$

$$y = a/b \qquad \text{–3–)}$$

3) Find $r' \in [1, N-1]$ such that

$$\overline{x}_{r'} < y \leqslant \overline{x}_{r'+1} \qquad \text{–3–)}$$

4) If $r' = r$, stop–otherwise, continue.

–) Compute $s = \text{sign}(r' - r)$, and

$$a' = a - s \sum_{n=\min(r,r')+1}^{\max(r,r')} \overline{x}_n (\overline{w}_n - \underline{w}_n) \qquad \text{–39)}$$

$$b' = b - s \sum_{n=\min(r,r')+1}^{\max(r,r')} (\overline{w}_n - \underline{w}_n) \qquad \text{–40)}$$

$$y' = a'/b' \qquad \text{–41)}$$

–) Set $y = y'$, $a = a'$, $b = b'$ and $r = r'$. – o to Step –3).

Note that similar to KMA, sometimes due to roundoff error $l' = l$ and-or $r' = r$ cannot be achieved and EKMA runs into infinite loops. Special attentions need to be paid to these cases, as done in the Appendix.

B. Enhanced Karnik-Mendel Algorithm with New Initialization (EKMANI)

–eh et al. –19– proposed Enhanced Karnik-Mendel Algorithm with new initialization –EKMANI) to compute the generalized centroid of general T2 FSs. It is based on the observation that for two α-planes –20–close to each other, the centroids of the resulting two IT2 FSs are also close to each other. So, it may be advantageous to use the switch points obtained from the previous α-plane to initialize the switch points in the next α-plane. Though EKMANI was primarily designed for computing the generalized centroid, it may also be used in type-reduction of IT2 FLSs, especially IT2 fuzzy logic controllers –FLCs), because usually the output of an IT2 FLC changes only a small amount each step.

EKMANI for computing y_l is identical to EKMA for computing y_l, except that in Step –2), if there is a switch point obtained from previous computation, then set l to it–otherwise, set $l = [N/2.4]$.

EKMANI for computing y_r is identical to EKMA for computing y_r, expect that in Step –2), if there is a switch point obtained from previous computation, then set r to it–otherwise, set $r = [N/1.7]$.

C. Iterative Algorithm with Stop Condition (IASC)

Melgare-o and his co-authors –1—, –1— proposed two efficient algorithms for computing the generalized centroid of IT2 FSs, which can also be used in type-reduction of IT2 FLSs.

The faster one, called iterative algorithm with stop condition –IASC) –1—, is introduced in this subsection[4]. It is based on the fact –4– that y_l in –10) first monotonically decreases and then monotonically increases with the increase of k, and y_r in –12) first monotonically increases and then monotonically decreases with the increase of k.

IASC for computing y_l:

1) The same as Step –1) of KMA for computing y_l.
2) Initialize

$$a=\sum_{n=1}^{N}\underline{x}_n\underline{w}_n, \qquad b=\sum_{n=1}^{N}\underline{w}_n$$
$$y_l=\underline{x}_N, \qquad l=0$$

3) Compute

$$\begin{aligned} l&=l+1\\ a&=a+\underline{x}_l(\overline{w}_l-\underline{w}_l)\\ b&=b+\overline{w}_l-\underline{w}_l\\ c&=a/b \end{aligned}$$

4) If $c>y_l$, set $l=l-1$ and stop– otherwise, set $y_l=c$ and go to Step –3).

IASC for computing y_r:

1) The same as Step –1) of KMA for computing y_r.
2) Initialize

$$a=\sum_{n=1}^{N}\overline{x}_n\overline{w}_n, \qquad b=\sum_{n=1}^{N}\overline{w}_n$$
$$y_r=\overline{x}_1, \qquad r=0$$

3) Compute

$$\begin{aligned} r&=r+1\\ a&=a-\overline{x}_r(\overline{w}_r-\underline{w}_r)\\ b&=b-\overline{w}_r+\underline{w}_r\\ c&=a/b \end{aligned}$$

4) If $c<y_r$, set $r=r-1$ and stop– otherwise, set $y_r=c$ and go to Step –3).

D. Enhanced Iterative Algorithm with Stop Condition (EIASC)

An Enhanced IASC –EIASC) is proposed in this subsection. It uses the following theorem:

Theorem 1: $f_l(k)$ in —) satisfies:

$$f_l(k)\begin{cases}\geqslant \underline{x}_k, & k\leqslant l\\ <\underline{x}_k, & k>l\end{cases} \qquad \text{–42)}$$

Similarly, $f_r(k)$ in –9) satisfies:

$$f_r(k)\begin{cases}\leqslant \overline{x}_k, & k>r\\ >\overline{x}_k, & k\leqslant r\end{cases}. \quad \square \qquad \text{–43)}$$

Proof: Because the proof for –43) is very similar to that for –42), we only provide the proof for –42) in this paper.

[4] –1— did not give the complete IASC presented in this subsection. It is the authors' integration of the ideas in –1— and –1—.

From –10) and –14) it follows that

$$f_l(l)\geqslant \underline{x}_l. \qquad \text{–44)}$$

For $\forall k\in[1,N-1]$, it is always true that

$$f_l(k)\geqslant f_l(l) \qquad \text{–4–)}$$

because by definition $f_l(l)=\min\limits_{k=1,\ldots,N-1} f_l(k)$. – hen $k\leqslant l$, it is also true that

$$\underline{x}_l\geqslant \underline{x}_k \qquad \text{–4–)}$$

because $\{\underline{x}_n\}$ has been sorted in ascending order and $k\leqslant l$. Considering –44)--4–) together, it follows that

$$f_l(k)\geqslant f_l(l)\geqslant \underline{x}_l\geqslant \underline{x}_k \qquad \text{–4–)}$$

which is the first line of –42). Next we focus on the second line of –42).

Beginning with —), we see that

$$f_l(l)\left[\sum_{n=1}^{l}\overline{w}_n+\sum_{n=l+1}^{N}\underline{w}_n\right]=\sum_{n=1}^{l}\underline{x}_n\overline{w}_n+\sum_{n=l+1}^{N}\underline{x}_n\underline{w}_n \qquad \text{–4–)}$$

Let $k>l$ and add $\sum\limits_{n=l+1}^{k}\underline{x}_n(\overline{w}_n-\underline{w}_n)$ to both sides of –4–), in order to see that

$$\begin{aligned} &f_l(l)\left[\sum_{n=1}^{l}\overline{w}_n+\sum_{n=l+1}^{N}\underline{w}_n\right]+\sum_{n=l+1}^{k}\underline{x}_n(\overline{w}_n-\underline{w}_n)\\ =&\sum_{n=1}^{k}\underline{x}_n\overline{w}_n+\sum_{n=k+1}^{N}\underline{x}_n\underline{w}_n \end{aligned} \qquad \text{–49)}$$

Because $k>l$, $\underline{x}_k>f_l(l)$, $\sum\limits_{n=1}^{k}\overline{w}_n+\sum\limits_{n=k+1}^{N}\underline{w}_n>0$ and $\overline{w}_n-\underline{w}_n>0$,

$$f_l(l)\left[\sum_{n=1}^{l}\overline{w}_n+\sum_{n=l+1}^{N}\underline{w}_n\right]<\underline{x}_k\left[\sum_{n=1}^{l}\overline{w}_n+\sum_{n=l+1}^{N}\underline{w}_n\right] \qquad \text{—0)}$$

$$\sum_{n=l+1}^{k}\underline{x}_n(\overline{w}_n-\underline{w}_n)<\underline{x}_k\sum_{n=l+1}^{k}(\overline{w}_n-\underline{w}_n) \qquad \text{—1)}$$

– e obtain the following inequality from the above three inequalities:

$$\begin{aligned} &\sum_{n=1}^{k}\underline{x}_n\overline{w}_n+\sum_{n=k+1}^{N}\underline{x}_n\underline{w}_n\\ <&\underline{x}_k\left[\sum_{n=1}^{l}\overline{w}_n+\sum_{n=l+1}^{N}\underline{w}_n\right]+\underline{x}_k\sum_{n=l+1}^{k}(\overline{w}_n-\underline{w}_n)\\ =&\underline{x}_k\left[\sum_{n=1}^{k}\overline{w}_n+\sum_{n=k+1}^{N}\underline{w}_n\right] \end{aligned} \qquad \text{—2)}$$

Because $\sum\limits_{n=1}^{k}\overline{w}_n+\sum\limits_{n=k+2}^{N}\underline{w}_n>0$, it follows that

$$\underline{x}_k>\frac{\sum_{n=1}^{k}\underline{x}_n\overline{w}_n+\sum_{n=k+1}^{N}\underline{x}_n\underline{w}_n}{\sum_{n=1}^{k}\overline{w}_n+\sum_{n=k+1}^{N}\underline{w}_n}=f_l(k). \quad \square \qquad \text{—3)}$$

Our proposed EIASC makes the following two improvements over the IASC:

1) New stopping criterion based on Theorem 1.
2) Both the IASC for computing y_l and the IASC for computing y_r start from switch point 1 and increase it gradually to find the correct switch points. This is reasonable for y_l, since it has been shown in –1– that for a variety of scenarios, its switch point l is smaller than $N/2$–so, it is more efficient to search from $l = 1$ instead of from $l = N - 1$. However, setting the initial switch point $r = 1$ may not be efficient for y_r, since it has been shown in –1– that generally its switch point $r > N/2$. So, it would be more efficient if for y_r one initializes the switch point as $r = N - 1$ and then gradually decreases it until the correct r is found.

In summary, the EIASC for computing y_l and y_r are:

EIASC for computing y_l:

1) The same as Step –1) of KMA for computing y_l.
2) Initialize

$$a = \sum_{n=1}^{N} \underline{x}_n \underline{w}_n, \qquad b = \sum_{n=1}^{N} \underline{w}_n$$
$$y_l = \underline{x}_N, \qquad l = 0$$

3) Compute

$$l = l + 1, \qquad a = a + \underline{x}_l(\overline{w}_l - \underline{w}_l)$$
$$b = b + \overline{w}_l - \underline{w}_l, \qquad c = a/b$$

4) If $c > y_l$, set $l = l - 1$ and stop– otherwise, set $y_l = c$ and go to Step –3).

EIASC for computing y_r:

1) The same as Step –1) of KMA for computing y_r.
2) Initialize

$$a = \sum_{n=1}^{N} \overline{x}_n \underline{w}_n, \qquad b = \sum_{n=1}^{N} \underline{w}_n$$
$$y_r = \overline{x}_1, \qquad r = N$$

3) Compute

$$a = a + \overline{x}_r(\overline{w}_r - \underline{w}_r), \qquad b = b + \overline{w}_r - \underline{w}_r$$
$$c = a/b, \qquad r = r - 1$$

4) If $c < y_r$, set $r = r + 1$ and stop– otherwise, set $y_r = c$ and go to Step –3).

Clearly, based on Theorem 1, given an initialization of l –or r), one can easily tell whether it is on the left or right of the true l –or r), and hence the new initialization idea in EKMANI can also be used in EIASC. – e implemented the corresponding algorithm, where the user can specify an initial l –or r)– otherwise, l is initialized to $[N/2.4]$ –r to $[N/1.7]$), the same as EKMA. Interestingly, this new algorithm only outperforms EIASC when $N > 1000$. Since in practice usually $k < 1000$, this new algorithm is not included in this paper. However, it can be downloaded from the first author's website at http:–www-scf.usc.edu–~dongruiw–files–TR_Algorithms.zip.

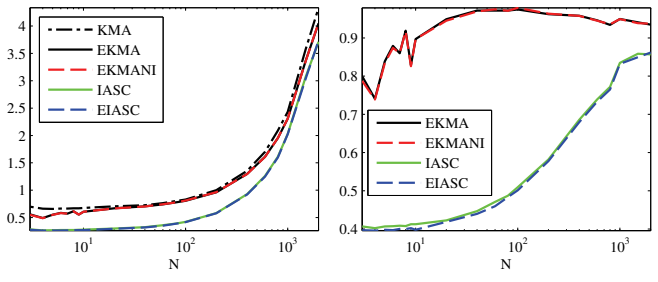
I–. Experimental Comparison

In this section we compare the performance of the five type-reduction algorithms –KMA, EKMA, EKMANI, IASC, and EIASC) using three simulations. The platform is a Lenovo Thinkpad T–00 with Intel Core2 Duo CPU P—00– 2.40Hz and 3– B memory, running – indows – 32-bit Home Premium and Matlab R2009a. The computational cost is measured by the computation time, obtained from Matlab *tic* and *toc* functions.

A. Comparison 1: Uniformly Distributed X_n and W_n

In this comparison, we assume the elements in X_n and W_n are uniformly distributed, $\underline{x}_n = \overline{x}$, and $N = \{3, 4, ..., 10, 20, ..., 100, 200, ..., 1000, 1500, 2000\}$ were used. For each N, –000 Monte Carlo simulations were used to compute y. The results are shown in Fig. 2. Observe that:

1) All four efficient algorithms outperformed KMA.
2) The performances of EKMA and EKMANI were almost identical in this simulation because for each N all –000 Monte Carlo simulations were independent, so it does not make sense to initialize a switch point from a previous simulation. – e did not supply the new initial switch points in EKMANI in this simulation, so it reduced to EKMA. Also, the performance improvement of EKMA over KMA is not as large as that reported in –1– since for each algorithm in this paper we added the same preprocessing steps to eliminate possible numerical problems and the preprocessing time was included in the computational cost, whereas in –1– preprocessing time was not considered.
3) Both IASC and EIASC significantly outperformed KMA, especially when N is small –$N \leqslant 100$). And, EIASC slightly outperformed IASC. The computational cost of both IASC and EIASC increase rapidly as N increases since they need to evaluate many possible switch points before finding the correct ones.



–a) –b)

Fig. 2. –a) Total computation time –seconds) of the –000 Monte Carlo simulations of uniformly distributed X_n and W_n –, $= 1, 2, ..., N$) for different N. –b) The ratio of the computation time of the four efficient algorithms with respect to KMA.

B. Comparison 2: Computing the Generalized Centroid of A General Type-2 FS

In this simulation, we compare the performance of the five type-reduction algorithms in computing the generalized

centroid of a general type-2 FS, which is originally defined in [20] as:

$$\mu_{\tilde{A}}(x) = \begin{cases} \frac{z-d(x)}{p(x)-d(x)}, & d(x) \leqslant z \leqslant p(x) \\ \frac{u(x)-z}{u(x)-p(x)}, & p(x) \leqslant z \leqslant u(x) \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

$$u(x) = \max(f_1(x), f_2(x)) \quad (5)$$

$$d(x) = \max(g_1(x), g_2(x)) \quad (6)$$

$$f_1(x) = \exp\left(-\frac{(x-3)^2}{8}\right) \quad (7)$$

$$f_2(x) = 0.8\exp\left(-\frac{(x-6)^2}{8}\right) \quad (8)$$

$$g_1(x) = 0.5\exp\left(-\frac{(x-3)^2}{2}\right) \quad (9)$$

$$g_2(x) = 0.4\exp\left(-\frac{(x-6)^2}{2}\right) \quad (10)$$

$$p(x) = d(x) + w(x)(u(x) - d(x)) \quad (11)$$

where $\mu_{\tilde{A}}(x)$ is the secondary membership grade, $z \in [d(x), u(x)]$ is the primary membership grade, and $w(x)$ is a random number in [0, 1]. An example of the type-2 FS is shown in Fig. 3.

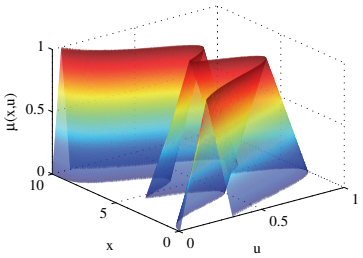


Fig. 3. An example of the type-2 FS in Comparison 2.

In [19] it has been shown that EKMANI achieved a much better performance than KMA and EKMA. As in [19], we fixed $N = 200$ and took the number of α-planes as $\{5, 10, 20, 50, 75, 100\}$. For each number of α-planes, we repeated the experiment 1000 times to get a more accurate computation time. The results are shown in Fig. 4. Observe that:

1) All four efficient algorithms outperformed KMA.
2) EKMANI outperformed EKMA significantly since in EKMANI the switch points from the previous α-plane were used as the initial switch points in the next α-plane. As usually y_l and y_r change very slightly from one α-plane to the next (especially when the number of α-planes is large), the switch points in two α-planes are very close to each other, and hence the new initialization strategy is better than that in EKMA.
3) Both IASC and EIASC significantly outperformed all other three algorithms, and EIASC also outperformed IASC slightly.

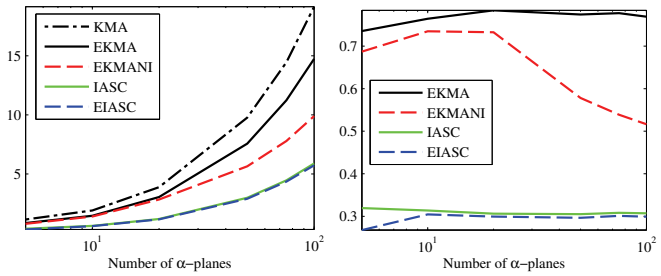


Fig. 4. (a) Total computation time (seconds) of the 1000 runs of centroid computation for different number of α-planes. (b) The ratio of the computation time of the four efficient algorithms with respect to KMA.

C. Comparison 3: IT2 Fuzzy Logic Controller Design

In this subsection we compare the performance of the five algorithms in FLC design using evolutionary computation [21], where the performance of a large number of (usually randomly generated) FLCs are evaluated. The following simple first-order plus dead-time plant is employed as the nominal system [22]:

$$G(s) = \frac{K}{\tau s + 1}e^{-Ls} = \frac{1}{10s+1}e^{-2.5s} \quad (12)$$

The goal is to design an IT2 fuzzy PI controller

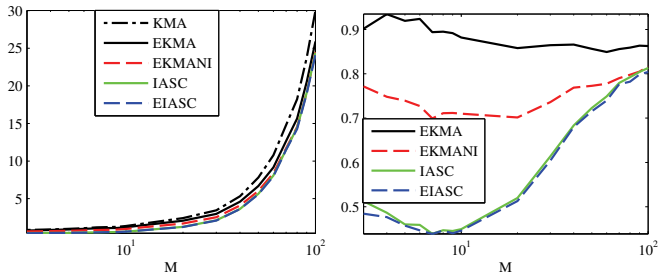
$$\dot{u} = k_P\dot{e} + k_I e \quad (13)$$

where $\dot{u}$ is the change in control signal, e is the error, $\dot{e}$ is the change of error, and k_P and k_I are proportional and integral gains. Assume there are M (Gaussian IT2 FSs in each domain (for e and $\dot{e}$), and each IT2 FSs is determined by three parameters (one mean, m_m, and two standard deviations, σ_m^1 and σ_m^2, $m = 1, 2, ..., M$). Each of the M^2 rule consequents is represented by a crisp number y_n, $n = 1, ..., M^2$. Then, for each input pair $(e, \dot{e})$, all $N = M^2$ rules are fired, and a type-reduction algorithm is needed to compute the output of the IT2 FLC.

Assume the population consists of 100 randomly generated IT2 FLCs (all m_m, σ_m^1, σ_m^2, and y_n are random), and the performance of each FLC is evaluated by a step response in the first 100 seconds with sampling frequency 1 Hz. (We recorded the time that the five algorithms were used to perform type-reduction for these $100 \times 100 = 10000$ input pairs. $M = \{3, 4, ..., 10, 20, ..., 100\}$ were used. The results are shown in Fig. (). Observe that:

1) All four efficient algorithms outperformed KMA.
2) EKMANI outperformed EKMA significantly since in EKMANI the switch points from the previous step were used as the initial switch points in the next step. As usually e and $\dot{e}$ change very slightly from one step to the next (especially when the sampling frequency is high, and/or during steady state), the switch points in two steps are very close to each other, and hence the new initialization strategy is better than that in EKMA.

3) Both IASC and EIASC significantly outperformed all other three algorithms when M is small ($M \leqslant 20$), and EIASC also outperformed IASC slightly.



fia) fib)

Fig. fi. fia) Total computation time fiseconds) of the 100 IT2 FLCs for different M, the number of IT2 FSs in each input domain. Note that $N = M^2$. fib) The ratio of the computation time of the efficient algorithms with respect to KMA.

fi. Conclusions

Type-reduction algorithms are very important for type-2 fuzzy sets and systems. However, the most popular Karnik-Mendel Algorithms is computationally intensive. In this paper we have reviewed several more efficient type-reduction algorithms, and also improved over one of them. Experimental results showed that our proposed algorithm is the most efficient one to use. Particularly, when $N \leqslant 100$, which is true in most practical type-reduction computations, our proposed algorithm fEIASC) can save over fi0% computational cost over the Karnik-Mendel Algorithms. An Matlab implementation of EIASC is also given. It includes preprocessing steps to eliminate numerical problems. This algorithm will be very helpful in promoting the popularity of type-2 fuzzy sets and systems.

Appendix A
Matlab Implementation of EIASC

This appendix presents an Matlab implementation of EIASC. Note that it is more complex than the algorithm presented in Section III-D since here we include the following preprocessing steps to eliminate possible numerical problems:

1) fi hen all $\underline{w}_n$ are zero or all $\overline{w}_n$ are very close to zero, e.g., $\overline{w}_n < 10^{-10}$ for $\forall n$, we return $y_l = \min_n \underline{x}_n$ and $y_r = \max_n \overline{x}_n$.
2) All very small $\overline{w}_n$, e.g., $\overline{w}_n < 10^{-10}$, and their associated $\underline{x}_n$, $\overline{x}_n$, and $\underline{w}_n$, are removed to prevent numerical instability.
3) $\{\underline{x}_n\}$ and $\{\overline{x}_n\}$ are made unique respectively to prevent possible infinite loops.

A user can easily modify the codes for additional computational cost saving. For example, Step fi3) above may be done only once *a priori* in an FLC. Additionally, $\{\underline{x}_n\}$ and $\{\overline{x}_n\}$ can be sorted only once *a priori* in an FLC, and they do not need to be sorted in computing the generalized centroid of type-2 fuzzy sets.

The following four input parameters are mandatory:

- *Xl*: A row vector containing $\underline{x}_n$, $n = 1, 2, ..., N$.
- *Xr*: A row vector containing $\overline{x}_n$, $n = 1, 2, ..., N$. If $\underline{x}_n = \overline{x}_n$ $\forall n$, then use ffi as input for *Xr*.
- *Wl*: A row vector containing $\underline{w}_n$, $n = 1, 2, ..., N$.
- *Wr*: A row vector containing $\overline{w}_n$, $n = 1, 2, ..., N$.

The following input is optional:

- *needSort*: "1" if at least one of *Xl* and *Xr* is not in ascending order. "0" if both *Xl* and *Xr* are in ascending order. Default "1."

The outputs are:

- *y*: Mid-point of Y in ffi). $y = (yl + yr)/2$.
- *yl*: y_l in ffi).
- *yr*: y_r in ffi).
- *l*: Switch point l for computing *yl*.
- *r*: Switch point r for computing *yr*.

Matlab implementations of all other algorithms introduced in this paper can be found from the first author's website at http:ffiwww-scf.usc.edufi~dongruiwffilesfiTR_Algorithms.zip.

```
function [y,yl,yr,l,r]=EIASC(Xl,Xr,...
                         Wl,Wr,needSort)

ly=length(Xl);
XrEmpty=isempty(Xr);
if XrEmpty; Xr=Xl; end

if max(Wl)==0
    yl=min(Xl);  yr=max(Xr);
    y=(yl+yr)/2;
    l=1;   r=ly-1;
    return;
end

index=find(Wr<10^(-10));
if length(index)==ly
    yl=min(Xl);  yr=max(Xr);
    y=(yl+yr)/2;
    l=1;  r=ly-1;
    return;
end

Xl(index)=[]; Xr(index)=[];
Wl(index)=[]; Wr(index)=[];

if nargin==4; needSort=1; end

% Compute yl
if  needSort
    [Xl,index]=sort(Xl);
    Xr=Xr(index);
    Wl=Wl(index);
    Wr=Wr(index);
end
Wl2=Wl;  Wr2=Wr;
```

```
for i=length(Xl):-1:2 % Make Xl unique
    if Xl(i)==Xl(i-1)
        Wl(i)=Wl(i)+Wl(i-1);
        Wr(i)=Wr(i)+Wr(i-1);
        Xl(i)=[];
        Wl(i-1)=[]; Wr(i-1)=[];
    end
end

ly=length(Xl);
if ly==1
    yl=Xl;  l=1;
else
    yl=Xl(end); l=0;
    a=Xl*Wl'; b=sum(Wl);
    while yl > Xl(l+1)
        l=l+1;
        a=a+Xl(l)*(Wr(l)-Wl(l));
        b=b+Wr(l)-Wl(l);
        yl=a/b;
    end
end

% Compute yr
if ~XrEmpty && needSort==1
    [Xr,index]=sort(Xr);
    Wl=Wl2(index); Wr=Wr2(index);
end

if ~XrEmpty
    for i=length(Xr):-1:2 % Make Xr unique
        if Xr(i)==Xr(i-1)
            Wl(i)=Wl(i)+Wl(i-1);
            Wr(i)=Wr(i)+Wr(i-1);
            Xr(i)=[];
            Wl(i-1)=[]; Wr(i-1)=[];
        end
    end
end

ly=length(Xr);
if ly==1
    yr=Xr; r=1;
else
    r=ly; yr=Xr(1);
    a=Xr*Wl'; b=sum(Wl);
    while yr < Xr(r)
        a=a+Xr(r)*(Wr(r)-Wl(r));
        b=b+Wr(r)-Wl(r);
        yr=a/b; r=r-1;
    end
end

y=(yl+yr)/2;
```

REFERENCES

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

[2] N. N. Karnik and J. M. Mendel, "Centroid of a type-2 fuzzy set," *Information Sciences*, vol. 132, pp. 195–220, 2001.

[3] 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 Trans. on Fuzzy Systems*, vol. 15, no. 2, pp. 309–320, 2007.

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

[5] D. Wu and J. M. Mendel, "Aggregation using the linguistic weighted average and interval type-2 fuzzy sets," *IEEE Trans. on Fuzzy Systems*, vol. 15, no. 6, pp. 1145–1161, 2007.

[6] ——, "Corrections to 'Aggregation using the linguistic weighted average and interval type-2 fuzzy sets'," *IEEE Trans. on Fuzzy Systems*, vol. 16, no. 6, pp. 1664–1666, 2008.

[7] J. M. Mendel and D. Wu, *Perceptual Computing: Aiding People in Making Subjective Judgments*. Hoboken, NJ: Wiley-IEEE Press, 2010.

[8] D. Wu and J. M. Mendel, "Computing with words for hierarchical decision making applied to evaluating a weapon system," *IEEE Trans. on Fuzzy Systems*, vol. 18, no. 3, pp. 441–460, 2010.

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

[10] D. Wu and W. W. Tan, "Computationally efficient type-reduction strategies for a type-2 fuzzy logic controller," in *Proc. IEEE Int'l Conf. on Fuzzy Systems*, Reno, NV, May 2005, pp. 353–358.

[11] M. Nie and W. W. Tan, "Towards an efficient type-reduction method for interval type-2 fuzzy logic systems," in *Proc. IEEE Int'l Conf. on Fuzzy Systems*, Hong Kong, June 2008, pp. 1425–1432.

[12] M. Begian, W. Melek, and J. Mendel, "Stability analysis of type-2 fuzzy systems," in *Proc. IEEE Int'l Conf. on Fuzzy Systems*, Hong Kong, June 2008, pp. 947–953.

[13] X. Du and H. Ying, "Control performance comparison between a type-2 fuzzy controller and a comparable conventional mamdani fuzzy controller," in *Proc. Annual Meeting of the North American Fuzzy Information Processing Society*, San Diego, CA, June 2007, pp. 100–105.

[14] S. Greenfield, F. Chiclana, S. Coupland, and R. John, "The collapsing method of defuzzification for discretised interval type-2 fuzzy sets," *Information Sciences*, 2007, in press.

[15] R. Sepulveda, O. Castillo, P. Melin, and O. Montiel, "An efficient computational method to implement type-2 fuzzy logic in control applications," in *Analysis and Design of Intelligent Systems using Soft Computing Techniques*, P. Melin, O. Castillo, E. Ramirez, J. Kacprzyk, and W. Pedrycz, Eds. Berlin/Heidelberg: Springer, 2007, pp. 45–52.

[16] D. Wu, "An interval type-2 fuzzy logic system cannot be implemented by traditional type-1 fuzzy logic systems," in *Proc. World Conference on Soft Computing*, San Francisco, CA, May 2011.

[17] M. Melgarejo, "A fast recursive method to compute the generalized centroid of an interval type-2 fuzzy set," in *Proc. Annual Meeting of the North American Fuzzy Information Processing Society*, San Diego, CA, June 2007, pp. 190–194.

[18] K. Duran, H. Bernal, and M. Melgarejo, "Improved iterative algorithm for computing the generalized centroid of an interval type-2 fuzzy set," in *Proc. Annual Meeting of the North American Fuzzy Information Processing Society*, New York, May 2007, pp. 1–7.

[19] C.-W. Wu, W.-H. Jeng, and S.-J. Lee, "An enhanced type-reduction algorithm for type-2 fuzzy sets," *IEEE Trans. on Fuzzy Systems*, 2011, in press.

[20] F. Liu, "An efficient centroid type-reduction strategy for general type-2 fuzzy logic system," *Information Sciences*, vol. 178, no. 9, pp. 2224–2236, 2008.

[21] D. B. Fogel, *Evolutionary computation: Toward a new philosophy of machine intelligence*, 3rd ed. Hoboken, NJ: Wiley-IEEE Press, 2006.

[22] D. Wu and W. W. Tan, "Interval type-2 fuzzy PI controllers: why they are more robust," in *Proc. IEEE Int'l. Conf. on Granular Computing*, San Jose, CA, August 2010, pp. 802–807.