

Approaches for Reducing the Computational Cost of Interval Type-2 Fuzzy Logic Systems: Overview and Comparisons

Dongrui Wu, *Member, IEEE*

Abstract—Interval type-2 fuzzy logic systems (IT2 FLSs) have demonstrated better abilities to handle uncertainties than their type-1 (T1) counterparts in many applications; however, the high computational cost of the iterative Karnik–Mendel (KM) algorithms in type-reduction means that it is more expensive to deploy IT2 FLSs, which may hinder them from certain cost-sensitive real-world applications. This paper provides a comprehensive overview and comparison of three categories of methods to reduce their computational cost. The first category consists of five enhancements to the KM algorithms, which are the most popular type-reduction algorithms to date. The second category consists of 11 alternative type-reducers, which have closed-form representations and, hence, are more convenient for analysis. The third category consists of a simplified structure for IT2 FLSs, which can be combined with any algorithms in the first or second category for further computational cost reduction. Experiments demonstrate that almost all methods in these three categories are faster than the KM algorithms. This overview and comparison will help researchers and practitioners on IT2 FLSs choose the most suitable structure and type-reduction algorithms, from a computational cost perspective. A recommendation is given in the conclusion.

Index Terms—Computational cost, fuzzy logic control, interval type-2 fuzzy logic system (IT2 FLS), Karnik–Mendel (KM) algorithms, type-reduction (TR).

I. INTRODUCTION

TYPE-2 fuzzy sets (T2 FSs) were first proposed by Zadeh in 1975 [65] as an extension to type-1 (T1) FSs. The main difference between a T2 FS and a T1 FS is that the memberships in a T2 FS are themselves T1 FSs instead of crisp numbers in a T1 FS. T2 FSs and fuzzy logic systems (FLSs) have been gaining considerable attentions recently. Particularly, people are interested in interval type-2 (IT2) FSs [33], whose memberships are intervals (instead of T1 FSs in a general T2 FS), for their simplicity and reduced computational cost. IT2 FSs and FLSs have been used in many applications, including computing with words [35], [41], [51], linguistic data summarization [38], [52], modeling and control [6], [9], [18], [20], [40], [58], [59], pattern recognition [19], [32], [36], [42], [62], [66], recommendation

Manuscript received October 27, 2011; revised February 26, 2012; accepted May 11, 2012. Date of publication May 30, 2012; date of current version January 30, 2013.

The author is with the Machine Learning Lab, GE Global Research, Niskayuna, NY 12065 USA (e-mail: wud@ge.com).

Color versions of one or more of the figures in this paper are available online at <http://ieeexplore.ieee.org>.

Digital Object Identifier 10.1109/TFUZZ.2012.2201728

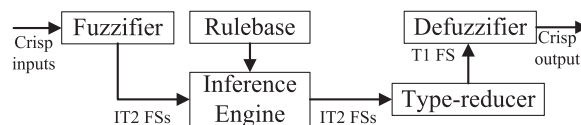


Fig. 1. Schematic diagram of an IT2 FLS.

systems [17], [25], etc., and they often demonstrate better performance than their T1 counterparts.

Fig. 1 shows the schematic diagram of an IT2 FLS. It is similar to its T1 counterpart, the major difference being that at least one of the FSs in the rulebase is an IT2 FS. Hence, the outputs of the inference engine are IT2 FSs, and a type-reducer [24], [33] is needed to convert them into a T1 FS before defuzzification can be carried out. Type-reduction (TR) is usually performed by the iterative Karnik–Mendel (KM) algorithms [24], which are computationally intensive. The extra computational cost of IT2 FLSs over T1 FLSs may hinder them from certain cost-sensitive real-world applications, because an IT2 FLS requires a more powerful processor, which increases the cost of the overall system.

There have been many different approaches to reduce the computational cost of IT2 FLSs, including both hardware implementation optimization (e.g., [43]) and software algorithmic optimization. This paper presents a comprehensive overview and comparison of methods in the latter approach, which can be grouped into three categories:¹

- 1) *Enhancements to the KM TR algorithms* [11], [21], [22], [31], [50], [53], [64]: These improve directly over the original KM TR algorithms to speed them up.

¹All algorithms in these three categories were originally proposed to save computational cost or to simplify IT2 FLS design. There are also other algorithms proposed to improve the accuracy of the TR set or the control performance. For example, Liu *et al.* [30] proposed several weighted enhanced KM (WEKM) algorithms for computing the centroid of IT2 FSs and showed that they can obtain more accurate centroids than the enhanced KM algorithms [50], given the same number of discretizations (the ground truth centroids are computed from the continuous case); however, they are slower than the enhanced KM algorithms and cannot be used for TR of IT2 FLSs. Therefore, the WEKM algorithms are not considered in this paper. Ulu *et al.* [45] proposed a dynamic defuzzification method for IT2 FLCs, in which the TR interval $[y_l, y_r]$ is still computed by the KM algorithms, but the defuzzified output is computed as $\alpha(t)y_l + (1 - \alpha(t))y_r$ [instead of $(y_l + y_r)/2$ in the standard defuzzification approach], where $\alpha(t)$ is a function of the controller inputs. They claimed that the dynamic defuzzification method can achieve better control performance. However, since the primary goal of this paper is to compare different approaches to reduce the computational cost of IT2 FLSs instead of comparing their performance and the dynamic defuzzification method has the same computational cost as the standard defuzzification method, it is not elaborated in this paper.

- 2) *Alternative TR algorithms* [2], [8], [10], [13], [16], [27], [28], [37], [44], [56], [61]: Unlike the iterative KM algorithms, most alternative TR algorithms have closed-form representations. They are usually fast approximations of the KM algorithms.
- 3) *Simplified IT2 FLSs* [54], [59]: In these, the architecture of an IT2 FLS is simplified by using only a small number of IT2 FSs for the most critical input regions and T1 FSs for the rest.

Note that Categories I and II are mutually exclusive, i.e., a method in Category I cannot be combined with a method in Category II for further computational cost reduction; however, the method in Category III can be combined with a method in Category I or II to further improve speed. We will demonstrate that in Section V. In addition, the first two categories of methods can be applied to any IT2 FLS, whereas the simplified structure was designed for IT2 fuzzy logic controllers (FLCs) and it has not been used in other FLSs.

The rest of this paper is organized as follows. Section II presents background materials on IT2 FLSs, including their operations and the KM algorithms. Section III introduces five enhanced versions of KM algorithms. Section IV describes 11 alternative TR algorithms, which have closed-form representations. Section V presents a simplified architecture for IT2 FLCs and discusses how the simplified architecture can be combined with a Category 1 or 2 method for further computational cost reduction. Finally, Section VI draws the conclusion.

II. INTERVAL TYPE-2 FUZZY SETS AND FUZZY LOGIC SYSTEMS

For the completeness of this paper, background materials on IT2 FSs and FLSs are presented in this section.

A. Interval Type-2 Fuzzy Sets

An IT2 FS [33], [34] \tilde{X} is characterized by its membership function (MF) $\mu_{\tilde{X}}(x, u)$, i.e.,

$$\tilde{X} = \int_{x \in D_{\tilde{X}}} \int_{u \in J_x \subseteq [0,1]} \mu_{\tilde{X}}(x, u)/(x, u)$$

where x , called the *primary variable*, has domain $D_{\tilde{X}}$; $u \in [0, 1]$, called the *secondary variable*, has domain $J_x \subseteq [0, 1]$ at each $x \in D_{\tilde{X}}$; J_x is also called the *support of the secondary MF*; and, the amplitude of $\mu_{\tilde{X}}(x, u)$, called a *secondary grade* of \tilde{X} , equals 1 for $\forall x \in D_{\tilde{X}}$ and $\forall u \in J_x \subseteq [0, 1]$.

An example of an IT2 FS \tilde{X} is shown in Fig. 2. Observe that unlike a T1 FS, whose membership grade for each x is a number, the membership of an IT2 FS is an interval. Observe also that an IT2 FS is bounded from above and below by two T1 FSs, \bar{X} and \underline{X} , which are called *upper membership function (UMF)* and *lower membership function (LMF)*, respectively. The area between \bar{X} and \underline{X} is the *footprint of uncertainty (FOU)*. An *embedded T1 FS*² is any T1 FS within the FOU. \underline{X} and \bar{X} are two such sets.

²According to the Mendel–John representation theorem [34], an embedded T1 FS can be subnormal and nonconvex. Recently, there are arguments that only convex and normal T1 FSs [46], or only T1 FSs assuming a particular shape [1], [12], should be considered as embedded T1 FSs.

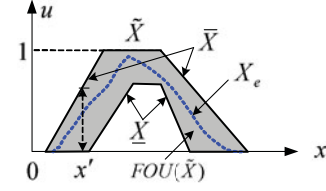


Fig. 2. IT2 FS. \underline{X} (the LMF), \bar{X} (the UMF), and X_e are the three embedded T1 FSs.

B. Interval Type-2 Fuzzy Logic Sets

An IT2 FLS is an FLS containing at least one IT2 FSs. Without loss of generality, consider the rulebase of an IT2 FLS consisting of \tilde{N} rules assuming the following form:

\tilde{R}^n : IF x_1 is \tilde{X}_1^n and \dots and x_I is \tilde{X}_I^n , THEN y is Y^n where \tilde{X}_i^n ($i = 1, \dots, I$) are IT2 FSs, and $Y^n = [y^n, \bar{y}^n]$ is an interval, which can be understood as the centroid [23], [33] of a consequent IT2 FS,³ or the simplest Takagi–Sugeno–Kang (TSK) model. In many applications [55], [58], [59], we use $\underline{y}^n = \bar{y}^n$, i.e., each rule consequent is represented by a crisp number.

For an input vector $\mathbf{x}' = (x'_1, x'_2, \dots, x'_I)$, typical computations in an IT2 FLS involve the following steps.

- 1) Compute the membership interval of x'_i on each X_i^n , $[\mu_{\underline{X}_i^n}(x'_i), \mu_{\bar{X}_i^n}(x'_i)]$, $i = 1, 2, \dots, I$, and $n = 1, 2, \dots, N$.
- 2) Compute the firing interval of the n th rule F^n :

$$\begin{aligned} F^n &= [\mu_{\underline{X}_1^n}(x'_1) \times \dots \times \mu_{\underline{X}_I^n}(x'_I), \\ &\quad \mu_{\bar{X}_1^n}(x'_1) \times \dots \times \mu_{\bar{X}_I^n}(x'_I)] \\ &\equiv [\underline{f}^n, \bar{f}^n], \quad n = 1, \dots, N. \end{aligned} \quad (1)$$

Note that the *minimum t-norm* may also be used in (1). However, this paper focuses only on the *product t-norm*.

- 3) Perform TR to combine F^n and the corresponding rule consequents. There are many such methods [33]. The most commonly used one is the center-of-sets type-reducer [33]:

$$\begin{aligned} Y_{\text{cos}} &= \frac{\sum_{n=1}^N Y^n F^n}{\sum_{n=1}^N F^n} = \bigcup_{\substack{y^n \in Y^n \\ f^n \in F^n}} \frac{\sum_{n=1}^N y^n f^n}{\sum_{n=1}^N f^n} \\ &= [y_l, y_r] \end{aligned} \quad (2)$$

where

$$y_l = \min_{k \in [1, N-1]} \frac{\sum_{n=1}^k \underline{y}^n \bar{f}^n + \sum_{n=k+1}^N \underline{y}^n \underline{f}^n}{\sum_{n=1}^k \bar{f}^n + \sum_{n=k+1}^N \underline{f}^n} \quad (3)$$

$$y_r = \max_{k \in [1, N-1]} \frac{\sum_{n=1}^k \bar{y}^n \underline{f}^n + \sum_{n=k+1}^N \bar{y}^n \bar{f}^n}{\sum_{n=1}^k \underline{f}^n + \sum_{n=k+1}^N \bar{f}^n}. \quad (4)$$

³The rule consequents can be IT2 FSs; however, when the popular center-of-sets TR method [33] is used, these consequent IT2 FSs are replaced by their centroids in the computation; therefore, it is more convenient to represent the rule consequents as intervals directly.

In (3) and (4), k is a possible *switch point*. In an exhaustive search approach, all k in $[1, N - 1]$ need to be evaluated until the true switch point is identified. Fortunately, y_l and y_r can also be computed more efficiently by the KM algorithms introduced in the next section, or their many enhanced versions introduced in Section III.

4) Compute the defuzzified output as

$$y = \frac{y_l + y_r}{2}. \quad (5)$$

C. Karnik–Mendel Algorithms

The KM algorithms consist of two parts, one for computing y_l in (3) and the other for computing y_r in (4). Define

$$f_l(k) = \frac{\sum_{n=1}^k \underline{y}^n \bar{f}^n + \sum_{n=k+1}^N \underline{y}^n \underline{f}^n}{\sum_{n=1}^k \bar{f}^n + \sum_{n=k+1}^N \underline{f}^n} \quad (6)$$

$$f_r(k) = \frac{\sum_{n=1}^k \bar{y}^n \underline{f}^n + \sum_{n=k+1}^N \bar{y}^n \bar{f}^n}{\sum_{n=1}^k \underline{f}^n + \sum_{n=k+1}^N \bar{f}^n} \quad (7)$$

where k is an integer in $[1, N - 1]$, and $\{\underline{y}^n\}$ and $\{\bar{y}^n\}$ have been sorted in ascending order, respectively. Furthermore, in this paper, it is assumed that $\{\underline{y}^n\}$ ($\{\bar{y}^n\}$) has no duplicate elements, which can be easily achieved by combining the weights for duplicate elements. Then, y_l in (3) and y_r in (4) can be reexpressed as [33]

$$\begin{aligned} y_l &= \min_{k \in [1, N-1]} f_l(k) \equiv f_l(L) \\ &= \frac{\sum_{n=1}^L \underline{y}^n \bar{f}^n + \sum_{n=L+1}^N \underline{y}^n \underline{f}^n}{\sum_{n=1}^L \bar{f}^n + \sum_{n=L+1}^N \underline{f}^n} \\ y_r &= \max_{k \in [1, N-1]} f_r(k) \equiv f_r(R) \\ &= \frac{\sum_{n=1}^R \bar{y}^n \underline{f}^n + \sum_{n=R+1}^N \bar{y}^n \bar{f}^n}{\sum_{n=1}^R \underline{f}^n + \sum_{n=R+1}^N \bar{f}^n} \end{aligned}$$

where L and R are *switch points* satisfying

$$\begin{aligned} \underline{y}^L &\leq y_l < \underline{y}^{L+1} \\ \bar{y}^R &< y_r \leq \bar{y}^{R+1}. \end{aligned}$$

Note that in [33], the two aforementioned inequalities are

$$\underline{y}^L \leq y_l \leq \underline{y}^{L+1} \quad (8)$$

$$\bar{y}^R \leq y_r \leq \bar{y}^{R+1} \quad (9)$$

because there $\{\underline{y}^n\}$ and $\{\bar{y}^n\}$ can have duplicate elements. When duplicate elements are combined, it follows that $\underline{y}^L < \underline{y}^{L+1}$ for $\forall L \in [1, N - 1]$ and $\bar{y}^R < \bar{y}^{R+1}$ for $\forall R \in [1, N - 1]$, and hence the two equalities in (8) or (9) cannot be satisfied simultaneously.

The KM algorithm to compute y_l and y_r is given in Table I. The main idea 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_{\cos} . Since \underline{y}^n increases from the left to the right along the horizontal axis of Fig. 3(a), we should choose a large weight (upper bound of the firing interval) for \underline{y}^n on the left and a small weight (lower bound of

TABLE I
KM ALGORITHMS

Step	For computing y_l	For computing y_r
1.	Initialize $f^n = \frac{\underline{f}^n + \bar{f}^n}{2}$ and compute $y = \frac{\sum_{n=1}^N \underline{y}^n f^n}{\sum_{n=1}^N f^n}$	Initialize $f^n = \frac{\underline{f}^n + \bar{f}^n}{2}$ and compute $y = \frac{\sum_{n=1}^N \bar{y}^n f^n}{\sum_{n=1}^N f^n}$
2.	Find $l \in [1, N - 1]$ s.t. $\underline{y}^l < y \leq \underline{y}^{l+1}$	Find $r \in [1, N - 1]$ s.t. $\bar{y}^r < y \leq \bar{y}^{r+1}$
3.	Set $f^n = \begin{cases} \bar{f}^n, & n \leq l \\ \underline{f}^n, & n > l \end{cases}$ and compute $y' = \frac{\sum_{n=1}^N \underline{y}^n f^n}{\sum_{n=1}^N f^n}$	Set $f^n = \begin{cases} \underline{f}^n, & n \leq r \\ \bar{f}^n, & n > r \end{cases}$ and compute $y' = \frac{\sum_{n=1}^N \bar{y}^n f^n}{\sum_{n=1}^N f^n}$
4.	If $y' = y$, stop and set $y_l = y$ and $L = l$; otherwise, set $y = y'$ and go to Step 2.	If $y' = y$, stop and set $y_r = y$ and $R = r$; otherwise, set $y = y'$ and go to Step 2.

Note that $\{\underline{y}^n\}_{n=1, \dots, N}$ and $\{\bar{y}^n\}_{n=1, \dots, N}$ have been sorted in ascending order, respectively.

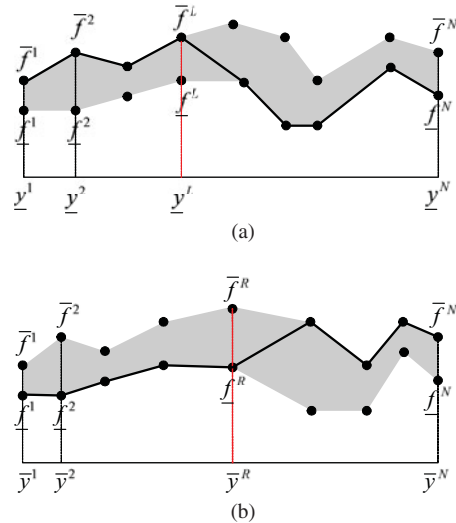


Fig. 3. Switch points in computing y_l and y_r . (a) Computing y_l : Switch from the upper bounds of the firing intervals to the lower bounds. (b) Computing y_r : Switch from the lower bounds of the firing intervals to the upper bounds.

the firing interval) for \underline{y}^n on the right. The KM algorithm for y_l finds the switch point L . For $n \leq L$, the upper bounds of the firing intervals are used to calculate y_l ; for $n > L$, the lower bounds are used. This ensures y_l is the minimum.

III. ENHANCEMENTS TO THE KARNIK–MENDEL ALGORITHMS

Five enhancements to the KM algorithms are introduced in this section. Their computational costs are also compared with the original KM algorithms. All of them require that $\{\underline{y}^n\}_{n=1, \dots, N}$ and $\{\bar{y}^n\}_{n=1, \dots, N}$ are sorted in ascending order, respectively.

TABLE II
EKM ALGORITHMS

Step	For computing y_l	For computing y_r
1.	Set $l = \lfloor N/2.4 \rfloor$ (the nearest integer to $N/2.4$), and compute $a = \sum_{n=1}^l y^n \bar{f}^n + \sum_{n=l+1}^N y^n \underline{f}^n$ $b = \sum_{n=1}^l \bar{f}^n + \sum_{n=l+1}^N \underline{f}^n$ $y = a/b$	Set $r = \lfloor N/1.7 \rfloor$ (the nearest integer to $N/1.7$), and compute $a = \sum_{n=1}^r \bar{y}^n \bar{f}^n + \sum_{n=r+1}^N \bar{y}^n \underline{f}^n$ $b = \sum_{n=1}^r \bar{f}^n + \sum_{n=r+1}^N \underline{f}^n$ $y = a/b$
2.	Find $l' \in [1, N-1]$ such that $\underline{y}^{l'} < y \leq \underline{y}^{l'+1}$	Find $r' \in [1, N-1]$ such that $\bar{y}^{r'} < y \leq \bar{y}^{r'+1}$
3.	If $l' = l$, stop and set $y_l = y$ and $L = l$; otherwise, continue.	If $r' = r$, stop and set $y_r = y$ and $R = r$; otherwise, continue.
4.	Compute $s = \text{sign}(l' - l)$, and $a' = a + s \sum_{n=\min(l,l')+1}^{\max(l,l')} y^n (\bar{f}^n - \underline{f}^n)$ $b' = b + s \sum_{n=\min(l,l')+1}^{\max(l,l')} (\bar{f}^n - \underline{f}^n)$ $y' = a'/b'$	Compute $s = \text{sign}(r' - r)$, and $a' = a - s \sum_{n=\min(r,r')+1}^{\max(r,r')} \bar{y}^n (\bar{f}^n - \underline{f}^n)$ $b' = b - s \sum_{n=\min(r,r')+1}^{\max(r,r')} (\bar{f}^n - \underline{f}^n)$ $y' = a'/b'$
5.	Set $y = y'$, $a = a'$, $b = b'$ and $l = l'$. Go to Step 2.	Set $y = y'$, $a = a'$, $b = b'$ and $r = r'$. Go to Step 2.

Note that $\{y^n\}_{n=1,\dots,N}$ and $\{\bar{y}^n\}_{n=1,\dots,N}$ have been sorted in ascending order, respectively.

A. Enhanced Karnik–Mendel Algorithms

The enhanced KM (EKM) algorithms [49], [50] are the earliest enhancement to the original KM algorithms. They have three improvements over the KM algorithms. 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. The detailed algorithms are given in Table II.

B. Enhanced Karnik–Mendel Algorithm With New Initialization

Yeh *et al.* [64] proposed an enhanced Karnik–Mendel algorithm with new initialization (EKMANI) to compute the generalized centroid of general T2 FSSs [33]. It is based on the observation that for two α -planes [29] close to each other, the centroids of the resulting two IT2 FSSs are also close to each other. Therefore, it may be advantageous to use the switch points obtained from the previous α -plane to initialize the switch points in the current α -plane (the similar idea is also used in [63]). Although the EKMANI was primarily designed to compute the generalized centroid, it may also be used in the TR of IT2 FLSs, because, usually, the output of an IT2 FLS changes only a small amount in each step.

The EKMANI for computing y_l is identical to the EKM algorithm for computing y_l , except that in Step (1), if there is a switch point obtained from previous computation, then set l to it; otherwise, set $l = \lfloor N/2.4 \rfloor$.

The EKMANI for computing y_r is identical to the EKM algorithm for computing y_r , except that in Step (1), if there is a switch point obtained from previous computation, then set r to it; otherwise, set $r = \lfloor N/1.7 \rfloor$.

C. Iterative Algorithm With Stop Condition

Melgarejo and his coauthors [11], [31] proposed two efficient algorithms to compute the generalized centroid of IT2 FSSs, which can also be used in TR of IT2 FLSs. The faster one, called iterative algorithm with stop condition (IASC) [11], is considered in this paper and presented in Table III. It is based on the fact [33] that $f_l(k)$ in (6) first monotonically decreases and then monotonically increases with the increase of k , and

TABLE III
IASC ALGORITHMS

Step	For computing y_l	For computing y_r
1.	Initialize $a = \sum_{n=1}^N y^n \underline{f}^n$ $b = \sum_{n=1}^N \underline{f}^n$ $y_l = y^N$ $l = 0$	Initialize $a = \sum_{n=1}^N \bar{y}^n \bar{f}^n$ $b = \sum_{n=1}^N \bar{f}^n$ $y_r = \bar{y}^1$ $r = 0$
2.	Compute $l = l + 1$ $a = a + y^l (\bar{f}^l - \underline{f}^l)$ $b = b + \bar{f}^l - \underline{f}^l$ $c = a/b$	Compute $r = r + 1$ $a = a - \bar{y}^r (\bar{f}^r - \underline{f}^r)$ $b = b - \bar{f}^r + \underline{f}^r$ $c = a/b$
3.	If $c > y_l$, set $L = l - 1$ and stop; otherwise, set $y_l = c$ and go to Step 2.	If $c < y_r$, set $R = r - 1$ and stop; otherwise, set $y_r = c$ and go to Step 2.

Note that $\{y^n\}_{n=1,\dots,N}$ and $\{\bar{y}^n\}_{n=1,\dots,N}$ have been sorted in ascending order, respectively.

$f_r(k)$ in (7) first monotonically increases and then monotonically decreases with the increase of k . Therefore, the IASC algorithms enumerate the switch point for y_l from 1 to $N-1$ until $f_l(k)$ stops decreasing, at which point y_l is obtained. Similarly, they enumerate the switch point for y_r from 1 to $N-1$ until $f_r(k)$ stops increasing, at which point y_r is obtained.

D. Enhanced Iterative Algorithm With Stop Condition

The enhanced IASC (EIASC) [53], presented in Table IV, makes the following two improvements over the IASC.

- 1) New stopping criterion based on the fact that $f_l(k)$ in (6) satisfies

$$f_l(k) \begin{cases} \geq \underline{y}^k, & k \leq L \\ < \underline{y}^k, & k > L \end{cases} \quad (10)$$

and $f_r(k)$ in (7) satisfies

$$f_r(k) \begin{cases} > \bar{y}^k, & k \leq R \\ \leq \bar{y}^k, & k > R \end{cases} \quad (11)$$

which have been proved in [53].

- 2) Both the IASC to compute y_l and the IASC to compute 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 [50] that for a variety of scenarios,

TABLE IV
EIASC ALGORITHMS

Step	For computing y_l	For computing y_r
1.	Initialize $a = \sum_{n=1}^N \underline{y}^n \underline{f}^n$ $b = \sum_{n=1}^N \underline{f}^n$ $L = 0$	Initialize $a = \sum_{n=1}^N \bar{y}^n \underline{f}^n$ $b = \sum_{n=1}^N \underline{f}^n$ $R = N$
2.	Compute $L = L + 1$ $a = a + \underline{y}^L (\bar{f}^L - \underline{f}^L)$ $b = b + \bar{f}^L - \underline{f}^L$ $y_l = a/b$	Compute $a = a + \bar{y}^R (\bar{f}^R - \underline{f}^R)$ $b = b + \bar{f}^R - \underline{f}^R$ $y_r = a/b$ $R = R - 1$
3.	If $y_l \leq \underline{y}^{L+1}$, stop; otherwise, go to Step 2.	If $y_r \geq \bar{y}^R$, stop; otherwise, go to Step 2.

Note that $\{\underline{y}^n\}_{n=1,\dots,N}$ and $\{\bar{y}^n\}_{n=1,\dots,N}$ have been sorted in ascending order, respectively.

its switch point L is smaller than $N/2$; therefore, it is more efficient to search from $L = 1$ instead of $L = N - 1$. However, setting the initial switch point $R = 1$ may not be efficient for y_r , since it has been shown in [50] that generally its switch point $R > N/2$. Therefore, 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.

Clearly, based on (10) and (11), given an initialization of L (or R), one can easily tell whether it is on the left or right of the true switch point, and hence, the new initialization idea in the EKMANI can also be used in EIASC. We implemented the corresponding algorithm, where the user can specify an initial L (or R); otherwise, L is initialized to $\lceil N/2.4 \rceil$ (R to $\lfloor N/1.7 \rfloor$), the same as the EKM algorithm. Experiments showed that this new algorithm only outperforms the EIASC when $N > 1000$. Since for FLSs usually $N < 1000$, this new algorithm is not included in this paper.

E. Enhanced Opposite Direction Searching Algorithms

Hu *et al.* [21], [22] proposed two algorithms to speed up the KM algorithms. The faster ones, called the enhanced opposite direction search (EODS) algorithms, are considered in this paper and presented in Table V. The EODS algorithms are based on the fact that $\underline{y}^L \leq f_l(L) \leq \underline{y}^{L+1}$ and $\bar{y}^R \leq f_r(R) \leq \bar{y}^{R+1}$, where $f_l(k)$ and $f_r(k)$ are defined in (6) and (7), respectively, and L and R are the switch points for y_l and y_r . For each of y_l and y_r , the EODS algorithms iteratively compute a positive search process (see S_l in Table V) and a negative search process (see S_r in Table V). The corresponding switch point is obtained when S_l meets S_r .

The EODS algorithms in Table V use exactly the same idea as that introduced in [22]; however, the implementation is slightly different from that in [22] because we employed two improvements.

- 1) We change S_l and S_r from arrays in [22] to scalars to reduce the memory requirement, as well as to improve speed.
- 2) In Step (5), we simplify the computation by computing y_l and y_r from \underline{y}^m and \bar{y}^m , instead of \underline{y}^L and \bar{y}^R in [22].

TABLE V
EODS ALGORITHMS

Step	For computing y_l	For computing y_r
1.	Initialize $m = 2$, $n = N - 1$ and compute $S_l = (\underline{y}^m - \underline{y}^1) \bar{f}^1$ $S_r = (\underline{y}^N - \underline{y}^n) \underline{f}^N$ $F_l = \underline{f}^N$ $F_r = \bar{f}^1$	Initialize $m = 2$, $n = N - 1$ and compute $S_l = (\bar{y}^m - \bar{y}^1) \underline{f}^1$ $S_r = (\bar{y}^N - \bar{y}^n) \bar{f}^N$ $F_l = \underline{f}^1$ $F_r = \bar{f}^N$
2.	If $m = n$, then go to Step 4.	If $m = n$, then go to Step 4.
3.	If $S_l > S_r$, then $F_l = F_l + \underline{f}^n$ $n = n - 1$ $S_r = S_r + F_l(\underline{y}^{n+1} - \underline{y}^n)$ else $F_r = F_r + \bar{f}^m$ $m = m + 1$ $S_l = S_l + F_r(\underline{y}^m - \underline{y}^{m-1})$	If $S_l > S_r$, then $F_r = F_r + \bar{f}^n$ $n = n - 1$ $S_r = S_r + F_r(\bar{y}^{n+1} - \bar{y}^n)$ else $F_l = F_l + \underline{f}^m$ $m = m + 1$ $S_l = S_l + F_l(\bar{y}^m - \bar{y}^{m-1})$
4.	Go to Step 2. If $S_l \leq S_r$, then $L = m$ $F_r = F_r + \bar{f}^m$ else $L = m - 1$ $F_l = F_l + \underline{f}^m$	Go to Step 2. If $S_l \leq S_r$, then $R = m$ $F_l = F_l + \underline{f}^m$ else $R = m - 1$ $F_r = F_r + \bar{f}^m$
5.	$y_l = \underline{y}^m + \frac{S_r - S_l}{F_r + F_l}$	$y_r = \bar{y}^m + \frac{S_r - S_l}{F_r + F_l}$

Note that $\{\underline{y}^n\}_{n=1,\dots,N}$ and $\{\bar{y}^n\}_{n=1,\dots,N}$ have been sorted in ascending order, respectively. $N \geq 3$ is assumed.

We compared the original EODS algorithms and the ones in Table V. Our implementation was much faster.

Note that the EODS algorithms only work when $N \geq 3$. Therefore, $N = 1$ and $N = 2$ need to be considered separately. Fortunately, they rarely occur in practice and there are closed-form solutions for y_l and y_r for these two special cases.

Two experiments are presented next to compare the computational cost of all algorithms presented in this section. The platform was a Lenovo Thinkpad T500 laptop computer with Intel Core2 Duo CPU 8600@2.4G Hz and 3-GB memory, running Windows 7 Home Premium 32-bit and MATLAB R2009a. We focus on IT2 FLCs; however, the conclusion should also hold for many other applications of IT2 FLSs.

In the first experiment, we compute the control surfaces of IT2 FLCs using Gaussian and trapezoidal MFs, respectively. This demonstrates the overall computational cost of different algorithms because every input in the input domain is considered and all inputs have the same weight in performance evaluation. In the second experiment, we compare the performance of the five algorithms in IT2 FLC design using evolutionary algorithms, in which a step response is used to evaluate the IT2 FLCs. This demonstrates the practical computational cost of different algorithms because, in practice, different regions of the control surface have different firing frequencies, e.g., the center portion of the control surface is generally fired more often than the boundaries. The step response enables us to simulate this behavior.

F. Computational Cost Comparison: Control Surface Computation

First, two-input single-output IT2 PI FLCs using Gaussian MFs are considered. Each input domain consisted of M MFs,

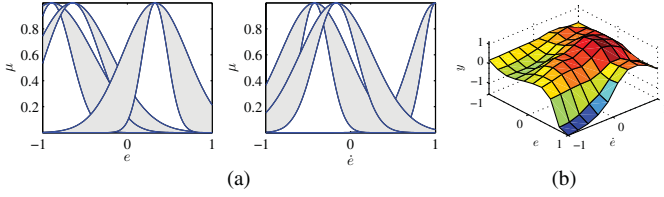


Fig. 4. Example of the generated Gaussian MFs (a) and the corresponding control surface (b).

and $M = \{2, 3, \dots, 10, 20, \dots, 50\}$ were used. The center of each MF was generated as a uniformly distributed random number in $[-1, 1]$ using MATLAB command $2*rand-1$. The uncertain standard deviations of each MF were generated as uniformly distributed random numbers in $[0.1, 0.5]$ using MATLAB command $0.1*sort(rand(1,2))+0.4$. The crisp consequent of each rule was generated as a uniformly distributed random number in $[-2, 2]$ using MATLAB command $4*rand-2$. Each input domain was discretized into ten points, and hence computing a complete control surface requires $10 \times 10 = 100$ TRs. An example of the generated MFs ($M = 3$) and the corresponding control surface are shown in Fig. 4.

To make the results statistically meaningful, we generated 100 random IT2 FLCs for each M and recorded the time that the five algorithms were used to perform these $100 \times 100 = 10\,000$ TRs. To compare the computational cost of IT2 FLCs with T1 FLCs, we also recorded the computation time for baseline T1 FLCs, whose MFs were the UMFs of the corresponding IT2 FSs. The results are shown in Fig. 5 and the first part of Table VI. Note that for fair comparison with the alternative TR methods in the next section, in Fig. 5 and Table VI, we include the time for computing the firing intervals of the rules,⁴ because some alternative TR algorithms may compute crisp firing strengths instead of firing intervals. Observe from Fig. 5 the following points.

- 1) All five enhanced algorithms are faster than the KM algorithms.
- 2) The EKM algorithms are faster than the EKMANI algorithms when $M > 6$.
- 3) Both the IASC and the EIASC algorithms are significantly faster than the KM, EKM, and EKMANI algorithms, especially when M is small ($M \leq 10$, i.e., the rulebase has no more than 100 rules). The EIASC algorithms also outperform the IASC algorithms slightly.
- 4) The EODS algorithms are the fastest when M is small ($M < 20$, i.e., the rulebase has less than 400 rules), which is the case for most practical IT2 FLCs.
- 5) Although the EODS algorithms are the fastest enhancement to the KM algorithms when $M < 20$, they are still about three times slower than the T1 FLC.

Next, we repeat the previous experiment using trapezoidal MFs. There are many different ways to generate trapezoidal IT2

⁴The comparisons in [53] did not include the time to compute the firing intervals of the rules; that is why the results in this paper may be different from those in [53]. However, the rankings of the algorithms are the same, because the time to compute the firing intervals of the rules is a constant for all methods presented in this section.

FSs. We used a simple method, as illustrated in Fig. 6 for three IT2 FSs in an input domain. The apexes of the UMFs were generated randomly under the constraint that for any point in the input domain, its firing levels on all UMFs add to 1. After the UMFs were generated, the LMF for each IT2 FS was also generated randomly with some constraints. Take the LMF of the middle IT2 FS as an example. e is a random number between a and b , f and g are two random numbers between b and c , i is a random number between c and d , and h is a random number in $[0, 1]$. An example of the actually generated trapezoidal MFs and the corresponding control surface are shown in Fig. 7, where $M = 3$ is used.

The experimental results for control surface computation using trapezoidal MFs are shown in Fig. 8 and the first part of Table VII. Observe the following points.

- 1) Except for the EKMANI algorithms, all other four enhancements are faster than the KM algorithms; however, the computational cost saving is not as large as that in Fig. 5. This is because at any time at most two trapezoidal IT2 FSs are fired in each input domain, and hence at most four rules are fired. Therefore, even though M may be a very large number, the actual N used in TR is always no larger than four (on the other hand, for Gaussian MFs, the actual N used in TR is always equal to M^2). As a result, all algorithms converge very quickly.
- 2) The EKMANI algorithms are slower than the EKM algorithms. When M becomes large (e.g., $M \geq 20$), they are even slower than the KM algorithms. This suggests that when trapezoidal IT2 FSs are used, it may not be advantageous to initialize the switch points from previous TR results.
- 3) The IASC and EIASC algorithms again have almost identical speed.
- 4) The EODS algorithms are the fastest enhancement for all M considered here. Again, this is because at any time at most two trapezoidal IT2 FSs are fired in each input domain, and hence at most four rules are fired. Therefore, even though M may be a very large number, the actual N used in TR is always no larger than four. The EODS algorithms are very fast for small N , which is also obvious from Fig. 5(b).
- 5) Although the EODS algorithms are the fastest enhancement to the KM algorithms, they are still about two to four times slower than the T1 FLC.

We can also observe from the first part of Tables VI and VII that the enhancements to the KM algorithms have larger computational cost saving over the KM algorithms when Gaussian IT2 FSs are used, and when the rulebase is small.

G. Computational Cost Comparison: Evolutionary Interval Type-2 Fuzzy Logic Controller Design

We also compare the computational cost of the KM algorithms and their five enhancements in IT2 FLC design using evolutionary computation, where the performance of a large number of (usually randomly generated) FLCs are evaluated. The following simple first-order plus dead-time plant is

TABLE VI
TOTAL COMPUTATION TIME (SECONDS) OF DIFFERENT TR ALGORITHMS WHEN GAUSSIAN IT2 FSS ARE USED

Category	Algorithm	M																	
		Control Surface Computation								Evolutionary IT2 FLC Design									
		2	4	6	8	10	20	30	40	50	2	4	6	8	10	20	30	40	50
I	KM	1.61	1.82	1.98	2.14	2.32	3.71	5.98	9.19	13.23	0.82	1.24	1.59	1.83	2.14	3.64	5.50	8.44	12.02
	EKM	1.25	1.53	1.63	1.71	1.86	2.97	4.88	7.60	11.07	0.80	1.11	1.39	1.60	1.87	3.12	4.80	7.25	10.37
	EKMANI	1.14	1.41	1.62	1.76	1.93	3.09	4.96	7.71	11.17	1.00	1.06	1.29	1.43	1.70	2.88	4.47	6.90	9.96
	IASC	0.55	0.57	0.64	0.72	0.83	1.84	3.50	5.89	9.04	0.55	0.61	0.74	0.81	1.04	2.12	3.77	6.02	9.10
	EIASC	0.52	0.56	0.62	0.70	0.82	1.81	3.45	5.82	8.96	0.54	0.59	0.71	0.79	1.01	2.06	3.65	5.87	8.89
	EODS	0.43	0.45	0.53	0.62	0.75	1.82	3.63	6.19	9.60	0.47	0.52	0.67	0.74	1.01	2.18	3.92	6.32	9.58
T1	0.15	0.15	0.18	0.20	0.24	0.53	1.00	1.63	2.49	0.17	0.18	0.23	0.25	0.33	0.66	1.11	1.70	2.49	
II	KM	2.03	1.84	1.97	2.13	2.35	3.82	6.10	9.21	13.28	0.91	1.16	1.58	1.80	1.98	3.42	5.68	8.50	11.74
	WM	0.91	0.83	0.89	0.97	1.09	1.95	3.36	5.36	7.98	0.74	0.83	0.99	1.14	1.37	2.24	3.78	5.62	7.59
	TTCC	0.57	0.55	0.62	0.72	0.84	1.82	3.43	5.71	8.76	0.56	0.65	0.79	0.94	1.16	2.26	4.09	6.49	9.60
	G	0.48	0.52	0.60	0.72	0.88	2.15	4.26	7.31	11.19	0.46	0.56	0.71	0.87	1.10	2.25	4.31	6.92	9.95
	CJ	0.54	0.51	0.60	0.72	0.87	2.14	4.24	7.24	11.18	0.45	0.55	0.69	0.85	1.08	2.25	4.36	7.10	10.27
	GCCJ	0.18	0.21	0.25	0.32	0.41	1.12	2.29	3.94	6.11	0.22	0.30	0.37	0.48	0.65	1.44	2.73	4.38	6.35
	LYZ	0.20	0.21	0.25	0.30	0.37	0.94	1.86	3.15	4.86	0.25	0.34	0.42	0.53	0.72	1.37	2.43	3.64	4.99
	BMM	0.22	0.22	0.25	0.30	0.37	0.88	1.69	2.86	4.39	0.31	0.31	0.40	0.49	0.61	1.17	2.09	3.21	4.46
	LM	0.21	0.22	0.25	0.30	0.36	0.85	1.66	2.80	4.31	0.24	0.30	0.38	0.47	0.58	1.11	2.02	3.09	4.35
	WT (NT)	0.18	0.19	0.23	0.27	0.33	0.84	1.64	2.77	4.26	0.21	0.28	0.34	0.43	0.57	1.11	2.02	3.08	4.31
	T1	0.15	0.16	0.18	0.20	0.24	0.53	0.98	1.61	2.46	0.17	0.20	0.23	0.28	0.35	0.66	1.16	1.76	2.49
III	KMf	1.61	1.82	2.00	2.14	2.32	3.70	5.92	9.07	13.21	0.93	1.22	1.53	1.87	2.10	3.55	5.42	8.15	12.07
	KMs	1.77	1.83	1.86	1.89	1.96	2.38	2.95	3.72	4.72	1.17	1.32	1.45	1.58	1.70	2.22	2.74	3.52	4.57
	EODSf	0.44	0.47	0.54	0.62	0.74	1.82	3.66	6.21	9.58	0.46	0.55	0.64	0.81	0.86	2.12	3.86	6.19	9.70
	EODSs	0.68	0.71	0.76	0.80	0.86	1.25	1.82	2.60	3.57	0.72	0.79	0.83	0.91	0.91	1.43	1.91	2.61	3.68
	WTf	0.17	0.19	0.23	0.27	0.33	0.84	1.69	2.83	4.29	0.21	0.27	0.33	0.43	0.42	1.09	1.84	2.83	4.41
	WTs	0.26	0.28	0.30	0.33	0.38	0.71	1.21	1.90	2.78	0.29	0.33	0.37	0.42	0.43	0.86	1.31	1.91	2.83
	T1	0.15	0.16	0.18	0.21	0.24	0.53	1.00	1.65	2.49	0.16	0.20	0.23	0.28	0.28	0.65	1.09	1.65	2.53

Note that the rulebase has M^2 rules.

employed as the nominal system [60]

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

The goal is to design an IT2 fuzzy PI controller

$$\dot{u} = k_P \dot{e} + k_I e \quad (12)$$

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.

First, assume that there are M Gaussian IT2 FSs in each domain (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 , $m = 1, 2, \dots, M$). Each of the M^2 rule consequents is represented by a crisp number y^n , $n = 1, \dots, M^2$. Then, for each input pair (e , \dot{e}), all $N = M^2$ rules are fired, and a TR algorithm is needed to compute the output of the IT2 FLC. The population consisted of 100 randomly generated IT2 FLCs (all m_m , σ_m^1 , σ_m^2 , and y^n were generated randomly), and the performance of each FLC was evaluated by a step response in the first 100 s with sampling frequency 1 Hz. We recorded the time that the five algorithms were used to perform these $100 \times 100 = 10\,000$ TRs. $M = \{2, 3, \dots, 10, 20, \dots, 50\}$ were used. The results are shown in Fig. 9 and the first part of Table VI. Observe the following points.

- 1) Generally, all five enhanced algorithms are faster than the KM algorithms.
- 2) The EKMANI algorithms are faster than the EKM algorithms when $M \geq 4$.
- 3) Both the IASC and the EIASC algorithms are significantly faster than the KM, EKM, and EKMANI algorithms, especially when M is small ($M \leq 20$, i.e., the rulebase has

no more than 400 rules). The EIASC algorithms also outperform the IASC algorithms slightly.

- 4) The EODS algorithms are the fastest when M is small ($M \leq 10$, i.e., the rulebase has no more than 100 rules), which is the case for most practical IT2 FLCs.
- 5) Although the EODS algorithms are the fastest enhancement to the KM algorithms when $M \leq 10$, they are still about three times slower than the T1 FLC.

We then repeated the experiment using trapezoidal MFs. The trapezoidal MFs were generated in the same way as those in the previous section. The results are shown in Fig. 10 and the first part of Table VII. We have the same observations as those from Fig. 8.

We can again observe from the first part of Tables VI and VII that the enhancements to the KM algorithms have larger computational cost saving over the KM algorithms when Gaussian IT2 FSs are used.

H. Summary

In summary, among the KM algorithms and their five enhanced versions, the EODS algorithms seem to be the fastest to use when there are less than 100 rules fired each time, which is usually true in practice. However, they are much more complex than the EIASC algorithms, which are only slightly (≤ 1.2 times) slower than the EODS algorithms for practical IT2 FLCs. For the point of ease in understanding and implementation, the EIASC algorithms may be preferred in practice.

IV. ALTERNATIVE TYPE-REDUCTION ALGORITHMS

As the iterative KM algorithms have high computational cost, as well as their iterative nature makes them difficult to

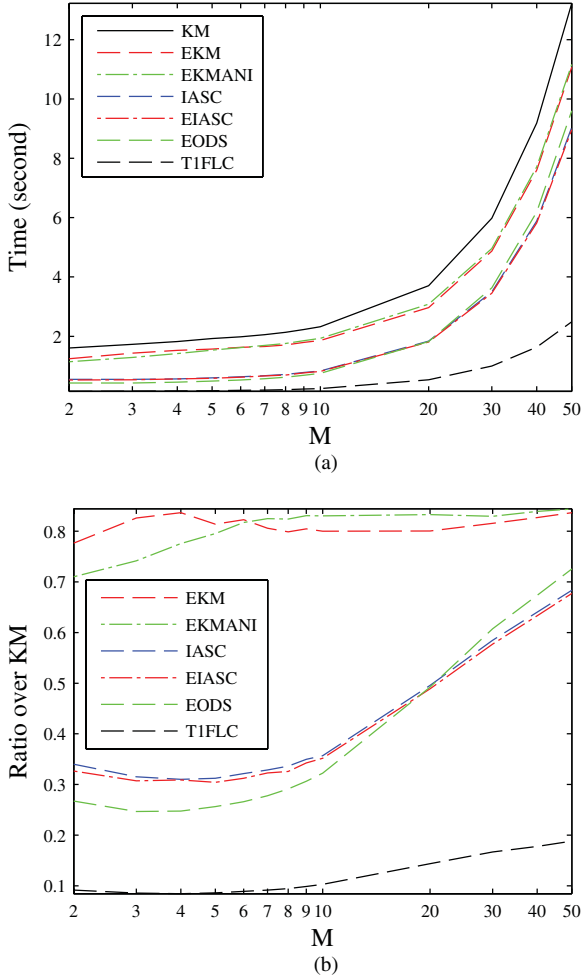


Fig. 5. Computational cost of the KM algorithms and their enhanced versions in control surface computation using Gaussian MFs. (a) Total computation time of the 100 control surfaces for different M (the number of IT2 FSs in each input domain). Note that $N = M^2$. (b) Ratio of the computation time of the enhanced algorithms to the KM algorithms. To facilitate reading, the order of the algorithms in the legend is ranked roughly according to their performance.

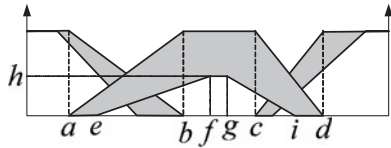


Fig. 6. Trapezoidal IT2 FSs used in the experiments.

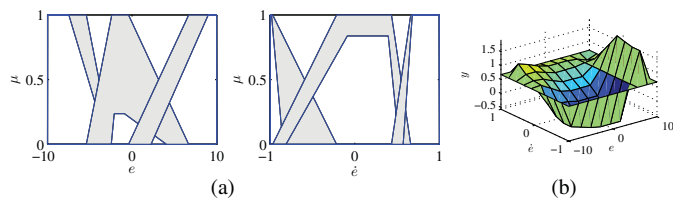


Fig. 7. Example of the generated trapezoidal MFs (a) and the corresponding control surface (b).

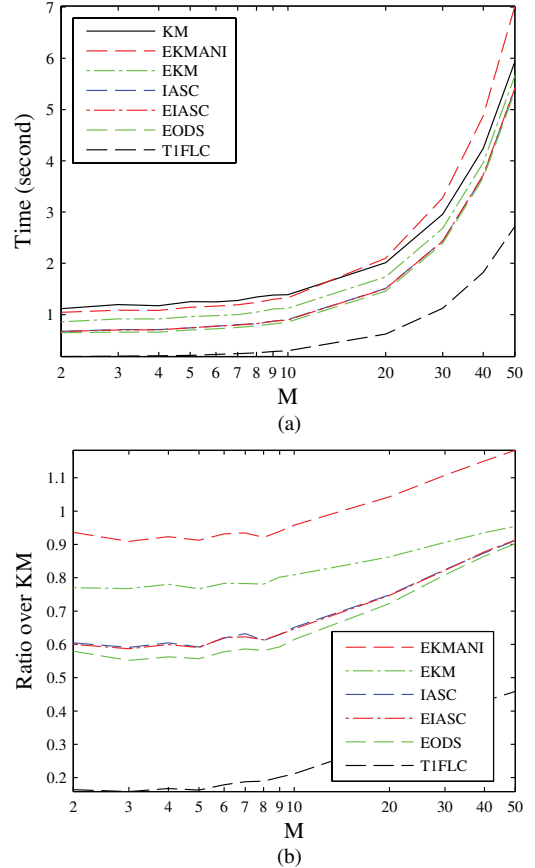


Fig. 8. Computational cost of the KM algorithms and their enhanced versions in control surface computation using trapezoidal MFs. (a) Total computation time of the 100 control surfaces for different M (the number of IT2 FSs in each input domain). Note that the rulebase has $N = M^2$ rules, but at any time, no more than four rules are fired. (b) Ratio of the computation time of the enhanced algorithms to the KM algorithms. To facilitate reading, the order of the algorithms in the legend is ranked roughly according to their performance.

analyze, people have proposed many alternative TR algorithms, which have closed-form expressions and are usually faster than the KM algorithms. Eleven of them [2], [8], [10], [13], [16], [27], [28], [37], [44], [56], [61] are introduced and compared in this section. They are presented in the chronological order.

A. Gorzalczany Method

Gorzalczany [14] proposed two defuzzification methods to obtain a number from the output of the Mamdani inference engine using interval-valued FSs. Since the focus of this paper is the TSK model, we adapt his methods to TSK models. Note that both of his methods only apply to $\underline{y}^n = \bar{y}^n \equiv y^n$ and $n = 1, 2, \dots, N$.

Given y^n and the firing intervals of the rules, $[f^n, \bar{f}^n]$, $n = 1, 2, \dots, N$, first we construct a polygon shown in Fig. 11, which can be viewed as a special IT2 FS. For each point in $[y^1, y^N]$,

TABLE VII
TOTAL COMPUTATION TIME (SECONDS) OF DIFFERENT TR ALGORITHMS WHEN TRAPEZOIDAL IT2 FSS ARE USED

Category	Algorithm	M																	
		Control Surface Computation								Evolutionary IT2 FLC Design									
		2	4	6	8	10	20	30	40	50	2	4	6	8	10	20	30	40	50
I	KM	1.12	1.17	1.25	1.34	1.39	2.01	2.96	4.24	5.93	1.01	1.18	1.25	1.30	1.46	2.03	2.97	4.28	6.01
	EKM	0.86	0.91	0.98	1.05	1.12	1.74	2.68	3.97	5.66	0.82	0.92	0.98	1.04	1.17	1.76	2.70	4.01	5.71
	EKMANI	1.04	1.08	1.16	1.24	1.33	2.10	3.28	4.88	7.02	1.00	1.08	1.16	1.23	1.38	2.13	3.32	4.93	7.11
	IASC	0.67	0.71	0.77	0.82	0.90	1.51	2.44	3.71	5.41	0.68	0.72	0.78	0.84	0.95	1.52	2.47	3.77	5.44
	EIASC	0.67	0.70	0.77	0.82	0.90	1.50	2.43	3.72	5.42	0.68	0.71	0.78	0.83	0.95	1.52	2.46	3.74	5.44
	EODS	0.65	0.66	0.72	0.78	0.85	1.46	2.39	3.67	5.36	0.68	0.66	0.72	0.78	0.89	1.48	2.41	3.68	5.38
T1	0.18	0.20	0.22	0.25	0.29	0.62	1.12	1.82	2.72	0.18	0.20	0.22	0.25	0.29	0.61	1.12	1.83	2.74	
II	KM	1.13	1.27	1.26	1.34	1.50	1.99	2.96	4.25	5.97	0.99	1.24	1.29	1.32	1.42	2.00	3.00	4.28	6.02
	WM	0.88	0.92	0.99	1.07	1.19	1.85	3.07	4.65	6.77	0.92	0.95	1.01	1.06	1.16	1.86	3.08	4.67	6.76
	TTCC	0.72	0.75	0.81	0.89	1.01	1.73	2.83	4.35	6.37	0.70	0.75	0.81	0.89	1.00	1.75	2.88	4.39	6.36
	G	0.63	0.67	0.72	0.77	0.88	1.40	2.31	3.57	5.16	0.70	0.75	0.80	0.86	0.94	1.51	2.49	3.76	5.38
	CJ	0.63	0.66	0.71	0.76	0.86	1.39	2.30	3.54	5.16	0.68	0.73	0.77	0.84	0.92	1.48	2.45	3.72	5.34
	GCCJ	0.33	0.35	0.39	0.43	0.53	0.96	1.73	2.82	4.15	0.33	0.34	0.38	0.42	0.51	0.96	1.76	2.82	4.14
	LYZ	0.41	0.42	0.47	0.51	0.60	1.09	1.93	3.06	4.76	0.39	0.42	0.46	0.51	0.59	1.07	1.96	3.09	4.54
	BMM	0.45	0.48	0.52	0.56	0.65	1.13	1.94	3.06	4.47	0.45	0.48	0.52	0.55	0.64	1.11	1.96	3.04	4.45
	LM	0.40	0.43	0.48	0.52	0.61	1.06	1.87	2.95	4.36	0.40	0.42	0.47	0.51	0.60	1.06	1.89	2.97	4.35
	WT (NT)	0.32	0.33	0.38	0.42	0.51	0.96	1.76	2.79	4.17	0.32	0.34	0.38	0.42	0.51	0.96	1.77	2.82	4.17
	T1	0.21	0.22	0.25	0.28	0.33	0.64	1.14	1.85	2.74	0.21	0.22	0.25	0.28	0.34	0.64	1.15	1.85	2.76
III	KMf	1.20	1.15	1.32	1.29	1.44	2.07	2.95	4.24	5.96	1.01	1.15	1.26	1.31	1.43	2.04	2.96	4.25	5.98
	KMs	0.99	0.81	0.62	0.60	0.65	0.96	1.49	2.22	3.19	0.86	0.78	0.67	0.61	0.65	0.95	1.47	2.22	3.18
	EODSf	0.67	0.66	0.74	0.78	0.87	1.45	2.41	3.67	5.35	0.67	0.67	0.73	0.78	0.87	1.48	2.39	3.66	5.38
	EODSs	0.89	0.66	0.61	0.59	0.65	0.95	1.50	2.21	3.19	0.79	0.66	0.62	0.59	0.64	0.95	1.47	2.24	3.19
	WTf	0.31	0.33	0.38	0.41	0.50	0.97	1.75	2.79	4.14	0.31	0.33	0.38	0.42	0.49	0.97	1.75	2.80	4.19
	WTs	0.38	0.30	0.30	0.31	0.36	0.67	1.18	1.86	2.77	0.34	0.29	0.30	0.31	0.35	0.66	1.16	1.91	2.78
	T1	0.21	0.21	0.25	0.27	0.33	0.64	1.15	1.84	2.73	0.20	0.22	0.24	0.27	0.32	0.64	1.14	1.85	2.76

Note that the rulebase has M^2 rules.

we compute

$$\mu(y) = \frac{(\underline{f} + \bar{f})}{2} \cdot [1 - (\bar{f} - \underline{f})] \quad (13)$$

where $\bar{f} - \underline{f}$ is called the *bandwidth*. Then, the defuzzified output can be computed as

$$y_G = \arg \max_y \mu(y). \quad (14)$$

Gorzalczany [14] explained that (14) provides an element y_G , which most adequately satisfies the compromise between the maximization of the mean value and the minimization of the bandwidth of the inference engine output. He also pointed out that (14) yielded a constant error in his FLC. Therefore, he proposed another method to prevent such a situation, where the defuzzified output is chosen as the point that divides in half the region under the curve $\mu(y)$, i.e., y_G is the solution to the following equation:

$$\int_{y^1}^{y_G} \mu(y) dy = \int_{y_G}^{y^N} \mu(y) dy. \quad (15)$$

Gorzalczany did not point out how to efficiently compute y_G in (15). However, if we form a granule from $\mu(y)$, as shown in Fig. 12, then y_G in (15) is its centroid. Coupland and John's method for computing the geometric centroid of an IT2 FS, introduced in Section IV-E, can be used for this purpose, and it is used in our experiment.

B. Liang–Mendel Unnormalized Method

Liang and Mendel [28] proposed an *unnormalized* TR method, in which the defuzzified output is still computed by

(5), but

$$y_l = \sum_{n=1}^N \underline{f}^n y^n, \quad y_r = \sum_{n=1}^N \bar{f}^n y^n$$

where $y^n = \bar{y}^n \equiv y_n$ and $\{y^n\}$ do not need to be sorted. This method is called *unnormalized* because neither y_l nor y_r is normalized by the sum of the firing levels. In the literature and practice, most FLSs use normalized defuzzification.

C. Wu–Mendel Uncertainty Bound Method

The uncertainty bound method, proposed by Wu and Mendel [61], computes the output of the IT2 FLS by (5), but

$$y_l = \frac{y_l + \bar{y}_l}{2}, \quad y_r = \frac{y_r + \bar{y}_r}{2}$$

where

$$\begin{aligned} \bar{y}_l &= \min\{\underline{y}^{(0)}, \underline{y}^{(N)}\}, \quad \underline{y}_r = \max\{\bar{y}^{(0)}, \bar{y}^{(N)}\} \\ \bar{y}_l &= \bar{y}_l - \frac{\sum_{n=1}^N (\bar{f}^n - \underline{f}^n)}{\sum_{n=1}^N \bar{f}^n \sum_{n=1}^N \underline{f}^n} \\ &\quad \times \frac{\sum_{n=1}^N \underline{f}^n (\underline{y}^n - \underline{y}^1) \sum_{n=1}^N \bar{f}^n (\underline{y}^N - \underline{y}^n)}{\sum_{n=1}^N \underline{f}^n (\underline{y}^n - \underline{y}^1) + \sum_{n=1}^N \bar{f}^n (\underline{y}^N - \underline{y}^n)} \\ \bar{y}_r &= \underline{y}_r + \frac{\sum_{n=1}^N (\bar{f}^n - \underline{f}^n)}{\sum_{n=1}^N \bar{f}^n \sum_{n=1}^N \underline{f}^n} \\ &\quad \times \frac{\sum_{n=1}^N \bar{f}^n (\bar{y}^n - \bar{y}^1) \sum_{n=1}^N \underline{f}^n (\bar{y}^N - \bar{y}^n)}{\sum_{n=1}^N \bar{f}^n (\bar{y}^n - \bar{y}^1) + \sum_{n=1}^N \underline{f}^n (\bar{y}^N - \bar{y}^n)} \end{aligned}$$

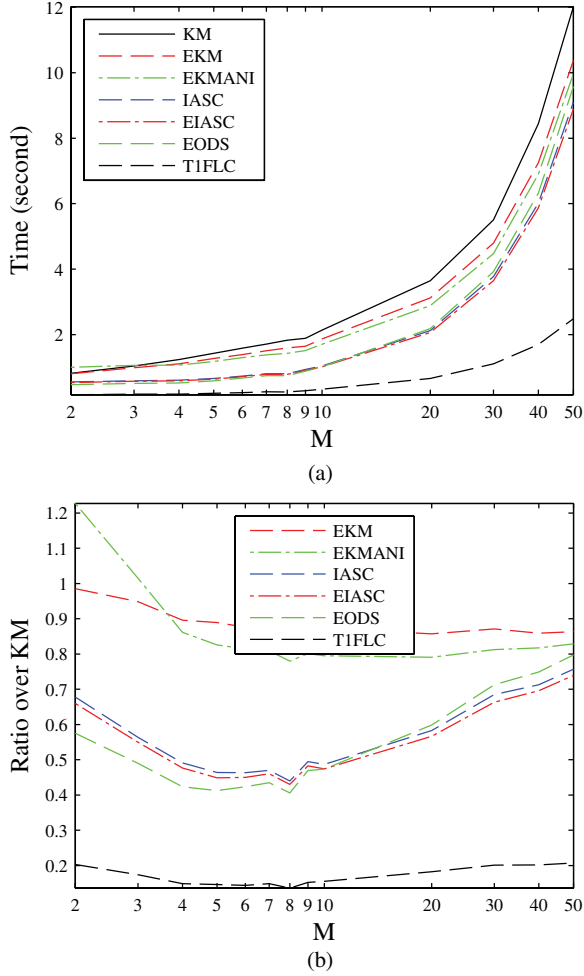


Fig. 9. Computational cost of the KM algorithms and their enhanced versions in evolutionary IT2 FLC design using Gaussian MFs. (a) Total computation time of the 100 IT2 FLCs for different M (the number of IT2 FSs in each input domain). Note that $N = M^2$. (b) Ratio of the computation time of the enhanced algorithms to the KM algorithms. To facilitate reading, the order of the algorithms in the legend is ranked roughly according to their performance.

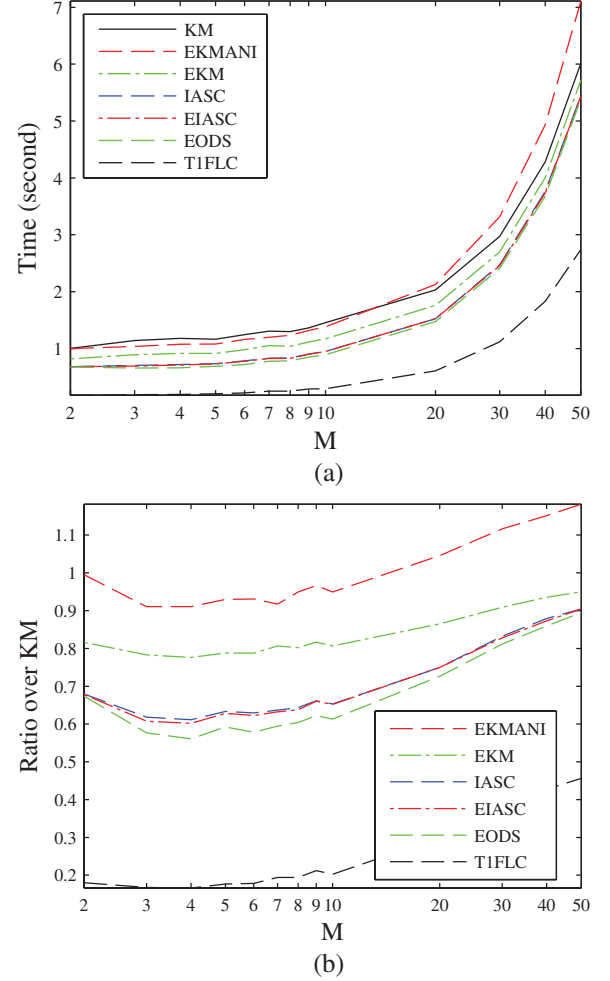


Fig. 10. Computational cost of the KM algorithms and their enhanced versions in evolutionary IT2 FLC design using trapezoidal MFs. (a) Total computation time of the 100 IT2 FLCs for different M (the number of IT2 FSs in each input domain). Note that the rulebase has $N = M^2$ rules, but at any time, no more than four rules are fired. (b) Ratio of the computation time of the enhanced algorithms to the KM algorithms. To facilitate reading, the order of the algorithms in the legend is ranked roughly according to their performance.

in which

$$\underline{y}^{(0)} = \frac{\sum_{n=1}^N \underline{y}^n \underline{f}^n}{\sum_{n=1}^N \underline{f}^n}, \quad \underline{y}^{(N)} = \frac{\sum_{n=1}^N \underline{y}^n \bar{f}^n}{\sum_{n=1}^N \bar{f}^n}$$

$$\bar{y}^{(0)} = \frac{\sum_{n=1}^N \bar{y}^n \underline{f}^n}{\sum_{n=1}^N \underline{f}^n}, \quad \bar{y}^{(N)} = \frac{\sum_{n=1}^N \bar{y}^n \bar{f}^n}{\sum_{n=1}^N \bar{f}^n}.$$

Unlike the KM algorithms, the uncertainty-bound method does not require $\{\underline{y}^n\}$ and $\{\bar{y}^n\}$ to be sorted, although it still needs to identify the minimum and maximum of $\{\underline{y}^n\}$ and $\{\bar{y}^n\}$.

D. Wu-Tan Method

Wu and Tan [56] proposed a closed-form TR and defuzzification method by making use of the equivalent T1 membership grades [57]. The basic idea is to first find an equivalent T1 membership grade $\mu_{X_i^n}(x_i)$ to replace each firing interval

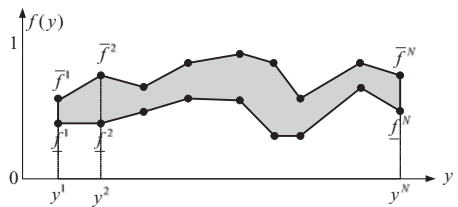


Fig. 11. Polygon used in Gorzalcany's method to compute $\mu(y)$ in (13).

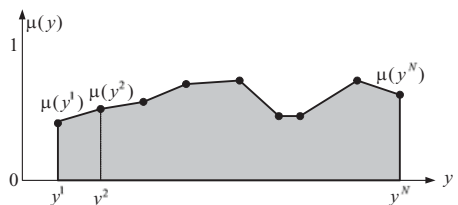


Fig. 12. Polygon used in computing y_G in (15).

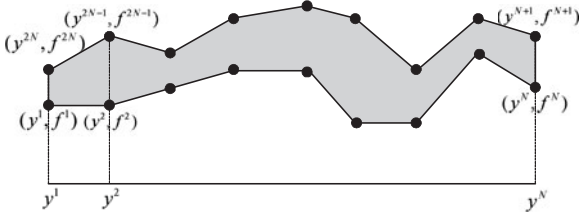


Fig. 13. Closed polygon used in the Coupland–John method.

$[\mu_{\underline{X}_i^n}(x_i), \mu_{\overline{X}_i^n}(x_i)]$, i.e.,

$$\mu_{X_i^n}(x_i) = \mu_{\overline{X}_i^n}(x_i) - h_i^n(\mathbf{x})[\mu_{\overline{X}_i^n}(x_i) - \mu_{\underline{X}_i^n}(x_i)]$$

where $h_i^n(\mathbf{x})$ is a function of the input \mathbf{x} , and is different for different IT2 FSs (In [56] $\mathbf{x} = (e, \dot{e})$ and $h_i^n(\mathbf{x}) = \alpha e + \beta \dot{e}$, where α and β change with i and n , and they were identified by Genetic Algorithms). This property is motivated by the *adaptiveness* of an IT2 FLC [47], which means that the embedded T1 FSs used to compute the bounds of the TR interval change as input changes.

Since the firing strengths of the rules become point (instead of interval) numbers f^n computed from these $\mu_{X_i^n}(x_i)$, the IT2 FLS becomes an adaptive T1 FLC, and its output is computed as

$$y = \frac{\sum_{n=1}^N y^n f^n}{\sum_{n=1}^N f^n}.$$

The Wu–Tan (WT) method also does not require $\{y^n\}$ to be sorted.

E. Coupland–John Geometric Method

Coupland and John [8] proposed a geometric method for TR and defuzzification of Mamdani IT2 FLSs. In this paper, we extend it to TSK IT2 FLSs.

The Coupland–John method constructs a closed polygon from y^n and the corresponding firing intervals, and relabels the boundary points as those shown in Fig. 13. For an IT2 FLS with N rules, there are $2N$ points on the boundary of the closed polygon, (y^n, f^n) , $n = 1, \dots, 2N$. Then, the centroid of the polygon is viewed as the defuzzification output of the IT2 FLS:

$$y = \frac{\sum_{n=1}^{2N} (y^n + y^{n+1})(y^n f^{n+1} - y^{n+1} f^n)}{3 \sum_{i=1}^{2N} (y^n f^{n+1} - y^{n+1} f^n)}$$

where (y^{2N+1}, f^{2N+1}) is the same as (y^1, f^1) . Observe that the geometric method requires $\{y^n\}_{n=1, \dots, N}$ to be sorted so that the closed polygon can be constructed.

F. Nie–Tan Method

Nie and Tan [37] proposed another closed-form TR and defuzzification method, where the output of an IT2 FLS is computed as

$$y = \frac{\sum_{n=1}^N y^n (f^n + \overline{f}^n)}{\sum_{n=1}^N (f^n + \overline{f}^n)}.$$

Observe that the Nie–Tan (NT) method does not require $\{y^n\}$ to be sorted, and it is a special case of the WT method when

$h_i^n(\mathbf{x}) = 0.5$. However, by specifying $h_i^n(\mathbf{x})$ to be a constant for all inputs, the resulting IT2 FLS loses adaptiveness, which is considered as one of the two fundamental differences between IT2 and T1 FLCs [47].

G. Begian–Melek–Mendel Method

Begian *et al.* [2] proposed another closed-form TR and defuzzification method for IT2 FLSs, i.e.,

$$y = \alpha \frac{\sum_{n=1}^N \underline{f}^n y^n}{\sum_{n=1}^N \underline{f}^n} + \beta \frac{\sum_{n=1}^N \overline{f}^n y^n}{\sum_{n=1}^N \overline{f}^n}. \quad (16)$$

where α and β are adjustable coefficients. Observe that it views the output of an IT2 FLS as a combination of the outputs of two T1 FLSs: one constructed only from the LMFs and the other constructed only from the UMFs.

The Begian–Melek–Mendel (BMM) method does not require $\{y^n\}$ to be sorted. It is also similar to Niewiadomski *et al.*'s [39] fourth method for TR of IT2 FSs; however, Niewiadomski *et al.* have not extended their methods to IT2 FLSs.

The BMM method requires $y^n = \overline{y}^n \equiv y^n$. Li *et al.* [26] extended it to the case that $\underline{y}^n \neq \overline{y}^n$, i.e.,

$$y = \alpha \frac{\sum_{n=1}^N \underline{f}^n \underline{y}^n}{\sum_{n=1}^N \underline{f}^n} + \beta \frac{\sum_{n=1}^N \overline{f}^n \overline{y}^n}{\sum_{n=1}^N \overline{f}^n}. \quad (17)$$

Since (17) and (16) have the same computational cost, only the BMM method is considered in this paper.

Note that Castillo *et al.* [7] also proposed a new defuzzification method, in which the output of an IT2 FLS is approximated by the average of two T1 FLSs, which are tuned by evolutionary algorithms. Once the two T1 FLSs are obtained, the computational cost of Castillo *et al.*'s approach is similar to the BMM method; therefore, it is not considered separately in this paper.

H. Greenfield–Chiclana–Coupland–John Collapsing Method

Greenfield *et al.* [16] proposed a collapsing method for TR of IT2 FLSs, where each IT2 FS is replaced by a representative embedded T1 FS whose membership grades are computed recursively. To simplify the computation, the representative embedded T1 FS can be approximated by a pseudo representative embedded T1 FS

$$\mu_X(x) = \frac{\mu_{\underline{X}}(x) + \mu_{\overline{X}}(x)}{2}.$$

Once all IT2 FSs in an IT2 FLS are replaced by their pseudo representative embedded T1 FSs, the IT2 FLS is reduced to a T1 FLS, and the defuzzification is straightforward. Again, the collapsing method does not require $\{y^n\}$ to be sorted.

Observe that the collapsing method looks very similar to the NT method. In fact, when there is only one input, these two methods are identical; however, they are different when there are more than one input, because the firing level of \tilde{R}^n (see Section II-B) in the Greenfield–Chiclana–Coupland–John (GCCJ) method is

$$f_{GCCJ}^n = \prod_{i=1}^I \frac{\mu_{\underline{X}_i^n}(x_i) + \mu_{\overline{X}_i^n}(x_i)}{2}$$

whereas the firing level of \tilde{R}^n in the NT method is

$$f_{\text{NT}}^n = \frac{\prod_{i=1}^I \mu_{X_i^n}(x_i) + \prod_{i=1}^I \mu_{\bar{X}_i^n}(x_i)}{2}.$$

Greenfield *et al.* [15] also proposed a sampling method for TR, where only a relatively small random sample of the totality of embedded T1 FSs is processed. Because its output is not deterministic, it is not considered in this paper.

I. Li–Yi–Zhao Method

Li *et al.* [27] proposed a new TR method based on interval analysis without considering the dependence of f^n in the numerator and denominator of (2). They still computed the output of the IT2 FLS by (5), but

$$y_l = \min \left[\frac{\sum_{n=1}^N \min(\underline{f}^n \underline{y}^n, \bar{F}^n \underline{y}^n)}{\sum_{n=1}^N \underline{f}^n}, \frac{\sum_{n=1}^N \min(\underline{f}^n \bar{y}^n, \bar{F}^n \bar{y}^n)}{\sum_{n=1}^N \bar{F}^n} \right]$$

$$y_r = \max \left[\frac{\sum_{n=1}^N \max(\underline{f}^n \bar{y}^n, \bar{F}^n \bar{y}^n)}{\sum_{n=1}^N \underline{f}^n}, \frac{\sum_{n=1}^N \max(\underline{f}^n \underline{y}^n, \bar{F}^n \underline{y}^n)}{\sum_{n=1}^N \bar{F}^n} \right].$$

The Li–Yi–Zhao (LYZ) method does not require $\{y^n\}$ to be sorted.

Li *et al.* [27] also showed that the $[y_l, y_r]$ computed from the KM algorithms is a subset of the $[y_l, y_r]$ computed earlier, and the absolute value of the difference between the defuzzified output computed by the KM algorithms and their defuzzified output is upper bounded by $\max(\max_n |y^n|, \max_n |\bar{y}^n|) \cdot \frac{\sum_{n=1}^N (\bar{F}^n - \underline{f}^n)}{\sum_{n=1}^N \underline{f}^n}$. However, this is a very loose bound, especially when $\sum_{n=1}^N \underline{f}^n$ approaches 0. Consider a simple example, in which only two rules are fired, and $[\underline{f}^1, \bar{F}^1] = [0.001, 0.2]$, $[\underline{y}^1, \bar{y}^1] = [0.5, 0.6]$, $[\underline{f}^2, \bar{F}^2] = [0, 0.8]$, and $[\underline{y}^2, \bar{y}^2] = [0.9, 1]$. The bounds are computed to be 999, which is 999 times of the maximum consequent \bar{y}^2 . In fact, $[y_l, y_r]$ computed by the KM algorithms is $[0.5, 0.9995]$, whereas $[y_l, y_r]$ computed by the LYZ method is $[0.0005, 920]$. Clearly, the difference is huge.

J. Du–Ying Method

Du and Ying [10] proposed an average defuzzifier. It first computes 2^N crisp outputs obtained by all possible combinations of the lower and upper firing levels, i.e.,

$$y_m = \frac{\sum_{n=1}^N y^n f^{n*}}{\sum_{n=1}^N f^{n*}}, \quad m = 1, 2, \dots, 2^N$$

where $f^{n*} \in \{\underline{f}^n, \bar{F}^n\}$. The final defuzzified output is then computed as the average of all these 2^N y_m , i.e.,

$$y = \frac{1}{2^N} \sum_{m=1}^{2^N} y_m.$$

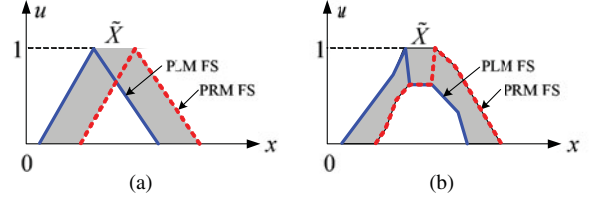


Fig. 14. (a) Possible-left-most and possible-right-most embedded T1 FSs used in [44]. (b) Possible-left-most and possible-right-most embedded T1 FSs for an arbitrary FOU.

The DY method does not require $\{y^n\}$ to be sorted.

Although the DY method makes the analysis of the resulting IT2 FLS easier, it has higher computational cost than the KM type-reducer. Additionally, its computational cost increases exponentially with the number of rules since there are 2^N T1 FLSs to be computed.

K. Tao–Taur–Chang–Chang Method

Tao *et al.* [44] proposed a simplified IT2 FLS, whose output is computed as

$$y = \alpha y_{\text{P.L.M}} + (1 - \alpha) y_{\text{P.R.M}}$$

where $y_{\text{P.L.M}}$ is the output of a T1 FLS constructed only from the *possible-left-most* embedded T1 FSs, and $y_{\text{P.R.M}}$ is the output of a T1 FLS constructed only from the *possible-right-most* embedded T1 FSs. In [44], an IT2 FS was obtained by blurring a triangular T1 FS left and right, and hence, the possible-left-most and possible-right-most embedded T1 FSs can be easily identified, as shown in Fig. 14(a); however, Tao *et al.* did not discuss how to identify these embedded T1 FSs for IT2 FSs with arbitrary FOU. In this paper, we construct these embedded T1 FSs as shown in Fig. 14(b). It is easy to observe that Tao *et al.*'s construction method is a special case of ours.

The Tao–Taur–Chang–Chang (TTCC) method does not require $\{y^n\}$ to be sorted. Observe that it is similar to the BMM method in that both methods compute the output of the IT2 FLS as a linear combination of the outputs of two T1 FLSs, which are constructed from the embedded T1 FSs. However, the BMM method uses the upper and lower MFs, whereas the TTCC method uses the possible-left-most and possible-right-most convex and normal embedded T1 FSs.

L. Computational Cost Comparison: Control Surface Computation

A comparison of the computational cost of the alternative TR algorithms in control surface computation using Gaussian MFs is shown in Fig. 15 and the second part of Table VI. The experimental setup was the same as that in Section III-F. Note that we did not include the DY method because its computational cost is much higher than others and increases exponentially with respect to N . Observe from Fig. 15 and Table VI the following points.

- 1) All ten alternative TR algorithms in Fig. 15(b) are faster than the KM algorithms, especially when M is small, e.g., $M < 10$.

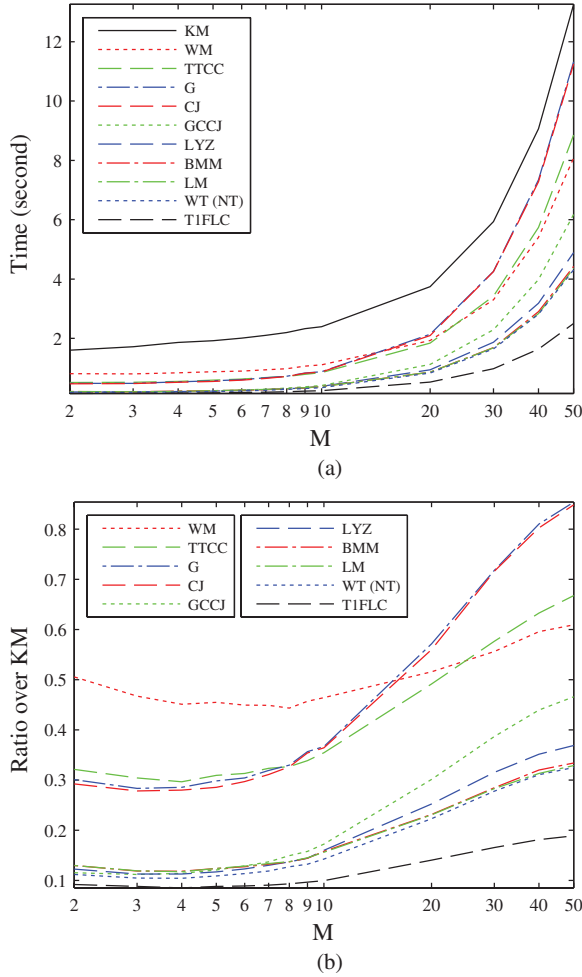


Fig. 15. Computational cost of the alternative TR algorithms in control surface computation using Gaussian MFs. (a) Total computation time of the 100 IT2 FLCs for different M (the number of IT2 FSs in each input domain). Note that $N = M^2$. (b) Ratio of the computation time of alternative type-reducers to the KM algorithms. To facilitate reading, the order of the algorithms in the legend is ranked roughly according to their performance.

- 2) The WT, NT, Liang–Mendel (LM), and BMM methods have similar computational cost, and generally, they are faster than other alternative TR algorithms.
- 3) The WT and NT methods are about 1.2–1.7 times slower than a T1 FLC.

We can observe from the first two parts of Table VI that the WT, NT, LM, and BMM methods are much faster than the EODS algorithms, the fastest enhancement to the KM algorithms. However, there is an important difference between the methods presented in this section and those in the previous section: All the enhanced versions of the KM algorithms introduced in Section III give exactly the same outputs as the original KM algorithms, which have some fundamentally different characteristics from T1 FLCs [47]. On the other hand, the alternative TR algorithms presented in this section have very different characteristics from the original KM algorithms, and hence their outputs are different. More comparisons on this are given later in this section.

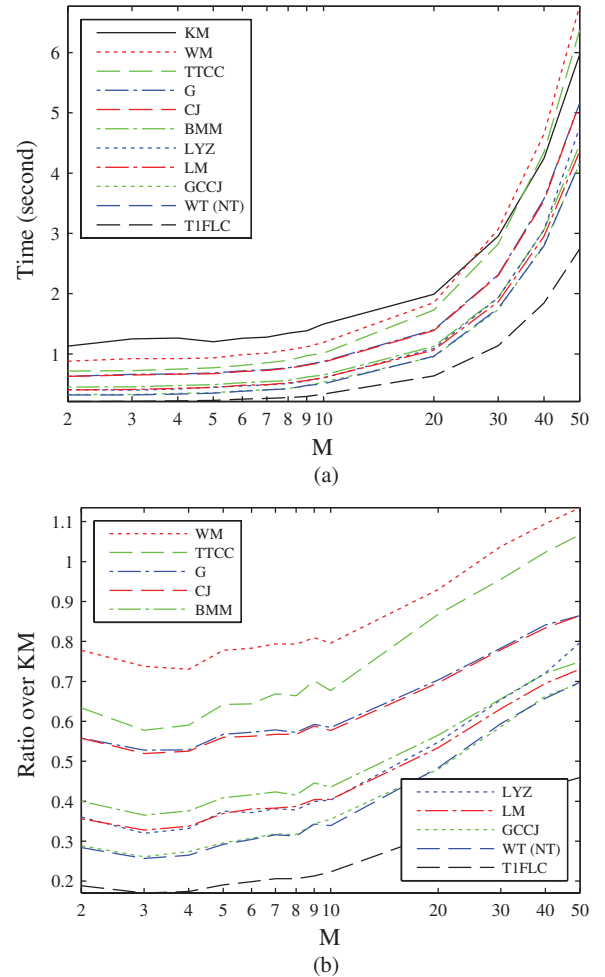


Fig. 16. Computational cost of the alternative TR algorithms in control surface computation using trapezoidal MFs. (a) Total computation time of the 100 IT2 FLCs for different M (the number of IT2 FSs in each input domain). Note that the rulebase has $N = M^2$ rules, but at any time, no more than four rules are fired. (b) Ratio of the computation time of alternative type-reducers to the KM algorithms. To facilitate reading, the order of the algorithms in the legend is ranked roughly according to their performance.

We repeated the previous experiments for trapezoidal IT2 FSs. The results are shown in Fig. 16 and the second part of Table VII. Observe the following points.

- 1) The computational cost saving of all algorithms, including the T1 FLC, over the KM algorithms is not as large as that in the Gaussian MF case. This is because for trapezoidal MFs at any time at most four rules are fired, so all algorithms converge very quickly.
- 2) The WT and NT methods are still the fastest; however, the GCCJ method is equally fast. This is because when a very small number of rules are fired, the time used to construct the closed polygon in the GCCJ method is negligible.

We can observe from the first two parts of Table VII that the WT, NT, and GCCJ methods are much faster than the EODS algorithms, which are the fastest enhancement to the KM algorithms.

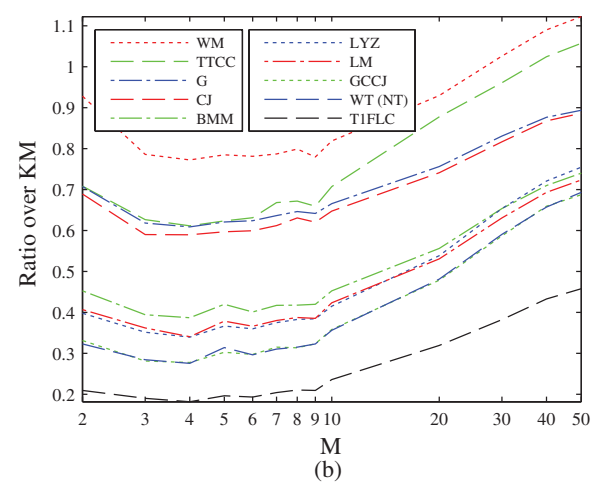
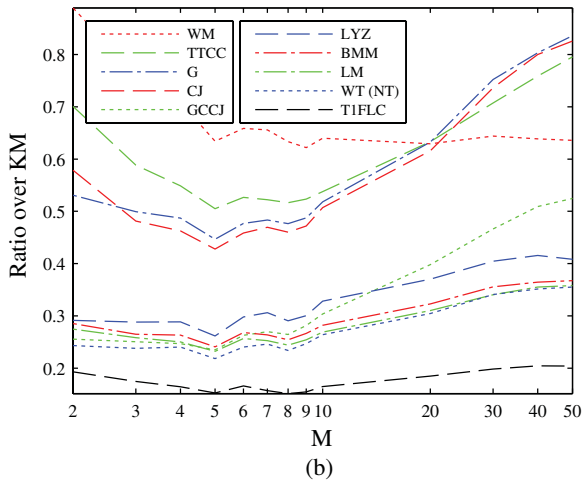
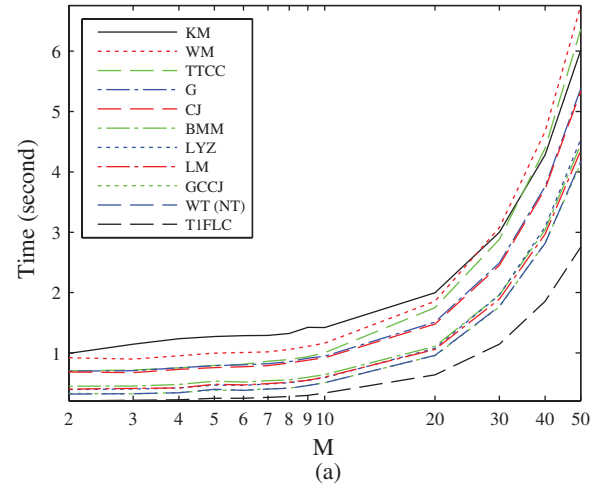
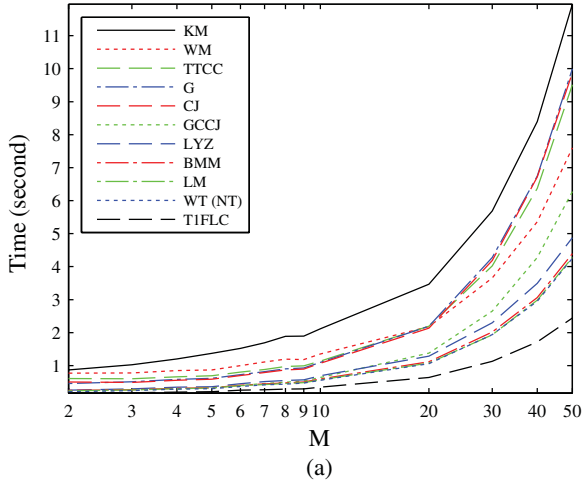


Fig. 17. Computational cost of the alternative TR algorithms in evolutionary IT2 FLC design using Gaussian MFs. (a) Total computation time of the 100 IT2 FLCs for different M : the number of IT2 FSs in each input domain. Note that $N = M^2$. (b) Ratio of the computation time of alternative type-reducers to the KM algorithms. To facilitate reading, the order of the algorithms in the legend is ranked roughly according to their performance.

Fig. 18. Computational cost of the alternative TR algorithms in evolutionary IT2 FLC design using trapezoidal MFs. (a) Total computation time of the 100 IT2 FLCs for different M : the number of IT2 FSs in each input domain. Note that $N = M^2$. (b) Ratio of the computation time of alternative type-reducers to the KM algorithms. To facilitate reading, the order of the algorithms in the legend is ranked roughly according to their performance.

M. Computational Cost Comparison: Evolutionary Fuzzy Logic Controller Design

We also compare the computational cost of the alternative TR algorithms in evolutionary FLC design. The results are shown in Figs. 17 and 18, and the second part of Tables VI and VII, for Gaussian IT2 FSs and trapezoidal IT2 FSs, respectively. All alternative TR algorithms are much faster than the KM algorithms when Gaussian IT2 FSs are used; however, their computational cost savings are not so large when trapezoidal IT2 FSs are used.

We can observe from the first two parts of Table VI that the WT, NT, LM, and BMM methods are much faster than the EODS algorithms, the fastest enhancement to the KM algorithms. We can also observe from the first two parts of Table VII that the WT, NT, and GCCJ methods are much faster than the EODS algorithms.

N. Comparison of the Defuzzified Outputs

The 11 alternative TR algorithms, generally, give different defuzzified outputs from the KM algorithms and their enhanced versions. Since the KM algorithms are the most popular TR algorithms and many theoretical (e.g., [47]) and experimental results are based on them, it is interesting to examine how close the outputs of the 11 alternative TR algorithms are to those of the KM algorithms. Surprisingly, no one has done these types of studies before. We believe that the conclusions are application dependent and a comprehensive comparison is out of the scope of this paper. We only use the results obtained in the previous experiments to perform some qualitative comparison, and we believe this should also give us some useful insight. The statistics of the absolute difference between the output of the KM TR method and ten alternative TR methods are shown in Table VIII. Observe the following points.

- 1) The WM method gives the closest approximation to the KM TR method.

TABLE VIII
STATISTICS OF THE ABSOLUTE DIFFERENCE BETWEEN THE OUTPUTS OF TEN ALTERNATIVE TR ALGORITHMS AND THOSE OF THE KM ALGORITHMS

TR Algorithm	Control Surface Computation				Evolutionary IT2 FLC Design			
	Gaussian		Trapezoidal		Gaussian		Trapezoidal	
	mean	std	mean	std	mean	std	mean	std
WM	0.043	0.087	0.077	0.104	0.035	0.075	0.078	0.103
G	0.169	0.185	0.127	0.183	0.103	0.155	0.127	0.179
CJ	0.235	0.251	0.142	0.195	0.132	0.192	0.151	0.202
TTCC	0.062	0.148	0.153	0.291	0.148	0.238	0.131	0.260
BMM	0.065	0.153	0.173	0.251	0.153	0.243	0.166	0.249
NT (WT)	0.065	0.157	0.196	0.310	0.158	0.257	0.172	0.281
GCCJ	0.066	0.163	0.202	0.314	0.166	0.266	0.178	0.285
T1	0.097	0.162	0.199	0.311	0.174	0.253	0.177	0.283
LM	2.702	4.536	0.323	0.228	1.164	2.815	0.261	0.226
LYZ	∞	∞	211	15661	∞	∞	214	14187

- Generally, the G, CJ, TTCC, BMM, NT, WT, and GCCJ methods give better approximation to the KM TR method than a T1 FLS, in which the UMFs of the corresponding IT2 FSs are used as its MFs.
- The output of the LM method is significantly different from other methods because it is an unnormalized method.
- There is a huge difference between the output of the LYZ method and all other methods.

However, we need to point out that the alternative TR methods give different output from the KM TR method, but it does not mean that they have poor performance. In fact, many of them have demonstrated good performance in the literature. It only means that the MFs need to be retuned when a different TR method is used.

O. Summary

Observe from the first two parts of Tables VI and VII that the WT, NT, LM, BMM, LYZ, and GCCJ methods consistently outperform the EODS algorithms: the fastest KM algorithm based type-reducer. Among them, the WT and NT methods seem to be the fastest alternative TR algorithms. However, we should also point out that there are many other considerations [48], beyond the computational cost, in IT2 FLS design. Among these six fast alternative TR algorithms, the LM and LYZ methods give significantly different results from the KM algorithms. The BMM method is the only one whose stability [3] and robustness [4], [5] have been extensively studied. Therefore, it may be preferred in practice.

V. SIMPLIFIED INTERVAL TYPE-2 FUZZY LOGIC SETS

In the previous two sections, we consider IT2 FLSs whose MFs are all IT2 FSs, and the introduced TR computational cost reduction approaches can be applied to IT2 FLSs in a variety of applications. However, when specified to IT2 fuzzy logic control, the most widely used application of IT2 FLSs, some unique observations can be made. We have demonstrated that an IT2 FLC's ability to eliminate oscillations can be attributed to its control surface near the steady state [54], [55], [59]. To save computational cost, while maintaining their superior ability to eliminate oscillations, we proposed a simplified architecture for IT2 FLCs [54], [59], where IT2 FSs are only used for the most critical regions in the input domains. The rest of the input domains are covered by T1 FSs.

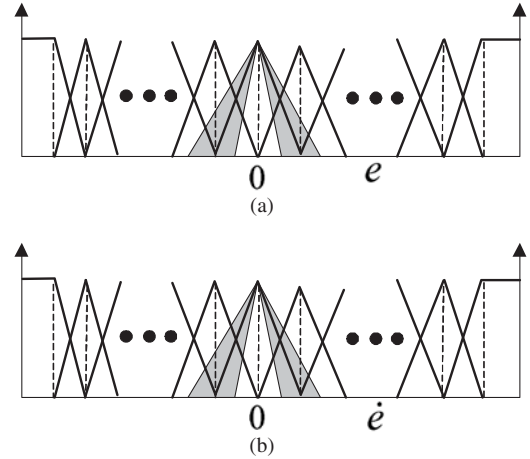


Fig. 19. MFs of the simplified IT2 FLS in (a) the e domain, and (b) the \dot{e} domain. Note that the middle MF in each domain is an IT2 FS. All other MFs are T1 FSs.

Consider the PI controller in (12). The control surface near $e = 0$ and $\dot{e} = 0$ is mainly responsible for eliminating oscillations. Therefore, in the simplified architecture shown in Fig. 19, we use some (usually only one) IT2 FSs to cover the areas around $e = 0$ and $\dot{e} = 0$ and T1 FSs for the rest of the input domains.

Clearly, a simplified IT2 FLC using trapezoidal MFs has two parts—a T1 part and an IT2 part. Different fuzzy partitions will be activated when the state of the plant is in different operating regions. During the transient stage, the FLC behaves like a T1 FLC since no IT2 FSs are fired. When the output approaches the set point, IT2 FSs will be fired and the plant is controlled by an IT2 FLC. Smoother control signals will be generated, which help eliminate oscillations.

There are two approaches to design a simplified IT2 FLC:

- 1) *the one-step approach*, where we prespecify the number of IT2 FSs near the steady state and then design the simplified IT2 FLC through experience or optimization algorithms;
- 2) *the two-step approach*, where we design a baseline T1 FLC first, change some T1 FSs near the steady state to be IT2 FSs, and retune the parameters using an optimization algorithm.

The simplified IT2 FLCs have been used in [54] and [59]. In [54], a simplified IT2 FLC, which used one IT2 FS around $e = 0$ and one around $\dot{e} = 0$, outperformed a T1 FLC with the same number of MFs and showed similar performance as an IT2 FLC whose MFs are all IT2 FSs. In [59], a simplified IT2 FLC, which used only one IT2 FS in the \dot{e} domain, outperformed a T1 FLC with the same number of MFs and showed similar performance as a T1 FLC with more MFs and an IT2 FLC whose all MFs are IT2 FSs.

A. Type-Reduction and Defuzzification of the Simplified Interval Type-2 Fuzzy Logic Controller

There are two categories of methods for TR and defuzzification of the simplified IT2 FLC. The first category is to use the KM algorithms or their enhanced versions introduced in Section III. The second is to use the alternative TR algorithms

introduced in Section IV. The second category of methods is very straightforward; therefore, only the first category of methods is described in this section.

Consider a simplified IT2 FLC where M out of the N rules contain only T1 FS in the antecedent. The remaining $N - M$ rules have at least one IT2 FS in the antecedent. There will, therefore, be M crisp firing strengths f^n , $n = 1, 2, \dots, M$, and $N - M$ interval firing strengths, F^n , $n = M + 1, M + 2, \dots, N$. In this case, the center-of-sets type-reducer in (2) becomes

$$\begin{aligned} Y_{\text{cos}} &= \frac{\sum_{n=1}^M Y^n f^n + \sum_{n=M+1}^N Y^n F^n}{\sum_{n=1}^M f^n + \sum_{n=M+1}^N F^n} \\ &= \frac{\beta + \sum_{n=M+1}^N Y^n F^n}{\alpha + \sum_{n=M+1}^N F^n} \\ &= \frac{\beta}{\alpha} + \frac{\sum_{n=M+1}^N Y^n F^n - \frac{\beta}{\alpha} \sum_{n=M+1}^N F^n}{\alpha + \sum_{n=M+1}^N F^n} \\ &= \frac{\beta}{\alpha} + \frac{\sum_{n=M+1}^N (Y^n - \frac{\beta}{\alpha}) F^n}{\alpha + \sum_{n=M+1}^N F^n} \end{aligned} \quad (18)$$

where

$$\begin{aligned} \alpha &= \sum_{n=1}^M f^n \\ \beta &= \sum_{n=1}^M Y^n f^n = \left[\sum_{n=1}^M \underline{y}^n f^n, \sum_{n=1}^M \bar{y}^n f^n \right] \\ \frac{\beta}{\alpha} &= \left[\frac{\sum_{n=1}^M \underline{y}^n f^n}{\sum_{n=1}^M f^n}, \frac{\sum_{n=1}^M \bar{y}^n f^n}{\sum_{n=1}^M f^n} \right] \\ Y^n - \frac{\beta}{\alpha} &\equiv \left[\underline{y}^n - \frac{\sum_{n=1}^M \underline{y}^n f^n}{\sum_{n=1}^M f^n}, \bar{y}^n - \frac{\sum_{n=1}^M \bar{y}^n f^n}{\sum_{n=1}^M f^n} \right]. \end{aligned}$$

Defining Y_*^n and F^{N+1} as

$$Y_*^n = \begin{cases} Y^n - \frac{\beta}{\alpha}, & n = M + 1, M + 2, \dots, N \\ 0, & n = N + 1 \end{cases} \quad (19)$$

$$F^{N+1} = \alpha \quad (20)$$

and (18) can be further simplified to

$$Y_{\text{cos}} = \frac{\beta}{\alpha} + \frac{\sum_{n=M+1}^{N+1} Y_*^n F^n}{\sum_{n=M+1}^{N+1} F^n}. \quad (21)$$

The second term on the right-hand side of (21) can be calculated by the KM algorithms or their enhanced versions introduced in Section III.

B. Computational Cost Comparison: Control Surface Computation

In this section, we compare the computational cost of the simplified IT2 FLCs with “full” IT2 FLCs (whose all MFs are IT2 FSs) in control surface computation. Three TR approaches

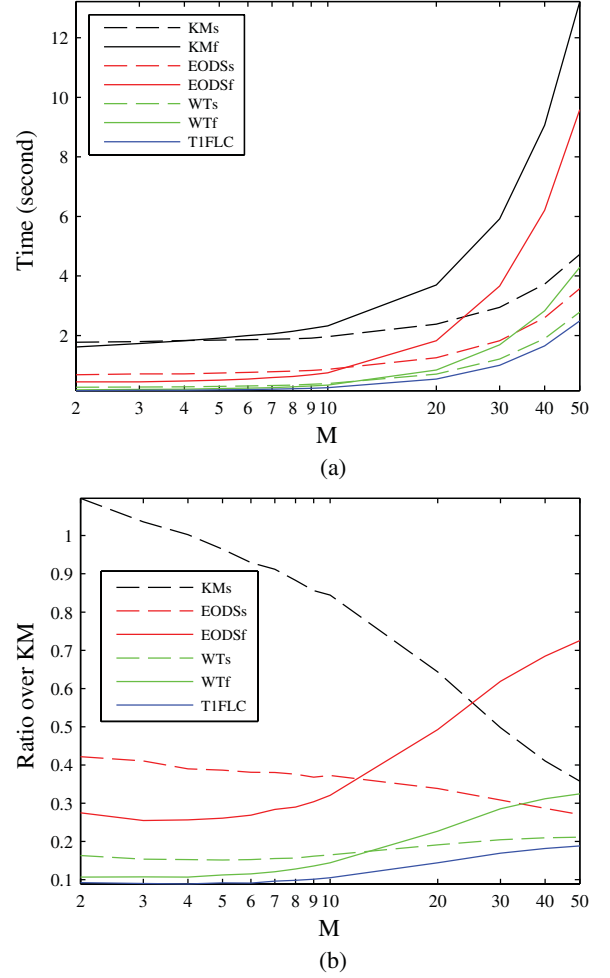


Fig. 20. Computational cost of the simplified IT2 FLC in control surface computation using Gaussian IT2 FSs. (a) Total computation time of the 100 IT2 FLCs for different M : the number of IT2 FSs in each input domain. Note that $N = M^2$. (b) Ratio of the computation time of different algorithms to the KM algorithm. In each legend, f means full, and s means simplified. To facilitate reading, the order of the algorithms in the legend is ranked roughly according to their performance.

are considered: 1) the standard KM type-reducer; 2) the EODS algorithms, which are the fastest enhancement to the KM algorithms in Section III; and 3) the WT (NT) method, which is the fastest alternative TR algorithm in Section IV. Additionally, we also record the computational cost of the corresponding T1 FLCs.

When Gaussian IT2 FSs are used and the simplified IT2 FLC has only the center MF in each input domain as IT2 FS, the results are shown in Fig. 20 and the third part of Table VI. The experimental setup was the same as that in Section III-F. Observe from Fig. 20 the following points.

- 1) When M is small, e.g., when there are fewer than ten MFs in each input domain, the computational cost of the simplified IT2 FLC is higher than the corresponding full IT2 FLC (KMf versus KM, EODSs versus EODSf, and WTf versus WT) because of the extra effort in constructing Y_*^n in (19) and F^{N+1} in (20). When M gets larger, the simplified IT2 FLC becomes faster than the full IT2

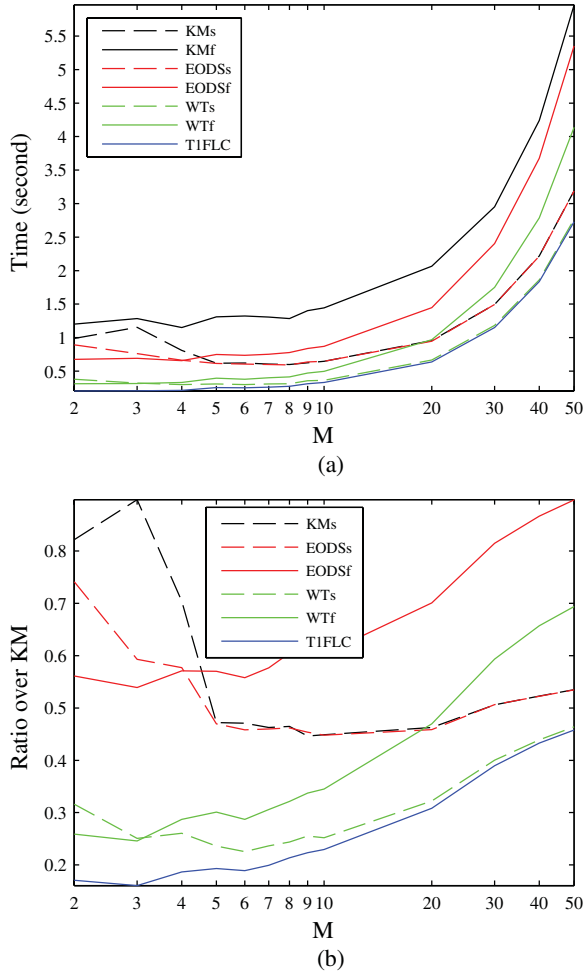


Fig. 21. Computational cost of the simplified IT2 FLC in control surface computation using trapezoidal IT2 FSSs. (a) Total computation time of the 100 control surfaces for different M : the number of IT2 FSSs in each input domain. Note that the rulebase has $N = M^2$ rules, but at any time, no more than four rules are fired. (b) Ratio of the computation time of different algorithms to the KM algorithms. In each legend, f means full, and s means simplified. To facilitate reading, the order of the algorithms in the legend is ranked roughly according to their performance.

TABLE IX
NUMBER OF RULES WHICH HAVE FIRING INTERVALS INSTEAD OF CRISP FIRING LEVELS IN THE SIMPLIFIED IT2 FLC

	Gaussian MFs	Trapezoidal MFs
When the input is far from the origin	$2M - 1$	0
When the input is around the origin	$2M - 1$	4

Note the total number of rules is M^2 .

FLC because the computational cost saving offered by the simplified structure outweighs the extra effort to construct Y_*^n and F^{N+1} .

- When M becomes larger, the speed of the simplified IT2 FLC, regardless of the TR method, approaches the T1 FLC because most part of the simplified IT2 FLC is essentially a T1 FLC.

We repeated the previous experiments for trapezoidal IT2 FSSs. The results are shown in Fig. 21 and the third part of Table VII. Observe that the simplified IT2 FLC is almost always faster than the corresponding full IT2 FLC. Recall that, in Fig. 20, this only happens when M is large. The reason is that

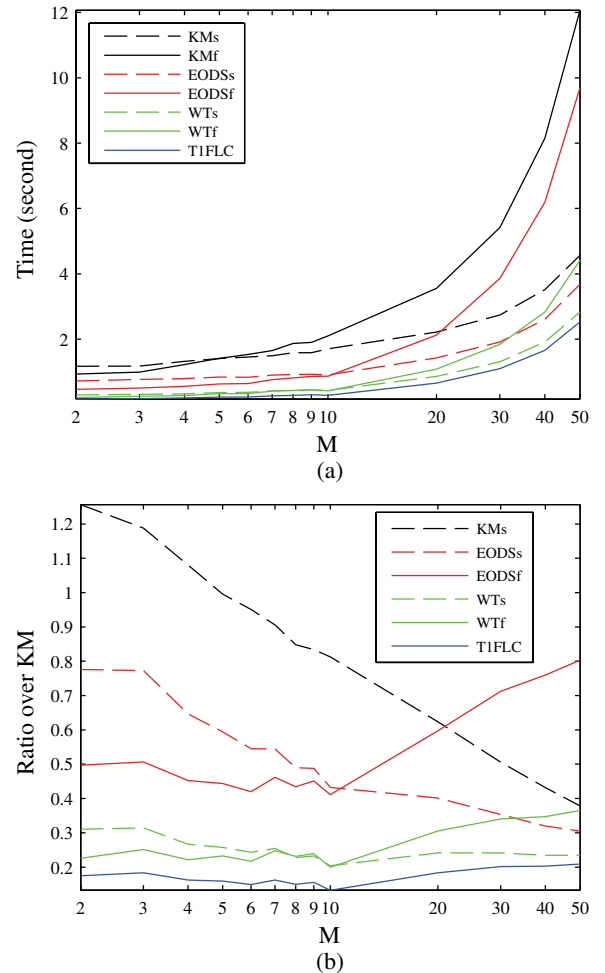


Fig. 22. Computational cost of the simplified IT2 FLC in evolutionary FLC design using Gaussian IT2 FSSs. (a) Total computation time of the 100 IT2 FLCs for different M : the number of IT2 FSSs in each input domain. Note that $N = M^2$. (b) Ratio of the computation time of different algorithms to the KM algorithms. In each legend, f means full, and s means simplified. To facilitate reading, the order of the algorithms in the legend is ranked roughly according to their performance.

when trapezoidal IT2 FSSs are used, except for the small area around the origin, the simplified IT2 FLC is identical to the T1 FLC since the firing strengths on the center IT2 FSSs are zero; however, when Gaussian IT2 FSSs are used, no matter where the input is, its firing strengths on the center IT2 FSSs are always nonzero intervals, and hence, there are more interval rule firing strengths in the Gaussian FS case than in the trapezoidal FS case. A comparison of the number of rules which have firing intervals is shown in Table IX.

C. Computational Cost Comparison: Evolutionary Fuzzy Logic Controller Design

We also compared the computational cost of the simplified IT2 FLC with the corresponding full IT2 FLC in evolutionary FLC design. The results are shown in Figs. 22 and 23, and the third part of Tables VI and VII, for Gaussian IT2 FSSs and trapezoidal IT2 FSSs, respectively. We have similar observations as those in the previous section, i.e., for Gaussian IT2 FSSs,

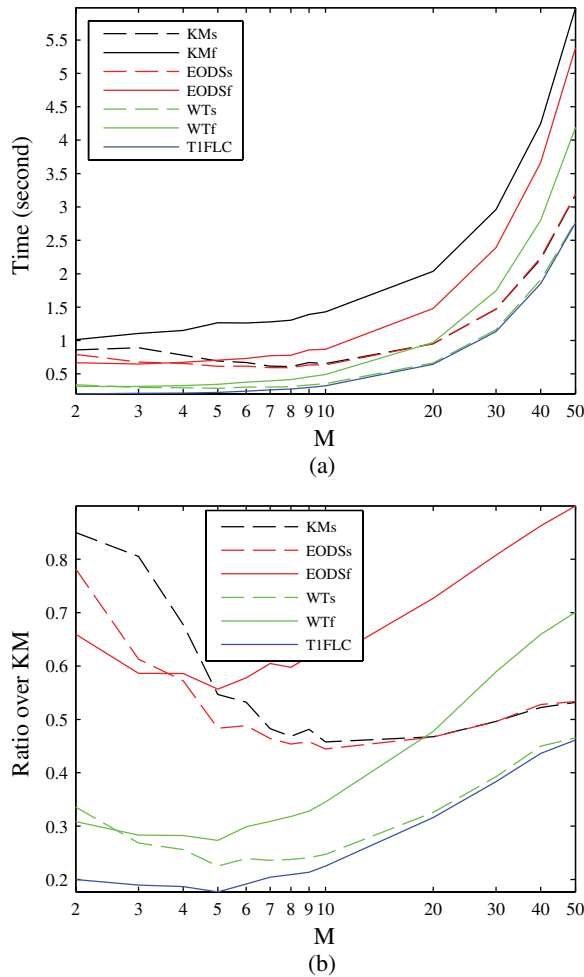


Fig. 23. Computational cost of the simplified IT2 FLC in evolutionary FLC design using trapezoidal IT2 FSs. (a) Total computation time of the 100 control surfaces for different M : the number of IT2 FSs in each input domain. Note that the rulebase has $N = M^2$ rules, but at any time, no more than four rules are fired. (b) Ratio of the computation time of different algorithms to the KM algorithms. In each legend, f means full, and s means simplified. To facilitate reading, the order of the algorithms in the legend is ranked roughly according to their performance.

the simplified IT2 FLC becomes faster than the corresponding full IT2 FLC when M is large, whereas for trapezoidal IT2 FSs, the simplified IT2 FLC is almost always faster than the corresponding full IT2 FLC.

D. Summary

In summary, we have demonstrated that a simplified IT2 FLC can save a significant amount of computational cost over a full IT2 FLC, especially when the number of rules is large and the EODS algorithms or the WT (NT) method is used in TR. Particularly, the simplified IT2 FLC using the WT (NT) method has the fastest speed.

VI. CONCLUSION

IT2 FLSs have demonstrated better abilities to handle uncertainties than their T1 counterparts in many applications; however, their high computational cost may hinder them from certain cost-sensitive real-world applications. In this paper, we

have provided a comprehensive overview and comparison of three categories of methods to reduce the computational cost of IT2 FLSs. The first category consists of five enhancements to the KM algorithms. Experiments demonstrated that generally they are all faster than the KM algorithms; among them, the EODS algorithms are the fastest for practical IT2 FLSs. Additionally, the EIASC algorithms, which are much simpler than the EODS algorithms and are at most 1.2 times slower, may also be preferred by practitioners for the ease in understanding and implementation. The second category consists of 11 alternative type-reducers, which have closed-form representation and, hence, are more convenient for analysis. Experiments demonstrated that except for the DY method, all the other ten methods are generally faster than the KM algorithms; among them, the WT and NT methods are the fastest. The BMM method may also be preferred because its properties, e.g., stability and robustness, have been extensively studied. The third category consists of a simplified structure for IT2 FLCs, which can be combined with any algorithm in the first or second category. Experiments demonstrated that a simplified IT2 FLC can save a significant amount of computational cost over a full IT2 FLC, especially when the number of rules is large. Particularly, the simplified IT2 FLC using the WT or NT method has the fastest speed. However, it is important to note that the first two categories of methods can be applied to all IT2 FLSs, whereas the simplified structure is only designed for control applications.

The investigations in this paper will help researchers and practitioners on IT2 FLSs choose the most suitable structure and TR algorithms. Because there is no comprehensive comparison on the performances of the KM algorithms-based TR approaches and the alternative TR approaches (this is an interesting open problem), our recommendation is to start from the full IT2 FLS using the EODS or EIASC TR algorithms, because most studies so far use the KM algorithm-based type-reducer. If more computational cost saving is desired, then alternative TR algorithms like the WT, NT, BMM, or GCCJ method may be considered because they are consistently faster than the EODS algorithms and their outputs are close to the outputs of the KM algorithms. Particularly, the BMM method may be preferred because its stability and robustness have also been extensively studied. If the IT2 FLS is used in control and the rulebase is large, then it may also be worthwhile to use the simplified structure to further save some computational cost.

REFERENCES

- [1] J. Aisbett, J. T. Rickard, and D. Morgenthaler, "Multivariate modeling and type-2 fuzzy sets," *Fuzzy Sets Syst.*, vol. 163, no. 1, pp. 78–95, 2011.
- [2] M. Begian, W. Melek, and J. Mendel, "Stability analysis of type-2 fuzzy systems," in *Proc. IEEE Int. Conf. Fuzzy Syst.*, Jun. 2008, pp. 947–953.
- [3] M. Biglarbegan, W. Melek, and J. Mendel, "On the stability of interval type-2 TSK fuzzy logic control systems," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 41, no. 5, pp. 798–818, Jun. 2010.
- [4] M. Biglarbegan, W. Melek, and J. Mendel, "Robustness of interval type-2 fuzzy logic systems," in *Proc. North Am. Fuzzy Inf. Process. Soc.*, Toronto, ON, Canada, Jul. 2010, pp. 1–6.
- [5] M. Biglarbegan, W. Melek, and J. Mendel, "On the robustness of type-1 and interval type-2 fuzzy logic systems in modeling," *Inf. Sci.*, vol. 181, no. 7, pp. 1325–1347, 2011.

- [6] O. Castillo and P. Melin, *Type-2 Fuzzy Logic Theory and Applications*. Berlin, Germany: Springer-Verlag, 2008.
- [7] O. Castillo, P. Melin, A. Alanis, O. Montiel, and R. Sepulveda, "Optimization of interval type-2 fuzzy logic controllers using evolutionary algorithms," *Soft Comput.*, vol. 15, no. 6, pp. 1145–1160, Jun. 2011.
- [8] S. Coupland and R. I. John, "Geometric type-1 and type-2 fuzzy logic systems," *IEEE Trans. Fuzzy Syst.*, vol. 15, no. 1, pp. 3–15, Feb. 2007.
- [9] T. Dereeli, A. Baykasoğlu, K. Altun, A. Durmusoğlu, and I. B. Turksen, "Industrial applications of type-2 fuzzy sets and systems: A concise review," *Comput. Ind.*, vol. 62, pp. 125–137, 2011.
- [10] X. Du and H. Ying, "Derivation and analysis of the analytical structures of the interval type-2 fuzzy-PI and PD controllers," *IEEE Trans. Fuzzy Syst.*, vol. 18, no. 4, pp. 802–814, Aug. 2010.
- [11] K. Duran, H. Bernal, and M. Melgarejo, "Improved iterative algorithm for computing the generalized centroid of an interval type-2 fuzzy set," in *Proc. North Amer. Fuzzy Inf. Process. Soc.*, New York, May 2008, pp. 1–5.
- [12] J. M. Garibaldi and S. Guadarrama, "Constrained type-2 fuzzy sets," in *Proc. IEEE Symposium Adv. Type-2 Fuzzy Logic Syst.*, Paris, France, Apr. 2011, pp. 66–73.
- [13] M. Gorzalczany, "Decision making in signal transmission problems with interval-valued fuzzy sets," *Fuzzy Sets Syst.*, vol. 23, pp. 191–203, 1987.
- [14] M. Gorzalczany, "Interval-valued fuzzy controller based on verbal model of object," *Fuzzy Sets Syst.*, vol. 28, pp. 45–53, 1988.
- [15] S. Greenfield, R. John, and S. Coupland, "A novel sampling method for type-2 defuzzification," in *Proc. UK Workshop Comput. Intell.*, London, U.K., Sep. 2005, pp. 120–127.
- [16] S. Greenfield, F. Chiclana, S. Coupland, and R. John, "The collapsing method of defuzzification for discretised interval type-2 fuzzy sets," *Inf. Sci.*, vol. 179, no. 13, pp. 2055–2069, 2008.
- [17] L. Gu and Y. Q. Zhang, "Web shopping expert using new interval type-2 fuzzy reasoning," *Soft Comput.*, vol. 11, no. 8, pp. 741–751, 2007.
- [18] H. Hagrass, "Type-2 FLCs: A new generation of fuzzy controllers," *IEEE Computat. Intell. Mag.*, vol. 2, no. 1, pp. 30–43, Feb. 2007.
- [19] D. Hidalgo, O. Castillo, and P. Melin, "Type-1 and type-2 fuzzy inference systems as integration methods in modular neural networks for multimodal biometry and its optimization with genetic algorithms," *Inf. Sci.*, vol. 179, no. 13, pp. 2123–2145, 2009.
- [20] M. Hsiao, T. H. S. Li, J. Z. Lee, C. H. Chao, and S. H. Tsai, "Design of interval type-2 fuzzy sliding-mode controller," *Inf. Sci.*, vol. 178, no. 6, pp. 1686–1716, 2008.
- [21] H. Z. Hu, G. Zhao, and H. N. Yang, "Fast algorithm to calculate generalized centroid of interval type-2 fuzzy set," *Control Decis.*, vol. 25, no. 4, pp. 637–640, 2010.
- [22] 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 J. Control*, vol. 14, no. 6, pp. 1–9, 2012.
- [23] N. N. Karnik and J. M. Mendel, "Centroid of a type-2 fuzzy set," *Inf. Sci.*, vol. 132, pp. 195–220, 2001.
- [24] N. N. Karnik, J. M. Mendel, and Q. Liang, "Type-2 fuzzy logic systems," *IEEE Trans. Fuzzy Syst.*, vol. 7, pp. 643–658, 1999.
- [25] C.-S. Lee, M.-H. Wang, and H. Hagrass, "A type-2 fuzzy ontology and its application to personal diabetic-diet recommendation," *IEEE Trans. Fuzzy Syst.*, vol. 18, no. 2, pp. 374–395, Apr. 2010.
- [26] C. Li, J. Yi, and T. Wang, "Stability analysis of SIRMs based type-2 fuzzy logic control systems," in *Proc. IEEE Int. Conf. Fuzzy Syst.*, Taipei, Taiwan, Jun. 2011, pp. 1–7.
- [27] C. Li, J. Yi, and D. Zhao, "A novel type-reduction method for interval type-2 fuzzy logic systems," in *Proc. 5th Int. Conf. Fuzzy Syst. Knowl. Discov.*, vol. 1, Jinan, China, Mar. 2008, pp. 157–161.
- [28] Q. Liang and J. M. Mendel, "Equalization of nonlinear time-varying channels using type-2 fuzzy adaptive filters," *IEEE Trans. Fuzzy Syst.*, vol. 8, no. 5, pp. 551–563, Oct. 2000.
- [29] F. Liu, "An efficient centroid type-reduction strategy for general type-2 fuzzy logic system," *Inf. Sci.*, vol. 178, no. 9, pp. 2224–2236, 2008.
- [30] X. Liu, J. Mendel, and D. Wu, "Study on enhanced karnik-mendel algorithms: Initialization explanations and computation improvements," *Inf. Sci.*, vol. 184, no. 1, pp. 75–91, 2012.
- [31] M. Melgarejo, "A fast recursive method to compute the generalized centroid of an interval type-2 fuzzy set," in *Proc. North Amer. Fuzzy Inf. Process. Soc.*, San Diego, CA, Jun. 2007, pp. 190–194.
- [32] P. Melin, O. Mendoza, and O. Castillo, "Face recognition with an improved interval type-2 fuzzy logic Sugeno integral and modular neural networks," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 41, no. 5, pp. 1001–1012, Oct. 2011.
- [33] J. M. Mendel, *Uncertain Rule-Based Fuzzy Logic Systems: Introduction and New Directions*. Upper Saddle River, NJ: Prentice-Hall, 2001.
- [34] J. M. Mendel and R. I. John, "Type-2 fuzzy sets made simple," *IEEE Trans. Fuzzy Syst.*, vol. 10, no. 2, pp. 117–127, Apr. 2002.
- [35] J. M. Mendel and D. Wu, *Perceptual Computing: Aiding People in Making Subjective Judgments*. Hoboken, NJ: Wiley-IEEE Press, 2010.
- [36] H. B. Mitchell, "Pattern recognition using type-II fuzzy sets," *Inf. Sci.*, vol. 170, nos. 2–4, pp. 409–418, 2005.
- [37] M. Nie and W. W. Tan, "Towards an efficient type-reduction method for interval type-2 fuzzy logic systems," in *Proc. IEEE Int. Conf. Fuzzy Syst.*, Jun. 2008, pp. 1425–1432.
- [38] A. Niewiadomski, "A type-2 fuzzy approach to linguistic summarization of data," *IEEE Trans. Fuzzy Syst.*, vol. 16, no. 1, pp. 198–212, Feb. 2008.
- [39] A. Niewiadomski, J. Ochelska, and P. Szczepaniak, "Interval-valued linguistic summaries of databases," *Control Cybern.*, vol. 35, no. 2, pp. 415–443, 2006.
- [40] S.-K. Oh, H.-J. Jang, and W. Pedrycz. (2011, Sep.). A comparative experimental study of type-1/type-2 fuzzy cascade controller based on genetic algorithms and particle swarm optimization. *Expert Syst. Appl.* [Online]. vol. 38, pp. 11217–11229. Available: <http://dx.doi.org/10.1016/j.eswa.2011.02.169>.
- [41] M. R. Rajati, D. Wu, and J. M. Mendel, "On solving Zadeh's Tall Swedes problem," in *Proc. World Conf. Soft Comput.*, San Francisco, CA, May 2011.
- [42] F. C.-H. Rhee, "Uncertainty fuzzy clustering: Insights and recommendations," *IEEE Comput. Intell. Mag.*, vol. 2, no. 1, pp. 44–56, Feb. 2007.
- [43] R. Sepulveda, O. Montiel, O. Castillo, and P. Melin, "Embedding a high speed interval type-2 fuzzy controller for a real plant into an FPGA," *Appl. Soft Comput.*, vol. 12, no. 3, pp. 988–998, 2012.
- [44] C. W. Tao, J. S. Taur, C.-W. Chang, and Y.-H. Chang, "Simplified type-2 fuzzy sliding controller for wing rock system," *Fuzzy Sets Syst.*, 2012, to be published.
- [45] C. Ulu, M. Guzelkaya, and T. Eksin, "A dynamic defuzzification method for interval type-2 fuzzy logic controllers," in *Proc. IEEE Int. Conf. Mechatronics*, Istanbul, Turkey, Apr. 2011, pp. 318–323.
- [46] D. Wu, "A constrained representation theorem for interval type-2 fuzzy sets using convex and normal embedded type-1 fuzzy sets, and its application to centroid computation," in *Proc. World Conf. Soft Comput.*, San Francisco, CA, May 2011.
- [47] D. Wu, "On the fundamental differences between interval type-2 and type-1 fuzzy logic controllers," *IEEE Trans. Fuzzy Syst.*, 2012, to be published.
- [48] D. Wu, "Twelve considerations in choosing between Gaussian and trapezoidal membership functions in interval type-2 fuzzy logic controllers," in *Proc. IEEE World Congr. Comput. Intell.*, Brisbane, Australia, Jun. 2012.
- [49] D. Wu and J. M. Mendel, "Enhanced Karnik–Mendel algorithms for interval type-2 fuzzy sets and systems," in *Proc. North Amer. Fuzzy Inf. Process. Soc.*, San Diego, CA, Jun. 2007, pp. 184–189.
- [50] D. Wu and J. M. Mendel, "Enhanced Karnik–Mendel algorithms," *IEEE Trans. Fuzzy Syst.*, vol. 17, no. 4, pp. 923–934, Aug. 2009.
- [51] D. Wu and J. M. Mendel, "Computing with words for hierarchical decision making applied to evaluating a weapon system," *IEEE Trans. Fuzzy Syst.*, vol. 18, no. 3, pp. 441–460, Jun. 2010.
- [52] D. Wu and J. M. Mendel, "Linguistic summarization using IF-THEN rules and interval type-2 fuzzy sets," *IEEE Trans. Fuzzy Syst.*, vol. 19, no. 1, pp. 136–151, Feb. 2011.
- [53] D. Wu and M. Nie, "Comparison and practical implementation of type-reduction algorithms for type-2 fuzzy sets and systems," in *Proc. IEEE Int. Conf. Fuzzy Syst.*, Taipei, Taiwan, Jun. 2011, pp. 2131–2138.
- [54] D. Wu and W. W. Tan, "A simplified architecture for type-2 FLSs and its application to nonlinear control," in *Proc. IEEE Conf. Cybern. Intell. Syst.*, Dec. 2004, pp. 485–490.
- [55] D. Wu and W. W. Tan, "A type-2 fuzzy logic controller for the liquid-level process," in *Proc. IEEE Int. Conf. Fuzzy Syst.*, vol. 2, Budapest, Hungary, Jul. 2004, pp. 953–958.
- [56] D. Wu and W. W. Tan, "Computationally efficient type-reduction strategies for a type-2 fuzzy logic controller," in *Proc. IEEE Int. Conf. Fuzzy Syst.*, Reno, NV, May 2005, pp. 353–358.
- [57] D. Wu and W. W. Tan, "Type-2 FLS modeling capability analysis," in *Proc. IEEE Int. Conf. Fuzzy Syst.*, Reno, NV, May 2005, pp. 242–247.
- [58] D. Wu and W. W. Tan, "Genetic learning and performance evaluation of type-2 fuzzy logic controllers," *Eng. Appl. Artif. Intell.*, vol. 19, no. 8, pp. 829–841, 2006.
- [59] D. Wu and W. W. Tan, "A simplified type-2 fuzzy controller for real-time control," *ISA Trans.*, vol. 15, no. 4, pp. 503–516, 2006.

- [60] D. Wu and W. W. Tan, "Interval type-2 fuzzy PI controllers: Why they are more robust," in *Proc. IEEE Int. Conf. Granular Comput.*, San Jose, CA, Aug. 2010, pp. 802–807.
- [61] H. Wu and J. M. Mendel, "Uncertainty bounds and their use in the design of interval type-2 fuzzy logic systems," *IEEE Trans. Fuzzy Syst.*, vol. 10, no. 5, pp. 622–639, Oct. 2002.
- [62] H. Wu and J. M. Mendel, "Classification of battlefield ground vehicles using acoustic features and fuzzy logic rule-based classifiers," *IEEE Trans. Fuzzy Syst.*, vol. 15, no. 1, pp. 56–72, Feb. 2007.
- [63] H.-J. Wu, Y.-L. Su, and S.-J. Lee, "A fast method for computing the centroid of a type-2 fuzzy set," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 42, no. 3, pp. 764–777, Jun. 2012.
- [64] C.-Y. Yeh, W.-H. Jeng, and S.-J. Lee, "An enhanced type-reduction algorithm for type-2 fuzzy sets," *IEEE Trans. Fuzzy Syst.*, vol. 19, no. 2, pp. 227–240, Apr. 2011.
- [65] L. A. Zadeh, "The concept of a linguistic variable and its application to approximate reasoning-1," *Inf. Sci.*, vol. 8, pp. 199–249, 1975.
- [66] J. Zeng and Z.-Q. Liu, "Type-2 fuzzy hidden Markov models and their applications to speech recognition," *IEEE Trans. Fuzzy Syst.*, vol. 14, no. 3, pp. 454–467, Jun. 2006.



Dongrui Wu (S'05–M'09) received the B.E. degree in automatic control from the University of Science and Technology of China, Hefei, China, in 2003, the M.Eng. degree in electrical engineering from the National University of Singapore, Singapore, in 2005, and the Ph.D. degree in electrical engineering from the University of Southern California, Los Angeles, in 2009.

He is currently with the Machine Learning Lab, GE Global Research, Niskayuna, NY. He has written for more than 60 publications, including a book *Perceptual Computing* (with J. M. Mendel, New York: Wiley-IEEE, 2010).

Dr. Wu received the 2005 IEEE International Conference on Fuzzy Systems Best Student Paper Award, the 2012 IEEE Computational Intelligence Society (CIS) Outstanding Ph.D. Dissertation Award and an Award of Excellence from GE Global Research in 2010 for outstanding performance. He is currently an Associate Editor of the *IEEE TRANSACTIONS ON FUZZY SYSTEMS* and an Editorial Board Member of the *International Journal of Human Computer Interaction*. He is also an Executive of Human–Machine Interaction Network on Emotion and a member of several IEEE CIS Technical Committees. He is the Chair of the IEEE CIS Affective Computing Task Force and the Vice Chair of the Computing with Words Task Force.