On Computing the Fuzzy Weighted Average Using the KM Algorithms

Feilong Liu and Jerry M. Mendel
Signal and Image Processing Institute,
Department of Electrical Engineering
University of Southern California,
3740 McClintock Ave,
Los Angeles, CA 90089-2564
e-mail: feilong@usc.edu/mendel@siipi.usc.edu

Abstract

By connecting the fuzzy weighted average (FWA) and the generalized centroid (GC) of an interval type-2 fuzzy set we have arrived at a new \( \alpha \)-cut algorithm for solving the FWA problem, one that is monotonically and super-exponentially convergent. Our new algorithm uses the Karnik-Mendel (KM) algorithms to compute the FWA \( \alpha \)-cut end-points. No faster algorithm for solving this problem exists to-date.

Keywords: Fuzzy weighted average, centroid, KM algorithms, interval type-2 fuzzy set

1 Introduction

The fuzzy weighted average (FWA), which is a weighted average involving type-1 fuzzy sets, and is explained mathematically below, is useful as an aggregation (fusion) method in situations where decisions \( x_i \) and expert weights \( w_i \) are modeled as type-1 fuzzy sets, or where the decisions are modeled as either crisp numbers or interval sets, and the expert weights are still modeled as type-1 fuzzy sets.

Sometimes the same or similar problem is solved in different settings. This is a paper about such a situation, namely computing the FWA, which is a problem that has been studied in multiple criteria decision making ([1]-[4], [7], [8]), and computing the generalized centroid of an interval type-2 fuzzy set (herein referred to as the generalized centroid—GC), which is a problem that has been studied in rule-based interval type-2 fuzzy logic systems ([5], [9], [10]). Here we demonstrate how very significant computational advantages are obtained for solving the FWA problem by using the GC algorithms developed by Karnik and Mendel\(^1\).

To begin, consider the following weighted average:

\[
y = \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i} = f(w_1, \ldots, w_n, x_1, \ldots, x_n)
\]

(1)

In (1), \( w_i \) are weights that act upon attributes \( x_i \). Normalization is achieved by dividing the weighted numerator sum by the sum of all of the weights. Without normalization, too much emphasis would be placed on one attribute over another. It is the normalization that makes the calculation of \( y \) very challenging in the FWA.

In (1) we are interested in the case when some or all \( x_i \) are type-1 (T1) fuzzy numbers, i.e. each \( x_i \) is described by the membership function (MF) of a T1 fuzzy set (FS), \( \mu_{x_i}(x_i) \), where this MF must be pre-specified, and some or all \( w_i \) are also T1 fuzzy numbers, i.e. each \( w_i \) is described by the MF of a T1 FS, \( \mu_{w_i}(w_i) \), where this MF must also be pre-specified. \( y \) is a T1 FS, \( Y_{FWA} \) with MF \( \mu_{Y_{FWA}}(y) \), but there is no known closed-form formula for computing \( \mu_{Y_{FWA}}(y) \). \( \alpha \)–cuts, an \( \alpha \)–cut Decomposition Theorem [6] of a T1

\(^1\) These algorithms are often referred to in the T2 FLS literature as the “KM algorithms.” We shall also refer to them in this way.
FS, and a variety of algorithms ([1]-[4], [7], [8]) can be used to compute \( \mu_{\text{FWA}}(y) \).

2 Previous Approaches

Beginning in 1987, various solutions to compute \( \mu_{\text{FWA}}(y) \) have been proposed. Dong and Wong [1] were apparently the first to develop a method for computing the FWA. Although their algorithm is based on \( \alpha \)-cuts and an \( \alpha \)-cut Decomposition Theorem [6], it is very computationally inefficient, because it uses an exhaustive-search. Liou and Wang [8] did some important analyses that led to an improved algorithm (IFWA) that drastically reduced the computational complexity of Dong and Wong’s algorithm. They were the first to observe that (for each \( \alpha \)-cut) since the \( x_i \) appear only in the numerator of (1), only the smallest values of the \( x_i \) are used to find the smallest value of (1), and only the largest values of the \( x_i \) are used to find the largest value of (1) (Karnik and Mendel [5], who were unaware of Liou and Wang’s work, based their algorithms on the same observations). Lee and Park [7] presented an efficient FWA algorithm (EFWA) that built on the observations of Liou and Wang. The total computational complexities of the three algorithms for \( m \) \( \alpha \)-cuts are: FWA- \( O(m^22^n) \), IFWA- \( O(mn^2) \), and EFWA- \( O(2mn \ln n) \).

These are not the only methods that have been published. Other methods are in [2]-[4]. Except for [2], all of these methods are based on \( \alpha \)-cuts and an \( \alpha \)-cut Decomposition Theorem. Of the existing algorithms that use \( \alpha \)-cuts, the EFWA represents the most computationally efficient one to compute the FWA.

3 The KM Algorithms for Computing the FWA

As in the previous \( \alpha \)-cut based methods, we begin by discretizing the complete range of the membership \([0,1]\) of the fuzzy numbers into \( m \) \( \alpha \)-cuts, \( \alpha_1, \ldots, \alpha_m \), where the degree of accuracy depends on the number of \( \alpha \)-cuts, i.e. \( m \). Then, for each \( \alpha \), we find the corresponding intervals for \( X_i \) in \( x_i \) and \( W_i \) in \( w_i \), the end-points of which are \([a_i(\alpha), b_i(\alpha)]\) and \([c_i(\alpha), d_i(\alpha)]\), respectively \( (i = 1,\ldots,n) \). As in [7], [8], we compute \((j = 1,\ldots,m)\)

\[
Y_{\text{FWA}}(\alpha_j) = \left[ f'_L(\alpha_j), f'_R(\alpha_j) \right]
\]

where

\[
f'_L(\alpha_j) = \min_{\forall w_i \in [c_i(\alpha_j), d_i(\alpha_j)]} \sum_{i=1}^{n} w_i(\alpha_j) a_i(\alpha_j)
\]

(3)

\[
f'_R(\alpha_j) = \max_{\forall w_i \in [c_i(\alpha_j), d_i(\alpha_j)]} \sum_{i=1}^{n} w_i(\alpha_j) b_i(\alpha_j)
\]

(4)

after which we define the indicator function

\[
I_{\alpha_j}(y) = \begin{cases} 
1 & \forall y \in \left[ f'_L(\alpha_j), f'_R(\alpha_j) \right] \\
0 & \forall y \notin \left[ f'_L(\alpha_j), f'_R(\alpha_j) \right]
\end{cases}
\]

(5)

from which we can then construct \( \mu_{\text{FWA}}(y) \), as

\[
\mu_{\text{FWA}}(y) = \sup_{\forall \alpha_j (j = 1,\ldots,m)} \alpha_j I_{\alpha_j}(y)
\]

(6)

What distinguishes our approach from earlier approaches are the ways in which we compute \( f'_L(\alpha_j) \) and \( f'_R(\alpha_j) \).

We have recognized that \( f'_L(\alpha_j) \) and \( f'_R(\alpha_j) \) are precisely the same calculations as needed in the computation of the GC of an interval T2 FS [5], [9]. \( f'_L \) in (3) and \( f'_R \) in (4) can be computed as (for notational convenience we do not show the explicit dependence of quantities on \( \alpha_j \)):

\[
f'_L = \min_{\forall w_i \in [c_i, d_i]} \sum_{i=1}^{n} w_i \sum_{j=1}^{k_i} a_j d_j + \sum_{j=k_i+1}^{n} a_j c_j
\]

(7)

\[
f'_R = \max_{\forall w_i \in [c_i, d_i]} \sum_{i=1}^{n} w_i \sum_{j=1}^{k_i} b_j c_j + \sum_{j=k_i+1}^{n} b_j d_j
\]

(8)

where switch points \( k_U \) and \( k_L \) are computed using the KM algorithms, which are stated next. Because of space limitations, we cannot provide derivations (our journal version of this paper will include them).
3.1 KM Algorithm for $k_U$ and $f_U^*$

0. Rearrangement for the $f_U^*$ calculations: In (1), $x_i$ are replaced by the $b_i$, where $b_i$ are arranged in ascending order, i.e. $b_1 < b_2 < \cdots < b_n$. Associate the respective $w_i$ with its (possibly) re-labeled $b_i$. In the remaining steps we shall assume that the notation used in (8) is for the re-ordered (if necessary) $w_i$ and $b_i$.

1. Initialize $w_i$ for $i = 1, 2, \ldots, n$, and then compute

$$f_U^* = \frac{n}{i=1} b_i w_i \bigg/ \sum_{i=1}^n w_i$$ (9)

Two ways for initializing $w_i$ are: (a) $w_i = (c_i + d_i) / 2$ for $i = 1, 2, \ldots, n$, and (b) $w_i = c_i$, $i \leq \lfloor (n+1)/2 \rfloor$ and $w_i = d_i$, $i > \lfloor (n+1)/2 \rfloor$, where $\lfloor \cdot \rfloor$ denotes the first integer equal to or smaller than $\cdot$.

2. Find $k$ ($1 \leq k \leq n-1$) such that $b_k \leq f_U^* < b_{k+1}$.

3. Set $w_i = c_i$ for $i \leq k$ and $w_i = d_i$ for $i \geq k+1$, and compute

$$f_U^*(b_i) = \frac{\sum_{i=1}^k b_c_i + \sum_{i=k+1}^n b_d_i}{\sum_{i=1}^k c_i + \sum_{i=k+1}^n d_i}$$ (10)

4. Check if $f_U^*(k) = f_U^*$. If yes, then stop. $f_U^*(k)$ is the maximum value of $\sum_{i=1}^n b_i w_i / \sum_{i=1}^n w_i$ and equals $f_U^*$, and $k$ equals the switch point $k_U$. If no, go to Step 5.

5. Set $f_U^*$ equal to $f_U^*(k)$ and go to Step 2.

3.2 KM algorithm for $k_L$ and $f_L^*$

0. Rearrangement for the $f_L^*$ calculations: In (1), $x_i$ are replaced by the $a_i$, where $a_i$ are arranged in ascending order, i.e. $a_1 < a_2 < \cdots < a_n$. Associate the respective $w_i$ with its (possibly) re-labeled $a_i$. In the remaining steps we shall assume that the notation used in (7) is for the re-ordered (if necessary) $w_i$ and $a_i$.

1. Initialize $w_i$ (in either of the two ways listed after (9)) for $i = 1, 2, \ldots, n$, and then compute

$$f_L^* = \frac{n}{i=1} a_i w_i \bigg/ \sum_{i=1}^n w_i$$ (11)

2. Find $k$ ($1 \leq k \leq n - 1$) such that $a_k \leq f_L^* < a_{k+1}$.

3. Set $w_i = a_i$ for $i \leq k$ and $w_i = d_i$ for $i \geq k+1$, and compute

$$f_L^*(a_i) = \frac{\sum_{i=1}^k a_i d_i + \sum_{i=k+1}^n a_i c_i}{\sum_{i=1}^k d_i + \sum_{i=k+1}^n c_i}$$ (12)

4. Check if $f_L^*(k) = f_L^*$. If yes, then stop. $f_L^*(k)$ is the minimum value of $\sum_{i=1}^n a_i w_i / \sum_{i=1}^n w_i$ and equals $f_L^*$, and $k$ equals the switch point $k_L$. If no, go to Step 5.

5. Set $f_L^*$ equal to $f_L^*(k)$ and go to Step 2.

4 Interpretation of the KM Algorithms

We state three properties for $f_U^*(k)$ so that the KM algorithm can be provided with a graphical interpretation for its computation. Similar properties exist for $f_L^*(k)$. It is assumed that $a_i$ have been sorted in increasing order so that $a_1 \leq a_2 \leq \cdots \leq a_n$. Additionally, recall that the minimum of $f_U^*(k)$ occurs at $k = k_L$; hence, $f_U^* = f_U^*(k_L)$. $k_L$ is called the switch point for the minimum calculation.

Property 1 (Location of the minimum): When $k = k_L$, then $a_{k_L} \leq f_U^*(k_L) < a_{k_L+1}$ where $f_U^*(k_L)$ is computed by setting $k = k_L$ in (12), and

$$\min (f_U^*(k)) = f_U^*(k_L) = f_U^*$.

This property locates the minimum $f_U^*$ by finding the minimum of $f_U^*(k)$, $k = 1, \ldots, n$, e.g. in Fig. 1, the minimum of $f_U^*(k)$ ($k = 1, \ldots, n$) is $f_U^*(4)$, so that $k_L = 4$. Observe that $a_4 \leq f_U^*(4) < a_5$, which locates the minimum.
Property 2 (Location of \( f_L^n(k) \) in relation to the line \( y = a_k \)): \( f_L^n(k) \) lies above the line \( y = a_k \) when \( a_k \) is to the left of \( f_L^n \), and lies below the line \( y = a_k \) when \( a_k \) is to the right of \( f_L^n \), i.e.

\[
\begin{align*}
  f_L^n(k) &> a_k, \quad \text{when } a_k < f_L^n k
  f_L^n(k) &< a_k, \quad \text{when } a_k > f_L^n k
\end{align*}
\]  

(13)

This property provides interesting relations between \( a_k \) and \( f_L^n(k) \) on both sides of the minimum \( f_L^n \), e.g. in Fig. 1 \( f_L^n(k) \) lies above \( a_k \) to the left of \( f_L^n \equiv f_L^n(4) \), and it lies below \( a_k \) to the right of \( f_L^n \equiv f_L^n(4) \).

This property does not imply \( f_L^n(k) \) is monotonic on either side of \( f_L^n \); but, it does demonstrate that \( f_L^n(k) \) cannot be above the line \( y = a_k \) to the right of \( f_L^n \). Property 3 is about the monotonicity of \( f_L^n(k) \). It will show, e.g. that \( f_L^n(k) \) is monotonically non-decreasing to the right of \( f_L^n \); but, this could occur in two very different ways, namely, \( f_L^n(k) \) could be above the line \( y = a_k \) or below that line to the right of \( f_L^n \). Property 2 rules out the former.

Property 3 (Monotonicity of \( f_L^n(k) \)): It is true that

\[
\begin{align*}
  f_L^n(k-1) &\geq f_L^n(k) \quad \text{when } a_k < f_L^n k
  f_L^n(k+1) &\geq f_L^n(k) \quad \text{when } a_k > f_L^n k
\end{align*}
\]  

(14)

This property shows that \( f_L^n(k) \) is a monotonic function (but not a strictly-monotonic function) on both sides of the minimum of the FWA. For example, in Fig. 1 \( f_L^n(k) \) monotonically decreases to the left of \( f_L^n \equiv f_L^n(4) \), whereas, it monotonically increases to the right of \( f_L^n \equiv f_L^n(4) \).

Fig. 1 provides a graphical interpretation of the KM algorithm that computes \( f_L^n \). The large dots are plots of \( f_L^n(k) \) for \( k = 1, \ldots, 9 \); note that \( k \) is associated with the subscript of \( a_k \). The 45-degree line \( y = a_k \) is shown because of the computations in Step 2 of the KM algorithm. \( f_L^n \) has been chosen according to Method (b) that is stated below (9). Because \( n = 9 \), \( \lfloor (n+1)/2 \rfloor = 5 \), which is why \( f_L^n \) is located at \( a_i \).

By virtue of Step 2 of the algorithm, we see that \( a_i \leq f_L^n < a_k \), and we then slide down the 45-degree line \( y = a_k \) until we reach \( a_k \), at which point \( f_L^n(4) \) is computed. This is the vertical line from \( a_k \) that intersects a large dot. Because \( f_L^n(4) \neq f_L^n \), the algorithm then goes through another complete iteration before it stops, at which time \( f_L^n(4) \) has been determined to be \( f_L^n \). For this example, the KM algorithm converges in two iterations.

5 Properties of the KM Algorithms

Recently, Mendel and Liu [10] have proven that:

1. The KM algorithms are monotonically convergent.

2. The KM algorithms are super-exponentially convergent. Let \( j \) denote an iteration counter (\( j = 1, 2, \ldots, \)). A formula for \( j \) as a function of prescribed bits of accuracy, \( \varepsilon \), is derived in [10]. Because we use this formula in Section 7 to support our simulations, we state it next for \( f_L^n \). A comparable formula exists for \( f_L^n \).

Let \( f_L^n(0) \) denote the first value of \( f_L^n \) as given in (11), and

\[
\delta \equiv \frac{f_L^n(1) - f_L^n}{f_L^n(0) - f_L^n} \leq 1
\]

(15)

Super-exponential convergence\(^2\) of the KM algorithm occurs to within \( \varepsilon \) bits of accuracy when

\[ \delta \leq 1 \]

\[ \text{follows from the monotonic convergence of the KM algorithms.} \]

\[ \text{Why this is “super-exponential convergence” rather than “convergence is explained fully in [10]. It is} \]

\[ \text{evidenced by the appearance of } \delta \text{ in (17), the amplitude of whose logarithm is not linear but is} \]

\[ \text{concave upwards, which is indicative of a super-exponential convergence factor.} \]
\[ |f_l^n(j) - f_l^n(j-1)| \leq \varepsilon \]  
(16)

which, as proven in [10], is satisfied by the first integer \( j_\varepsilon \geq 2 \) for which:

\[ |\delta^{j_\varepsilon} - \delta^{j_\varepsilon-1}| \leq \frac{\varepsilon \delta}{f'_l(0)-f'_L} \]  
(17)

In general, this equation has no closed form solution for \( j \); however, for small values of \( \delta \), we can drop the term \( \delta^{j_\varepsilon} \) (which is much smaller than the term \( \delta^{j_\varepsilon-1} \)), and solve for \( j_\varepsilon \) as:

\[ j_\varepsilon = \text{first integer larger than } j' \]  
(19)

We hasten to point out that the use of (18) and (19) requires knowing the answer \( f_L^* \) as well as \( f_L^*(1) \). The latter is only available after the first iteration of the KM algorithm; hence, the a priori use of these equations is limited. However, these equations can be used after-the-fact (as we do in Section 7) to confirm the very rapid convergence of the KM algorithms.

We have observed from many simulations that for two significant figures of accuracy, the convergence of the KM algorithms occurs in 2-6 iterations.

6 EFWA Algorithms

In Section 7 we compare convergence results for the KM algorithms with the EFWA algorithms. Based on our derivation of the KM algorithms, it is relatively easy to understand the following:

6.1 EFWA Algorithm for \( f_L^* \)

0. Rearrangement for the \( f_L^* \) calculations:

Same as Step 0 in the KM Algorithm.

1. Initialize \( first = 1 \) and \( last = n \).

2. Let \( k = \left\lfloor \frac{1}{2}(first + last) \right\rfloor \) and compute \( f_L^n(k) \) in (10).

3. If \( b_l \leq f_L^n(k) < b_{s1} \), stop and set \( f_L^* = f_L^n(k) \) and \( k = k_o; \) otherwise, go to Step 4.

4. If \( f_L^n(k) \geq b_l \), set \( first = k + 1; \) otherwise, set \( last = k \). Go to Step 2.

6.2 EFWA Algorithm for \( f_L^* \)

0. Rearrangement for the \( f_L^* \) calculations:

Same as Step 0 in the KM Algorithm.

1. Initialize \( first = 1 \) and \( last = n \).

2. Let \( k = \left\lfloor \frac{1}{2}(first + last) \right\rfloor \) and compute \( f_L^n(k) \) in (12).

3. If \( a_k \leq f_L^n(k) < a_{s1} \), stop and set \( f_L^* = f_L^n(k); \) otherwise, go to Step 4.

4. If \( f_L^n(k) \geq a_l \), set \( first = k + 1; \) otherwise, set \( last = k \). Go to Step 2.

Fig. 2 provides a graphical interpretation of the EFWA algorithm that computes the same \( f_L^* \) as in Fig. 1. As in Fig. 1, the large dots are plots of \( f_L^n(k) \) for \( k = 1, \ldots, 9 \). The 45-degree line \( y = a_k \) is shown because of the computations in Step 3 of the EFWA algorithm. Since \( n = 9 \), \( \lfloor (n+1)/2 \rfloor = 5 \), so that \( k = 5 \), which is why \( f_L^n(5) \) has been located at \( a_5 \). For this value of \( f_L^n(5) \) we fail the test in Step 3 [i.e., \( f_L^n(5) \) is not between \( a_5 \) and \( a_4 \) and as a result of Step 4 [i.e., \( f_L^n(5) < a_5 \)] we set \( last = 5 \). Returning to Step 2, we find \( k = 3 \) and then compute \( f_L^n(3) \), which is why there is a line with an arrow on it from \( f_L^n(5) \) to \( f_L^n(a_5) = f_L^n(3) \). For this value of \( f_L^n(3) \) we again fail the test in Step 3 [i.e., \( f_L^n(3) \) is not between \( a_3 \) and \( a_2 \) and as a result of Step 4 [i.e., \( f_L^n(3) > a_2 \)] we set \( first = 4 \). Returning to Step 2, we find \( k = 4 \) and then compute \( f_L^n(4) \), which is why there is an arrow on it from \( f_L^n(3) \) to \( f_L^n(a_4) = f_L^n(4) \). For this value of \( f_L^n(4) \) we pass the test in Step 3 [i.e., \( f_L^n(4) \) is between \( a_4 \) and \( a_3 \) ], stop and set \( f_L^* = f_L^n(4) \). This example, which is the same one as in Fig. 1, has taken the EFWA three iterations to converge whereas it took the KM algorithm two iterations to converge.

7 Experimental Results

In this section we present simulation results in which we compare the convergence numbers for the KM and EFWA algorithms. Because, for each \( \alpha \)-cut, both algorithms consist of two
Observe that:

1. The mean convergence number for the KM algorithm is approximately two, and it is \( \lceil (\ln n) - 1 \rceil \) for the EFWA algorithm, where \( \lceil \cdot \rceil \) denotes the first integer equal to or larger than \( \cdot \). The standard deviation of the convergence number for the KM algorithm is approximately 0.5, and it is approximately 1.2 for the EFWA algorithm.

2. According to central limit theory, the distribution of convergence numbers (for each \( n \)) is approximately normal with mean and standard deviation equal to the sample mean and standard deviation shown in Fig. 2. We can therefore use the sample mean + two times the sample standard deviation to evaluate the convergence numbers with 97.5% confidence. Consequently we are 97.5% confident that the KM algorithm converges in approximately three iterations, and that the EFWA algorithm converges in approximately \( \lceil (\ln n) + 1.4 \rceil \) iterations. We see, therefore, that the KM algorithm is computationally more efficient than the EFWA algorithm.

3. When \( n \geq 3 \), the KM algorithm needs a smaller number of iterations than does the EFWA algorithm. When \( n \leq 2 \), both algorithms need the same number of iterations.

4. The KM epsilon convergence numbers depicted in Fig. 3 agree with those obtained from (18) and (19), e.g. for \( n = 200 \), we observed from running the KM algorithm that \( f_L^*(0) = 4.4077 \), \( f_L^*(1) = 4.378361494 \), and \( f_L^* = 4.3783 \), so

\[
\delta = \left( f_L^*(1) / f_L^* - f_L^*(0) / f_L^* \right)
= 2.09 \times 10^{-3}
\]

and \( \ln \delta = -6.1706 \). For \( \epsilon = 0.01 \),

\[
\ln \left( \epsilon / (f_L^*(0) - f_L^*) \right) = -1.0784
\]

so that

\[
j = 1 + \ln \left( 1 + \ln \left( \epsilon / (f_L^*(0) - f_L^*) \right) / \ln \delta \right) / \ln 2
= 1 + \ln \left( 1 + [-1.0784 / -6.1706] / \ln 2 \right)
= 1.23
\]

---

\( ^4 \) Because iterations must be positive, this is a one-sided confidence interval.
Hence, \( j_c = 2 \) which equals the true mean convergence number of \( k = 2 \).

8 Conclusions

By connecting solutions from two different problems—the fuzzy weighted average (FWA) and the generalized centroid (GC) of an interval type-2 fuzzy set—we have arrived at a new \( \alpha \)-cut algorithm for solving the FWA problem, one that converges monotonically and super-exponentially fast. Our simulations demonstrate that for each \( \alpha \)-cut, convergence occurs (to within a 97.5% confidence interval) in three iterations. No faster \( \alpha \)-cut algorithm for solving this problem exists to-date.

If \( 2m \) parallel processors are available, we can use the new KM \( \alpha \)-cut algorithms to compute the FWA in three iterations (to within 97.5% confidence), because the calculations of \( f^L \) and \( f^U \) are totally independent, and all \( m \) \( \alpha \)-cut calculations are also independent.

Research is underway to extend the FWA from type-1 fuzzy sets to interval type-2 fuzzy sets, something we call the fuzzy-fuzzy weighted average (FFWA) [12]. The FWA plays a major role in computing the FFWA. We also believe that the FWA can be used to perform type-reduction for a general type-2 fuzzy set, something that we are also studying.

References


Figure 1: A graphical interpretation of the KM algorithm that computes $f_L^*$. The solid line shown for $y = a_k$ only has values at $a_1, a_2, ..., a_9$.

Fig. 2: A graphical interpretation of the EFWA algorithm that computes $f_L^*$. The solid line shown for $y = a_k$ only has values at $a_1, a_2, ..., a_9$.

Fig. 3: (a) Mean and (b) standard deviation of the convergence numbers when parameters are uniformly distributed.