# SFIST-based Fast Data Classification

**Balázs Tusor and Annamária R. Várkonyi-Kóczy**

Software Engineering Institute, John von Neumann Faculty of Informatics
Obuda University
Bécsi út 96/B, 1034 Budapest, Hungary

Department of Informatics
J. Selye University
3322 Bratislavská cesta, 945 01 Komárno, Slovakia

E-mail: tusor.balazs@uni-obuda.hu; koczya@ujs.sk

*Abstract: Fuzzy logic is a powerful tool in computer science, which has been used in countless applications since its conception in the late 80s. Numerous classifiers have been based on it, taking advantage of the flexibility and robustness against noise that is inherent in fuzzy systems. One such classifier called the "Sequential Fuzzy Indexing Tables Classifier" has been developed, to provide a fast and robust classification performance by combining the speed of indexing tables with the flexibility of fuzzy inference systems. One major disadvantage of it is its memory requirement that scales exponentially with the dimension size of the problem. To solve this problem, the authors have proposed the so-called Sequential Fuzzy Indexed Search Trees (SFIST) classifier that uses the same principle, but with a much smaller structure. In previous works, the authors have proposed two variants for the SFIST classifier, and both were shown to drastically reduce the required memory space compared to that of its predecessor, without any loss in classification performance. In this paper, a new, third variant is proposed that implements a hybrid approach between the first two, aiming to further improve the classification accuracy, without sacrificing too much operational speed.*

*Keywords: fuzzy inference; indexing tables; classification; search trees*

# 1 Introduction

Fuzzy logic [1] is a powerful tool in computer science, which has been used in countless applications since its conception in the late 80's. To mention some recent ones, fuzzy inference has been advantageously used in risk management in railway infrastructure projects [2], and fuzzy classifiers have been advantageously applied to implement alcohol classification [3] and medical applications [4].

In the simplest form of fuzzy inference-based classification, the knowledge acquired from the training data is expressed as a set of fuzzy rules, and to each rule a label is assigned (i.e., the class that most likely fits the samples adhering to the rule). This simplifies the classification process to simply evaluating each rule: calculating the fuzzy membership function (MF) values for each attribute, taking the minimum of the gained values (which shows how well the input sample fits the given rule), and of all the rules the one with the highest overall MF value provides the output of the classification (alongside the certainty of the results).

One key disadvantage of this simple method is that the number of fuzzy rules can be very large, making the evaluation process slow. In previous work, in order to solve this issue, the authors have proposed the so-called *Sequential Fuzzy Indexing Tables* (SFITs, [5]) classifier, which utilizes indexing tables to drastically reduce the required computation to a series of array accesses, but in return it requires large parts of the problem space to be stored in the memory (as a sequence of 2D arrays with increasing sizes), so its usage is limited to low dimensional problems.

To address this problem, the authors have proposed a new classifier called Sequential Fuzzy Indexed Search Trees (SFISTs, [6]). The base idea of the system is the same as that of the SFITs, i.e., the dimensional decomposition of the problem, as the search area in the problem space is being reduced attribute by attribute. The fuzzy MFs are stored as series of intervals. Triangular and trapezoidal fuzzy sets are utilized, so the intervals can either be plateaus (where the MF value is 1) or transitions (the overlapping region between two neighboring sets). To quickly find which interval a given input value falls into, self-balancing binary search trees (BSTs, like [7]) are used, so although their operation is slower than that of the SFITS, but only by a logarithmic order, while the amount of stored data is vastly reduced.

In previous works, the authors have proposed two variants for the SFIST classifier: the *basic* [6] and the *simplified* [8] variants. The former has been created to achieve a classification similar to that of the SFIT classifier (focusing on reducing the structure size), while the latter has been developed to enhance the speed of the basic SFIST. The two share the same architectural idea, with slight differences.

In this paper, a new, third variant is proposed that implements a hybrid approach between the first two, aiming to improve the classification accuracy, without sacrificing much speed.

The rest of the paper is as follows. In Chapter 2, the basic and simplified variants are described shortly, while in Chapter 3 the new variant is proposed: its architecture, training and evaluation algorithms. In Chapter 4 the classification performance of the proposed variant is showed and compared to that of the previous two variants, alongside an analysis regarding their computational complexities. Finally in Chapter 5, the paper is concluded and future work is described.

# 2 Previous Work – Basic and Simplified SFIST

## 2.1 Basic Sequential Fuzzy Indexed Search Trees

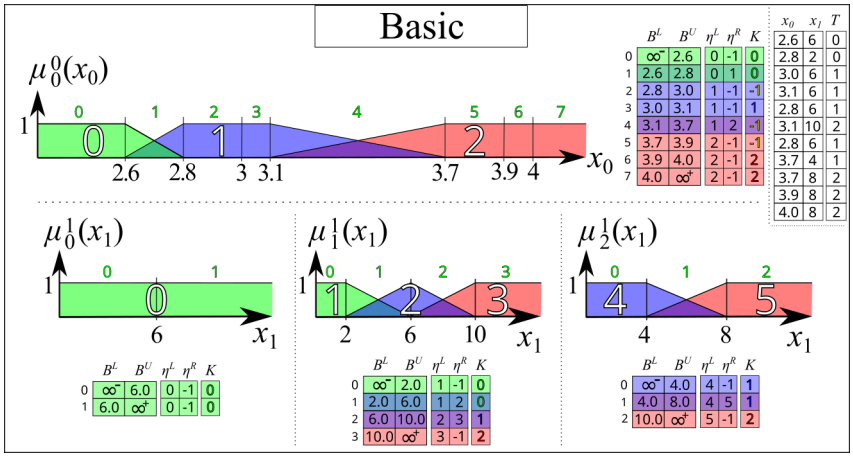


Figure 1
An example for the architecture of the basic SFIST variant

Figure 1 shows an example for a trained structure using the basic SFIST variant for a 2-dimensional problem. The data (with attributes $x_0$ and $x_1$, and class $T$) it has been trained with can be seen in the top right corner. The architecture consists of 2 layers (one for each attribute), where $\mu_j^i$ denotes the $j^{\text{th}}$ fuzzy MF in layer $i$. Each MF is divided into intervals (identified by the green numbers above them in the figure). For each interval $k$, the following descriptors are stored: $I(B_k^L, B_k^U, \eta_k^L, \eta_k^R, K)$

- $B_k^L$: the lower boundary value of interval $k$
- $B_k^U$: the upper boundary value of interval $k$
- $\eta_k^L$: the left index value of interval $k$
- $\eta_k^R$: the right index value of interval $k$
- $K$: the class label of interval $k$

These are stored in separate 1D arrays for each MF. Each interval has two boundary values, and can either depict an area belonging to a single fuzzy set, or two (neighboring ones).

**Remark:** in this research, only such alignments are considered. Each fuzzy set has an index value associated to it that shows which MF is needed to be evaluated in the next layer. These index values are stored in $\eta_k^L$ and $\eta_k^R$, respectively. In case of plateaus, only the left index is used, the right index is "-1". E.g., for input value

$x_0 = 2.7$, the BST (not pictured in the figure) in $\mu_0^0$ provides that the sought interval is #1, so in the next layer, $\mu_{\eta_1^L}^1 = \mu_0^1$ and $\mu_{\eta_1^R}^1 = \mu_1^1$ are to be evaluated.

Class value *K* for a given interval stores which class the training data that fall into the given interval belong to, or "-1" if the set of samples in question belong to multiple classes, serving as heuristics for the training algorithm to separate the areas more efficiently. The training is done incrementally, focusing on one given training sample *X* at a time. The input value is evaluated in the given MF, and depending on the interval it falls into, either the that interval is left as it is (if the class of the currently trained sample aligns with that of the interval), or changed (divided into two intervals, if their class do not match), then the training proceeds to the next layer. This is done until all training samples have been processed.

Let us consider a series of MFs in consecutive layers (one MF in each layer) a *route*. The evaluation algorithm traverses through the structure from the first to the last layer, evaluating the MFs on the routes indicated by the index values of the intervals its values fall into along the way, calculating the overall MF value as well. The output of the evaluation step is the class label of the route (acquired in the last layer, i.e., at the end of each route) that has the highest overall MF value.

**Remark:** the intervals at both ends (i.e., the leftmost one that starts at $\infty^-$, and the rightmost one that ends in $\infty^+$) of each MF are also restricted by an arbitrary $r$ range value, which has been shown to improve the classification accuracy. For more information on the basic variant, please see [6].

## 2.2 Simplified Sequential Fuzzy Indexed Search Trees

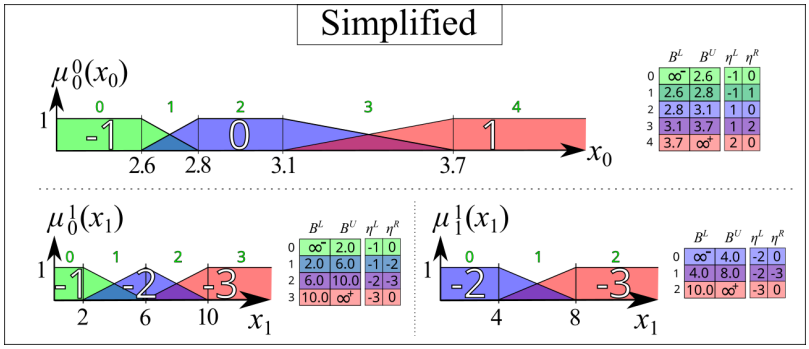


Figure 2

An example for the architecture of the simplified SFIST variant

The simplified variant has been developed for a faster evaluation step, which is achieved with significant reductions in the structure size. As it can be seen in Figure 2, the class labels are no longer stored in a dedicated array, but incorporated into the index values, where positive values denote an index value, while negative values

are for class labels. Furthermore, a route can end without needing to reach the last layer, as the expected label for it is already available in an earlier layer. This leads to fewer MFs, and an overall smaller structure. Due to this, the index value 0 here indicates plateaus (and thus, index value 1 is associated with the 0th MF ($\mu_0^1$) in the next layer, and index value -1 means class label 0). Lastly, while the basic variant implicitly stored the values gained from the training data, the simplified variant only stores the starting and ending points of each fuzzy set plateaus.

The training is done using all samples at once: for each MF in each layer, the training data values of the corresponding attribute are sorted and grouped. First, samples with the same attribute value are grouped together, then the groups that are close enough to each other value-wise and similar enough in class composition are united. The groups gained this way are appointed as plateau type intervals, and the areas between them as the transition type intervals.

To measure and quantify class-wise similarity, the concepts of *class distribution* and *class composition* are introduced. Let $\kappa$ be the number of classes. The *class distribution* of a given sample set $Q$ is an array that stores how many samples of each classes are represented in $Q$. For all j ($\in [0, \kappa - 1]$):

$$\Psi_j^Q = \sum_{\forall q \in Q} \{1 | T_q = j\} \tag{1}$$

The *class composition* value ($\psi^Q$) of a sample set $Q$ encodes the class label combinations in the set in a single number, by calculating the sum of the powers of two of the label number of each class that occurs in the set:

$$\psi^Q = \sum_{\forall q \in Q} (2^{T_q}) \tag{2}$$

This is basically the decimal value of the binary number where each bit is 1 if the class with the corresponding label is present in the set, and 0 otherwise.

For example, let set $Q$ consist of 6 samples, of which 1 sample is labelled as class #0, 3 samples as class #1, and 5 samples as class #3. The class composition of the set is $\psi^Q = 2^0 + 2^1 + 2^3 = 11$, and the class distribution is $\Psi^Q = [1, 3, 0, 5]$.

A set $Q_A$ is *compositionally part* of another set $Q_B$, if the power of two values that make up $\psi^{Q_A}$ are also part of the values that make up $\psi^{Q_B}$. E.g., $\psi^{Q_A} = 9$ (=$1001_B$ in binary) is compositionally part of $\psi^{Q_B} = 11$ (=$1011_B$), as $2^0$ and $2^3$ are both part of the sum that constitutes $\psi^{Q_B}$.

The *distance* between two class distributions of two sample sets $Q_A$ and $Q_B$ is gained from the Euclidean distance of their normalized distribution values:

$$D(Q_A, Q_B) = \sqrt{\sum_{k=0}^{\kappa-1} \left( \frac{\Psi_k^{Q_A}}{\sum_{j=0}^{\kappa-1} \left(\Psi_j^{Q_A}\right)^2} - \frac{\Psi_k^{Q_B}}{\sum_{j=0}^{\kappa-1} \left(\Psi_j^{Q_B}\right)^2} \right)^2} \tag{3}$$

These metrics are also used in the training of the newly proposed variant.

Overall, the vastly reduced structural size of the simplified variant significantly increases its evaluation speed, though at the cost of a bit of reduction in classification accuracy (as less parts of the problem space are explored to find the optimal solution). Furthermore, due to the same issue with the leftmost and rightmost intervals, similarly to that of the basic variant, a range value $r$ is used to limit the evaluation regarding values that fall out of the known domain area of the trained MFs. For more information on the simplified variant, please see [8].

# 3 Complete Sequential Fuzzy Indexed Search Trees

## 3.1 Architectural Changes

The Complete SFIST variant has been developed as a hybrid approach between the previous two: creating an as detailed representation of the problem space as possible given the training data (as in the case of the basic variant), but by analyzing the whole data value set for each MF and creating groups (like the simplified variant) based on sample set similarity. Moreover, the range issue on the leftmost and

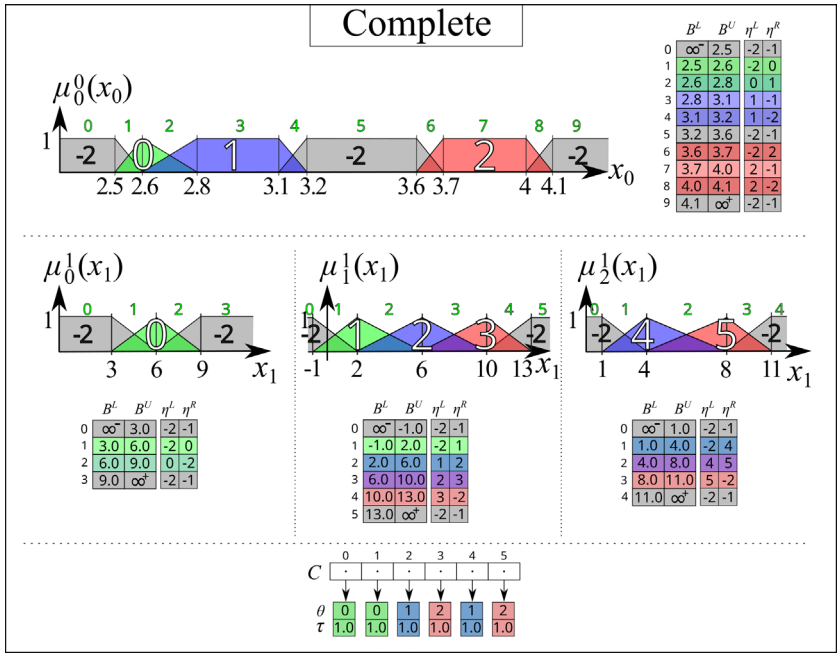


Figure 3

An example for the architecture of the complete SFIST variant

rightmost intervals that caused problems for the previous two variants have been addressed by appointing a so-called *unknown* class, which is indexed as "-2" and depicts an area for which there is no information based on the training dataset. The areas (fuzzy sets) for this class are also stored as intervals.

This can be seen in Figure 3, where the intervals belonging solely to the unknown class are depicted with gray color. Similarly, to the basic architecture, plateaus are indicated with the value "-1". The last major change is the introduction of the class-rate list array $C$ in the last layer, where each array element is a list that contains at least one element that shows which classes ($\theta \in [0, \kappa - 1]$) the index has been associated with in the last layer, according to the training data, and at which rate ($\tau \in [0, 1]$). E.g., if the list contains label #1 with rate 0.75 and #2 with rate 0.25 (meaning that ¾ of the training samples that ended in this interval during the training are of class #1, and ¼ are of class #2), then the route is calculated distinctly for class #1 and class #2 (weighing the overall MF values further with the rate). This results in a much more through classification, as it accounts for possible inconsistencies (e.g., caused by noise) in the training data.

## 3.2 The Training Algorithm

The idea behind the training algorithm of the complete variant is similar to that of the simplified one, but without the sorting step. This is achieved by dividing the value domain of each attribute into regions, filling the regions using to the values of the training data, then analyzing the regions in an ascending order to group them together, based on how different the subsequent regions are both in terms of their data values and class distributions. The groups made this way are appointed as the intervals that make up the fuzzy sets.

### 3.2.1 The Architecture of the Auxiliary Structures

Since after the first layer only a given subset of samples are regarded for the creation of each MF (e.g., to train $\mu_2^1$, only the samples falling into the intervals of fuzzy set #2 (between values 3.7 and 4) in $\mu_0^0$ are needed (namely, #7,#8,#9 and #10)), the indices of these samples are stored in index lists. Remark: An index list in this research is a simple list of integer values. In the following, let $L^I$ denote an index list, and $L^G$ be a list of index lists. In each layer, each list element of $L^G$ provides the sample indices for the individual MFs (so the number of MFs in layer *i* is decided by the number of list elements in $L^G$).

At the beginning of the training, $L^G$ contains a single index list that contains all indices from 0 to *P*-1, where *P* is the number of training data samples. It is used to create the single MF in the first layer. During the training a new list, $L^{G\prime}$ is created and filled with index lists, making an index list for each group that is found when analyzing the values of the training data. Figure 4 shows an illustration for this,

where the initial $L^G$ only contains a single index list $L^I_0$ with 11 samples, and the 3 newly made fuzzy sets divide this list into 3 smaller lists, based on which fuzzy set the given sample is used to create.

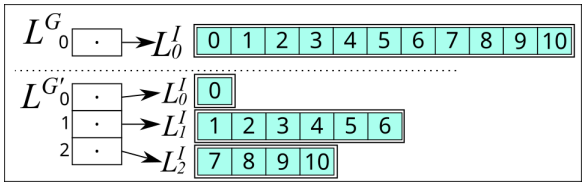


Figure 4

An illustration for the index list separation for MF #0 in layer 0, using the data from Figure 1. The initial index list in $L^G$ is separated into 3 new index lists in $L^{G'}$, which provides a quick way to find which samples are needed for each MF in layer 1

In order to create the fuzzy sets in a given MF in layer $i$, the $i^{\text{th}}$ attribute values of the samples in the corresponding index list are analyzed. The value domain is divided into regions using an arbitrary granularity parameter $\gamma$ (which is used to set how different two values are needed to be to fall into different regions). For each region, the following data is stored:

- $v^L$: The *lowest value* of the samples represented in the region
- $v^H$: The *highest value* of the samples represented in the region
- $\psi$: The *class composition* of the region (see: Chapter 2.2)
- $\Psi$: The *class distribution* array of the region (see: Chapter 2.2)
- $W$: A rudimentary timestamp, i.e., the index of the MF for which the region was last modified
- $L^I$: The list of indices of the data samples used to set up the region

To reduce the computational overhead from deleting old regions and creating new ones for each MF, the regions are created once and reused in the same layer. When a given region is analyzed, its $W$ parameter shows if its values are up to date or not (in the latter case, it is simply overwritten with new data).

An example for the regions can be seen in Fig. 5, which depicts the region array $R$ for the first attribute (layer #0) (of the same example presented in the previous figures). The structure can be implemented through simple 1D ($W$, $\psi$, $v^L$ and $v^H$) and 2D ($\Psi$ and $L^I$) arrays (or lists), or as an array of individual structures (using the Object-Oriented Paradigm). The former has the advantage of being simple to implement, while the latter has a potentially lower memory requirement (the regions that are not filled with data simply do not have to be created, which is indicated with the gray "-1" boxes in the figure). In our implementation, we use the latter. Remark: the granularity in the example: $\gamma = 0.2$.

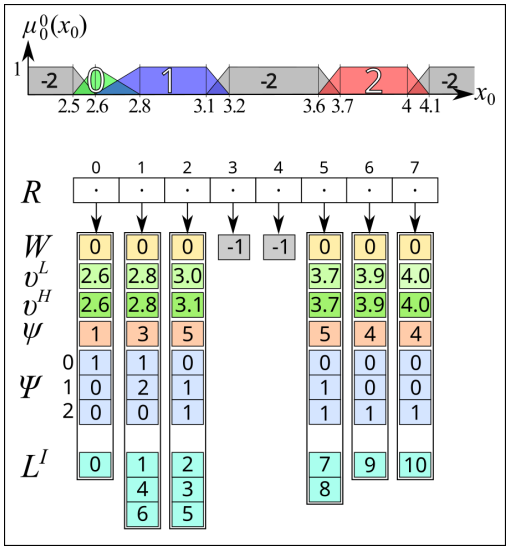


Figure 5
The regions created for the MF #0 of layer #0, using the data from Figure 1

### 3.2.2 Setting up the Structure

The training algorithm in each layer $i$ ($\in [0, N-1]$) starts by analyzing the data in the whole dataset to find the minimum and maximum values for attribute $i$. From the two values and granularity parameter $\gamma$, the size of $R$ can be calculated:

$$S_i = \left\lfloor \frac{\max_{p \in P} X_{p,i} - \min_{p \in P} X_{p,i}}{\gamma} \right\rfloor + 1 \tag{4}$$

The indices (i.e., the addresses) of the region array $R$ are calculated using a simple linear mapping, for which the scaling factors ($a_i$) and the biases ($b_i$) are:

$$a_i = \frac{S_i}{\max_{p \in P} X_{p,i} - \min_{p \in P} X_{p,i}} \tag{5}$$

$$b_i = \min_{p \in P} X_{p,i} \tag{6}$$

where $X_{p,i}$ is the $i^{\text{th}}$ attribute value of sample $p$.

Using the calculated values, the $S_i$-long region arrays are set up at the beginning of the processing of each layer $i$.

Aside from creating the intervals for the MFs in the current layer, the secondary goal is creating the group list for the next layer (let us denote it with $L^{G'}$).

### 3.2.3 Region Construction

The training algorithm goes through each index list $L^{I*}$ in $L^G$. Let $q$ denote the sequence number of the currently processed list, which can be used to see if a region is up to date or not.

For each index value $p$ in $L^{I*}$, first the index of the region into which each sample belongs to is calculated:

$$z_p = \lfloor (X_{p,i} - b_i) \cdot a_i \rfloor \tag{7}$$

The group distribution is set to the class of sample $p$ if the region is not up to date, otherwise it is updated:

$$\Psi_{z_p,t_p} = \begin{cases} 1, & if\ W_{z_p} < q \\ \Psi_{z_p,t_p}, & if\ W_{z_p} = q \end{cases} \tag{8}$$

where $t_p$ is the class of sample $p$. In the former case, the distribution value of the other classes are set to 0. The class composition is updated similarly:

$$\psi_{z_p} = \begin{cases} 2^{t_p}, & if\ W_{z_p} < q \\ \psi_{z_p} + 2^{t_p}, & if\ W_{z_p} = q \end{cases} \tag{9}$$

The lowest and highest values for region $z$ is also set:

$$v_z^L = \min(v_z^L, X_{p,i}) \tag{10}$$

$$v_z^H = \max(v_z^L, X_{p,i}) \tag{11}$$

If the region is up to date, then $p$ is added to $L_z^I$, otherwise a new list is started with $p$ as its only item.

Finally, $W_z$ is set to signal that the region is up to date:

$$W_z = q \tag{12}$$

Furthermore, the smallest ($\hat{z}^L$) and highest ($\hat{z}^H$) index values are stored, so in the region analysis step the algorithm only needs to regard the regions in this range:

$$\hat{z}^L = \min(\hat{z}^L, z_p) \tag{13}$$

$$\hat{z}^H = \max(\hat{z}^H, z_p) \tag{14}$$

### 3.2.3 Region Analysis and Grouping

With the regions built, the next step is analyzing their data and put them into groups, based on the distance between their values and their class distributions. Let $m$ be the current number of MFs in the current layer, which is initially 0. In each MF, the intervals are added consecutively (starting from 0), so in the following the addition of a new interval is simply denoted by $I(B^L,\ B^U,\ \eta^L,\ \eta^R)$, where $B^L$ and $B^U$ are

the lower and upper boundaries, while $\eta^L$ and $\eta^R$ are the left and right indices, respectively.

During the analysis of the regions, first the new MF is created. The first interval is already known: it is a plateau from $\infty^-$ to the $\delta$ range of lowest value of the leftmost non-empty region ($v_{\hat{z}^L}^L$), for which the index is provided by $\hat{z}^L$. This interval is of the unknown class, so its class ID is "-2"):

$$I(\infty^-,\ v_{\hat{z}^L}^L - \delta,\ -2, -1) \tag{15}$$

Then, the next interval is always a transition between the right boundary of the previous interval to the lowest value of the leftmost non-empty region. Let $g$ be the ID number of the group that is currently being built in the current MF, which is initially 0 (and its value is reset at the beginning of the creation of each MF):

$$I(v_{\hat{z}^L}^L - \delta,\ v_{\hat{z}^L}^L,\ -2,\ g = 0) \tag{16}$$

Let the overall distribution of the currently built group be $\overline{\Psi}$ (which is initially set to the distribition of the leftmost region $\hat{z}^L$, and the overall composition is $\bar{\psi}$ (similarly, set to the composition of $\hat{z}^L$):

$$\overline{\Psi} = \Psi_{\hat{z}^L} \tag{17}$$

$$\bar{\psi} = \psi_{\hat{z}^L} \tag{18}$$

Furthermore, let $\overline{L^I}$ denote the index list of the currently built group, initially set to the index list of $\hat{z}^L$:

$$\overline{L^I} = L_{\hat{z}^L}^I \tag{19}$$

Finally, the lowest and highest values (i.e., the real-valued boundaries of the group) are stored, which is also initially set to the corresponding values of $\hat{z}^L$:

$$\overline{v^L} = v_{\hat{z}^L}^L \tag{20}$$

$$\overline{v^H} = v_{\hat{z}^L}^H \tag{21}$$

The region analyzer algorithm starts from region $\hat{z}^L + 1$, and ends after region $\hat{z}^H$ has been processed. In each iteration $z$ ($\in [\hat{z}^L + 1, \hat{z}^H]$):

- If region $z$ is not up to date ($W_z < q$), then move on to the next region.
- Otherwise, the region is relevant, so the next step is checking if the lowest value of region $z$ is close enough to the highest value of the current group, i.e., if it is within a range of $\delta$: ($v_z^L \leq \overline{v^H} + \delta$)
    - If it is close enough, then it is checked if either of them compositionally part of the other, in which case their distribution distance is checked against the allowed similarity measure $\rho$: $D(\bar{\psi}, \psi_z) < \rho$?

- If one of them is compositionally part of the other, and their similarity measure is smaller than $\rho$, then region $z$ is added to the group. The highest value ($\overline{v^H}$) of the group is set to the highest value of the region, the new class composition is set to the larger value between the two, and the class distribution values are simply added to that of the group:

$$\overline{v^H} = v_z^H \tag{22}$$

$$\bar{\psi} = \max(\bar{\psi}, \psi_z) \tag{23}$$

$$\bar{\Psi}_t = \bar{\Psi}_t + \Psi_{z,t}, \forall t \in [0, \kappa] \tag{24}$$

  Finally, the index list of the region $z$ ($L_z^I$) is added to the end of index list of the group ($\overline{L^I}$).

- If neither of them is compositionally part of the other, or their similarity value is not low enough, then the currently built group is finished, the intervals of fuzzy set $g$ is ready to be set up. If the lowest and highest values of the group are not the same value ($\overline{v^L} < \overline{v^H}$), then a plateau interval is inserted:

$$I(\overline{v^L}, \overline{v^H}, g, -1) \tag{25}$$

  Otherwise, the set is a triangular fuzzy set, so no plateau is necessary.

  After that, a transition interval is inserted:

$$I(\overline{v^H}, v_z^L, g, g+1) \tag{26}$$

  The index list of the group ($\overline{L^I}$) is added to the group list of the next layer ($L^{G\prime}$), and group index is incremented ($g = g + 1$).

  The parameters of the new group are set to the parameters to the region:

$$\overline{v^L} = v_z^L \tag{27}$$

$$\overline{v^H} = v_z^H \tag{28}$$

$$\bar{\psi} = \psi_z \tag{29}$$

$$\bar{\Psi}_t = \Psi_{z,t}, \forall t \in [0, \kappa] \tag{30}$$

$$\overline{L^I} = L_z^I \tag{31}$$

- If the lowest value of region $z$ is too far away, then the current group is finished. If the lowest and highest values of the group are not the same ($\overline{v^L} < \overline{v^H}$), then a plateau is inserted using (25).

After that, distance between $\overline{v^H}$ and $v_z^L$ determines if an additional fuzzy set is needed to be added (representing the unknown class):

- If the highest value of the group is within range of the lowest value of region $z$ (i.e., their ranges overlap with each other, $\overline{v^H} + \delta \geq v_z^L - \delta$), then simply a transition interval is inserted between them using (26), then a new group is started (applying (27)-(31)).
- Otherwise, an additional fuzzy set is added, a transition between $\overline{v^H}$ and $\overline{v^H} + \delta$:

  $$I(\overline{v^H}, \overline{v^H} + \delta, g, -2) \tag{32}$$

  Then a plateau is added between $\overline{v^H} + \delta$ and $v_z^L - \delta$ if these two values are not the same (otherwise the fuzzy set is triangular, so no plateau is added):

  $$I(\overline{v^H} + \delta, v_z^L - \delta, -2, -1) \tag{33}$$

  After that, a transition is inserted between $v_z^L - \delta$ and $v_z^L$:

  $$I(v_z^L - \delta, v_z^L, -2, g + 1) \tag{34}$$

  Finally, the index list of the group is added to the group list of the next layer, the group index is incremented, and a new group is started (using (27)-(31)).

After the last index list is processed in the given layer *i*, then if it is not the last layer ($i < N - 1$), then the algorithm continues on to the next layer, with the newly created group list:

$$L^G = L^{G\prime} \tag{35}$$

In the last layer, class-rate array $C$ is created, its length is provided by the number of index lists in $L^G$. The samples indexed by each index list $L_i$ are regarded, and for each class that is represented by them a new item is added to $C_i$, noting the ID of the class and at what rate is the given class represented among the samples of $L_i$.

## 3.3 The Evaluation Algorithm

The evaluation algorithm is more or less the same as that of the basic and simplified SFIST variations, adapted to the architecture of the proposed variation. The goal of the (recursive) algorithm is to go through the structure from the first to the last layer, exploring all routes (i.e., MF combinations in subsequent layers connected by index values), calculating the fuzzy MF values for each, and returning the class ID belonging to the route with the highest overall MF value. Let $Y$ be a $\kappa$-long array that stores the largest overall MF value for each class, and $\bar{\mu}$ the overall MF value on a given route. At the beginning of each route, the former is all zeros, while the value of the latter is 1.

In each MF on the route (based on the type of the interval the corresponding attribute value of the input data falls into) a given route taken can split into two, following the left and right indices of the interval:

- If the interval is a plateau ($\eta^R = -1$) and it belongs to the unknown class ($\eta^L = -2$), then the route is finished, since the MF value is 0.
- If the interval is a plateau with class ID #0 or more (so it does not belong to the unknown class), then the MF value for that attribute is 1, the evaluation continues in the next layer with the MF ID given by the left index value ($\eta^L$).
- If the interval is a transition, then the MF value is calculated. In our implementation, we applied a simple linear transition:

  $$\mu(x) = 1 - \frac{x - B^L}{B^H - B^L} \tag{36}$$

  If $\eta^L > 0$, then the examination continues in the next layer with MF ID $\eta^L$ and overall value $\bar{\mu} = \min\big(\bar{\mu},\ \mu(x)\big)$, and after all new routes from it (given that further branching may occur) are finished, the same is done considering its other side (taking MF ID $\eta^R$ and overall value $\bar{\mu} = \min\big(\bar{\mu},\ 1 - \mu(x)\big)$).

In the last layer, the MF value is calculated and the acquired index value (or values, in case of a transition interval) is used to find which label list in *C* is to be regarded.

For plateau type intervals, the class array *Y* is updated for each *k* list element in list $C_{\eta^L}$ (if $\eta^L > 0$):

$$Y_{\theta_k} = \max\big(Y_{\theta_k},\ \min(\tau_k, \bar{\mu})\big), \forall k \in C_{\eta^L} \tag{37}$$

For transition type intervals, *Y* is similarly updated for both indices, for each *k* list element in list $C_{\eta^L}$ if $\eta^L > 0$, and in $C_{\eta^R}$ if $\eta^R > 0$:

$$Y_{\theta_k} = \max\big(Y_{\theta_k},\ \min(\tau_k \cdot \mu(x), \bar{\mu})\big), \forall k \in C_{\eta^L} \tag{38}$$

$$Y_{\theta_k} = \max\big(Y_{\theta_k},\ \min(\tau_k \cdot \big(1 - \mu(x)\big), \bar{\mu})\big), \forall k \in C_{\eta^R} \tag{39}$$

After all routes have finished, the output of the system is the index belonging to the highest value of *Y*, i.e., the label of the class that is the most similar to the inputs:

$$y = \underset{k}{\operatorname{argmax}}(Y_k), \forall k \in [0, \kappa - 1] \tag{40}$$

# 4 Experimental Results

The classification accuracy of the proposed SFIST variant, as well as that of the previous two variants have been investigated and compared on various benchmark data sets, using an average laptop PC (Lenovo Legion 7 16ACHg6, AMD Ryzen™ 9 5900HX CPU, 32GB RAM, Nvidia GeForce RTX 3080 16GB), implemented in MS Visual Studio Community 2022, C# .NET framework 4.7.2.

## 4.1 Experimental Results on the Iris Dataset

In the first set of experiments, the variants have been tested on the Iris dataset [10], which is a 3-class problem with $N$=4 attributes and $P$=150 overall samples. The setting of the parameters of each variant is done, by separating the dataset to a training and a testing sets at 90:10 ratio, then training and evaluating each variant 100 times for each value in a given range. The $r$ range value is investigated for the basic and similar methods, while the class distribution distance threshold ($\delta$) and the group distance threshold ($\rho$) are regarded for the simplified and complete variants. For the complete variant the granularity parameter ($\gamma$) is checked first (evaluated for different values, and the one with the highest average classification accuracy is chosen), then $\delta$ are $\rho$ evaluated for different values based on the chosen $\gamma$ value.

After the proper parameters have been set, the data has been separated into a training and a test set in 70:30, 80:20 and 90:10 ratios randomly, then each variant have been trained and evaluated for each. This has been repeated for 1000 times and the results have been averaged, which can be seen in Table 1.

Table 1
Classification accuracy on the Iris dataset (and the chosen parameter values)

| | **Basic** | **Simplified** | **Complete** |
|---|---|---|---|
| **70:30** | *85.67%* | *81.13%* | ***96.27%*** |
| **80:20** | *86.47%* | *82.8%* | ***96.84%*** |
| **90:10** | *89.93%* | *86.33%* | ***97.69%*** |
| **Parameters** | $r = 0.9$ | $r = 0.2$<br>$\rho = 0.01$<br>$\delta = 0.2$ | $\gamma = 1$<br>$\rho = 1.0$<br>$\delta = 1.3$ |

Interestingly, although the parameters with the same name serve more or less the same function in each variant, the values that produce the highest classification accuracy can be very different, as it can be seen in the table (which is the result of the difference in operation). Nevertheless, the tests have shown that the basic variant returns decent results (85.67-89.93%), while the simplified variant underperforms that by ~4%, and the complete variant overperforms both (96.24 - 97.69%).

One of the big reasons behind the lackluster performance of the first two has been found to be that they are very much dependent on the order of the attributes. By changing the training (and evaluation) order of the attributes of this particular problem slightly (evaluating attr. #3 first, then #0, #1 and #2) leads to the results seen in Table 2, where performance of the other two variants have improved quite a lot, while that of the complete variant stayed roughly the same.

Table 2
Classification accuracy on a slightly modified Iris dataset (with changed attribute order)

| | **Basic** | **Simplified** | **Complete** |
|---|---|---|---|
| **70:30** | *92.15%* | *92.21%* | ***96.61%*** |
| **80:20** | *92.49%* | *92.97%* | ***96.79%*** |
| **90:10** | *92.93%* | *93.4%* | ***97.69%*** |

Regarding the average structure sizes, a comparison can be seen in Table 3, as well as the required average times for the training and testing. The basic variant have created much more intervals on average than the other two, resulting in a much larger sized structure, and thus, longer evaluation time. On the other hand, the sorting step in the simplified variant have made its training much longer than that of the other two.

Table 3
Structure size and required time comparison on the Iris dataset

| | **Basic** | **Simplified** | **Complete** |
|---|---|---|---|
| **Avg. total numbers of MFs** | *23.48* | *6.07* | ***6*** |
| **Avg. total number of intervals** | *211.08* | *26.08* | ***29.92*** |
| **Estimated memory requirement** | *~9.08 KB* | *~1.03 KB* | ***~1.2 KB*** |
| **Avg. training time per sample** | *1.78 µs* | *3.11 µs* | *0.81 µs* |
| **Avg. evaluation time per sample** | *2.67 µs* | *2 µs* | *1.33 µs* |

## 4.2 Experimental Results on the WBC Dataset

The Wisconsin Breast Cancer (WBC, [11]) dataset presents a 2-class problem with

Table 4
Classification accuracy on the WBC dataset (and the chosen parameter values)

| | **SFIT** | **Basic** | **Simplified** | **Complete** |
|---|---|---|---|---|
| **70:30** | *94.04%* | *94.21%* | *94.63%* | ***95.6%*** |
| **80:20** | *93.72%* | *94.8%* | *95.04%* | ***95.76%*** |
| **90:10** | *94.7%* | *95.6%* | *95.28%* | ***95.98%*** |
| **Parameters** | - | $r = 0.1$ | $r = 0.1$<br>$\rho = 0.4$<br>$\delta = 0.2$ | $\gamma = 1$<br>$\rho = 1.0$<br>$\delta = 6.5$ |

*N*=9 attributes and *P*=699 samples. Similarly, to the previous tests, first the parameter values have been investigated, then a more thorough testing was done using the chosen values. The results can be seen in Table 4, where the complete variant outperformed the other two again. For reference, a comparison to the predecessor SFIT classifier is also included, all 3 variants have provided a better accuracy.

Table 5 shows the structure and required time comparisons between the variants. Interestingly, in this dataset the simplified variant could provide roughly the same classification accuracy as the other two with only a fraction of the structure size.

Table 5
Structure size and required time comparison on the WBC dataset

| | **Basic** | **Simplified** | **Complete** |
|---|---|---|---|
| **Avg. total numbers of MFs** | *198.45* | *34.25* | ***85.8*** |
| **Avg. total number of intervals** | *770.04* | *127.87* | ***375.03*** |
| **Estimated memory requirement** | *~33.1 KB* | *~5.01 KB* | ***~14.75 KB*** |
| **Avg. training time per sample** | *1.5 μs* | *2.73 μs* | *1.41 μs* |
| **Avg. evaluation time per sample** | *5 μs* | *2.35 μs* | *1.91 μs* |

## 4.3 Experimental Results on the Seismic Bumps Dataset

The Seismic Bumps [12] dataset presents a 2-class problem with *N*=15 attributes (originally 19, but the ones containing only zeros have been removed as they do not contribute to the classification) and *P*=2584 samples. Aside from being larger than the previously examined ones, this dataset also features attributes that differ drastically in domain range (some are binary categories, others are in the 5–6-digit range).

The results can be seen in Table 6, where the complete variant outperformed the other two again (by about 4%).

Table 6
Classification accuracy on the Seismic bumps' dataset (and the chosen parameter values)

| | **Basic** | **Simplified** | **Complete** |
|---|---|---|---|
| **70:30** | *90.26%* | *88.62%* | ***94.12%*** |
| **80:20** | *90.22%* | *88.63%* | ***94.08%*** |
| **90:10** | *90.29%* | *89.01%* | ***94.25%*** |
| **Parameters** | $r = 6.5$ | $r = 1.4$<br>$\rho = 0.5$<br>$\delta = 6.5$ | $\gamma = 1$<br>$\rho = 0.4$<br>$\delta = 7.5$ |

However, the effect of the varied attribute value domain ranges have caused the size of the structure of the complete variant to increase to a much larger extent than that of the other two, as Table 7 shows. This can be mitigated to some extent if the order

of the attributes is changed, taking the ones with the largest average values to the front of the attribute order have reduced the structure size by 1/5 (marked with *), showing that the proposed variant is still sensitive to the order of attributes, though in a different way than the other two.

Table 7
Structure size and required time comparison on the Seismic bumps' dataset

| | **Basic** | **Simplified** | **Complete** | **Complete*** |
|---|---|---|---|---|
| **Average total numbers of MFs** | *1722.64* | *26.83* | ***25210.65*** | ***5237.95*** |
| **Total Number of intervals** | *8312.86* | *491.5* | ***106846.7*** | ***26747.68*** |
| **Estimated memory requirement** | *~357.2 KB* | *~19.21 KB* | ***~4.08 MB*** | ***~1.28 MB*** |
| **Training time** | *59.65 µs* | *22.49 µs* | ***494.36 µs*** | ***280.34 µs*** |
| **Evaluation time** | *7.62 µs* | *2.79 µs* | ***3.45 µs*** | ***3.28 µs*** |

This also highlights the main issue with the complete variant in its presented form: it applies its parameters uniformly to all attributes, which works fine if all attributes in a problem are in the similar value range, but less so in ones that are very different.

## 4.5 Computational Complexity

The computational complexity of evaluating one input sample is the same for all the three variants: $O(N \cdot \log_2 M)$, where $M$ is the average number of intervals in the MFs (given that in each of the $N$ layers the interval is found in $\log_2 M$ steps, using the corresponding BST). In comparison, the predecessor SFIT classifier has a complexity of $O(N)$, so while the SFIST classifier is slower by a logarithmic order, but its speed still only linearly scales with the dimension number of the problem space.

There is a bigger difference considering the complexity of the training phase. The basic variant evaluates each of the $P$ training samples in each layer and changes the MF if necessary, thus has a complexity of $O(P \cdot N \cdot \log_2 M)$ (where $M=1$ in the beginning, but is increasing as more intervals are added). The simplified variant sorts the $P$-long arrays of each of the $N$ attribute values, which results in a complexity of $O(N \cdot P \cdot \log_2 P)$. Given that $P$ is typically larger than $M$, this usually means a slower training. The proposed complete variant substitutes the sorting step with the filling and analysis of a potentially large value array: $O\big(N \cdot (P + \bar{D})\big)$, where $\bar{D}$ is the average size of the value domains of the attributes of the given data. This shows why problems with attributes that take values from large ranges (such as the Seismic Bumps dataset) result in a much slower training.

## Conclusions

In this paper, a new variant is proposed to the Sequential Fuzzy Indexed Search Trees (SFIST) classifier. It is a hybrid approach between the two previously developed ones, focusing on increasing its classification accuracy. Its performance have been evaluated using benchmark datasets and compared to that of the previous variants, showing that it surpassed both of them. On the other hand, the new variant tends to generate a more bloated structure for problems that have attributes with large value domains.

In future work we will investigate the possibility of improving the proposed variant, by tailoring the parameters (namely, the granularity parameter which directly influences the size of the structure) for each attribute, thus potentially reducing the size of the resulting classifier. Furthermore, we will explore the effects of using different fuzzy set shapes (e.g., Gaussian) on overall performance.

## Acknowledgement

Supported by the ÚNKP-23-4-II-OE-59 New National Excellence Program of the Ministry for Culture and Innovation, from the source of the National Research, Development and Innovation Fund.

## References

[1] L. A. Zadeh, “Fuzzy logic,” Computer, Vol. 21, No. 4, pp. 83-93, 1988

[2] D. Macura, M. Laketić, D. Pamučar, D. Marinković, "Risk Analysis Model with Interval Type-2 Fuzzy FMEA – Case Study of Railway Infrastructure Projects in the Republic of Serbia," Acta Polytechnica Hungarica, Vol. 19, No. 3, 2022

[3] S. N. Shahbazova and D. Ekmekci, "An Input-weighted, Multi-Objective Evolutionary Fuzzy Classifier, for Alcohol Classification," Acta Polytechnica Hungarica, Vol. 19, No. 10, 2022

[4] A. Czmil, “Comparative study of fuzzy rule-based classifiers for medical applications,” Sensors, Vol. 23, No. 2, p. 992, 2023

[5] A. R. Várkonyi-Kóczy, B. Tusor and J. T. Tóth, “A multi-attribute classification method to solve the problem of dimensionality,” In: R. Jabłonski, R. Szewczyk, (eds.) Recent Global Research and Ed.: Techn. Challenges, AISC, Vol. 519, Springer, Cham, pp. 403-409, 2017

[6] B. Tusor, O. Takáč, Š. Gubo, and A. R. Várkonyi-Kóczy, “Fuzzy Inference with Sequential Fuzzy Indexed Search Trees," In International Conference on Global Research and Education, pp. 294-309, Cham: Springer Nature Switzerland, 2023

[7] S. A. Senbel, “Interesting exercise to demonstrate self-balancing trees,” Journal of Comp. Sci. in Colleges, Vol. 38, No. 3, pp. 185-185, 2022

[8] B. Tusor, A. R. Várkonyi-Kóczy, and Š. Gubo, “Improved Sequential Fuzzy Indexed Search Trees for Fast Classification," In 2024 IEEE Int. Symposium on Medical Measurements and Applications (MeMeA), pp. 1-6, 2024

[9] D. Dua, C. Graff, “UCI Machine Learning Repository,” http://archive.ics.uci.edu/ml (accessed Mar. 1, 2024)

[10] A. Unwin and K. Kleinman, “The iris data set: In search of the source of virginica,” Significance, Vol. 18, No. 6, pp. 26-29, 2021

[11] O. L. Mangasarian, W. H. Wolberg, “Cancer diagnosis via linear programming,” SIAM News, Vol. 23, No. 5, pp. 1-18, 1990

[12] M. Sikora, and L. Wrobel, “Seismic-bumps,” UCI Machine Learning Repository, 2013, https://doi.org/10.24432/C5W902