Recognition of Online Handwritten Mathematical Expressions

Utpal Garain and B. B. Chaudhuri, *Fellow, IEEE*

***Abstract*—This paper aims at automatic understanding of online handwritten mathematical expressions (MEs) written on an electronic tablet. The proposed technique involves two major stages: symbol recognition and structural analysis. Combination of two different classifiers have been used to achieve high accuracy for the recognition of symbols. Several online and offline features are used in the structural analysis phase to identify the spatial relationships among symbols. A context-free grammar has been designed to convert the input expressions into their corresponding TEX strings which are subsequently converted into MathML format. Contextual information has been used to correct several structure interpretation errors. A new method for evaluating performance of the proposed system has been formulated. Experiments on a dataset of considerable size strongly support the feasibility of the proposed system.**

***Index Terms*—Interpretation of two-dimensional structures, mathematical expression (ME), multiple-classifier system, online character recognition, performance evaluation.**

I. Introduction

MATHEMATICAL expressions (MEs) form an essential part of scientific and technical documents. There are several ways to input MEs into digital documents. So far the popular way to enter MEs is either in a linear format (e.g., TEX), or by using a structured editor (e.g., equation editor available with MS-Word). An alternative way is to write MEs by hand and employ a smart system that automatically interprets and enters them into the document under preparation. This paper is motivated toward this end.

Recognition of online handwritten MEs has two major aspects: 1) symbol recognition and 2) interpretation of two-dimensional (2-D) structures. Design of a symbol recognizer is a difficult task as it has to classify a large number of symbols like Arabic numerals, English alphabet, Greek symbols, mathematical signs, and symbols. This set becomes even larger if characters from non-English script are used, e.g., in Indian context, one may use numerals and characters of scripts like Devanagri (Hindi), Bangla, etc. Moreover, ME symbols vary in size to a large extent. Operators like integration, square root, sum or product, etc. can be quite large in size whereas, very small symbols like dot, comma, colon, etc. also appear in MEs. The same symbol can appear in different sizes in different context (e.g., script or limit symbols). Furthermore, writers have their own writing style giving different shapes for the same character. Many writers tend to connect and abbreviate the strokes of multistroke characters. Significant stroke number and order variation is observed with writer variation.

The structure of a ME can be significantly more complex than that of normal text lines. The spatial relationship among symbols are crucial to the interpretation of the expression. This means that even if all the characters are correctly recognized, there still remains the nontrivial problem of interpreting the two-dimensional (2-D) structure of an expression. Symbols use spatial relationships to indicate logical relationships among them. For example, structures like superscripts, subscripts, implied multiplication, matrix, etc. are indicated implicitly by the geometric layout of operands. Given a spatial relationship between two symbols, it is difficult to determine the logical relationship between them. Compared to printed MEs, the ambiguity of spatial relationships is substantially increased for handwritten versions. Moreover, several symbols ("horizontal line segment," "dot," etc.) have multiple meaning depending upon the context.

There exist several research efforts toward ME recognition. Existing approaches can be found in survey reports [1], [2] and are summarized in the next section. These surveys reveal that the studies dealing with processing of online handwritten MEs are few in number and additional research is needed to develop systems giving commercial level accuracy. The purpose of this paper is to describe an improved system for understanding online handwritten MEs. The present study differs from the previous ones in the following ways:

1) recognition of symbols involves two classifiers to capture wide variations in shape and size of the large number of ME symbols;
2) different methods for combination of classifiers have been attempted to arrive at an efficient fusion of the classifiers;
3) for interpretation of structure, online features are used along with several offline analysis;
4) a new performance measure has been presented to evaluate the system performance;
5) in addition, the proposed system supports use of numerals and several characters of an Indian Language (IL) script [Devanagri (Hindi)] to enter MEs containing IL digits and letters.

The rest of the paper is organized as follows. Section II presents a brief review of the previous studies. Section III describes the proposed system and the methodologies for symbol recognition and interpretation of structures. Experimental results and the proposed method of performance evaluation are presented in Section IV. Section V concludes the paper.

Manuscript received March 4, 2003; revised December 24, 2003. This paper was recommended by Associate Editor V. Govindaraju.

The authors are with the Computer Vision and Pattern Recognition Unit of Indian Statistical Institute, Kolkata 108, India (e-mail: utpal@isical.ac.in; bbc@isical.ac.in).

Digital Object Identifier 10.1109/TSMCB.2004.836817

II. Brief Review of Previous Studies

The earliest paper on online handwritten MEs is due to Anderson [3], [4] who assumed error-free symbol recognition and presented a coordinate grammar for analyzing 2-D structures of MEs. A partitioning strategy was used for rules with two nonterminal syntactic units on their right side and each partition might require considerable processing. Later on, Belaid, and Haton [5] proposed a syntactic technique for processing online MEs where symbols are segmented into basic primitives for recognition. Eight expressions written by ten persons four times to create a dataset of 320 expressions were used and a symbol recognition accuracy of 93% was achieved. All structures except six cases were properly recognized.

Koschinski *et al.* [6] and Winkler *et al.* [7] use the Hidden Markov Model (HMM) for symbol recognition and incorporate a soft-decision approach for analysis of ME structure. 82 symbols written 50 times by a subject were considered and 40 versions were used for training the HMMs. Writer-dependent recognition accuracy of 96.9% has been reported. Among other HMM based approaches, Kosmala *et al.* [8] proposed a neural net (NN)-HMMs based system. HMMs of different number of states have been proposed to recognize symbols. A graph grammar approach [9] was used for analysis of ME structure. In another method proposed by Xuejun *et al.* [10], symbol matching is done by an improved Kohn-Munkres algorithm. The approach was tested with 94 ME symbols written five times by 20 different persons and a writer-dependent symbol recognition accuracy of 90.52% was achieved.

Sakamoto *et al.* [11] used a 16-directional coding scheme to capture writing directions of a stroke. A dynamic programming is used for segmentation of a sequence of strokes into character units. In a related study [12], Fukuda *et al.* have used 3×5 mesh directional element features and some additional features for symbol recognition. For structure analysis, both [11], [12] have employed the same technique [13] that chooses one of the nine pre-defined relations for a pair of ME symbols. Four MEs written twice by 20 persons resulted in a database of 160 MEs. A character recognition accuracy of 99.35% is reported. The technique shows an efficiency of 98.46% toward identifying spatial relations between a pair of symbols.

Chan and Yeung [14] proposed a syntactic approach that used Definite Clause Grammar (DFG) to define a set of replacement rules for parsing MEs. The ME symbols are recognized by following a flexible structure matching approach [15]. Experiments are done on 60 MEs taken from four different domains of mathematics. Performance evaluation of the system is presented in a different paper [16] where effectiveness of both the symbol recognition and structural analysis stages is demonstrated by a single measure.

Toyozumi *et al.* [17] presented a system that recognizes each stroke by Freeman chain code. Several strokes are combined into a character based on their positions and combinations. Structural analysis is done by dividing a mathematical formula into blocks, but details of the method and dataset used are not presented. Accuracy for recognition of characters, mathematical structures and matrix structures are 80%, 92%, and 69%, respectively.

Later on, Zanibbi *et al.* [18] described a tree transformation based method to understand the 2-D structures. The approach makes use of search functions that exploit the left-to-right reading order of ME notations and operator dominance to recursively and efficiently extract baselines in an ME. This approach is further used in another system designed by Tapia and Rojas [20] who proposed a support vector machine based recognition of handwritten symbols and an accuracy of more than 99% has been reported for recognition of 43 distinct symbols used for writing MEs. However, recognition of symbols in [18] is achieved through another system [19]. Test results given in [18] put emphasis on recognition of typeset MEs and only five "fairly complicated" MEs written by 27 participants are used in the experiment. It is reported that the participants found the output to be useful.

III. Proposed System

Fig. 1 shows the architecture of our proposed system. Each stroke drawn on an electronic data tablet stroke goes through some preprocessing steps to remove variations (due to noise and uncontrolled pen movement) that would otherwise complicate the recognition process. Several preprocessing steps like four-connected to eight-connected region generation, interpolation of missing points, smoothing, etc. are used. The recognition method works at the stroke level. Individual strokes are then grouped into a meaningful symbol. A stroke is combined with a neighboring one by looking at some spatio–temporal information like time-gap between these two strokes, positional proximity, etc. A symbol written with multiple strokes is recognized by looking at the sequence of its strokes. This sequence is checked by using a rule base maintained against each symbol consisting of multiple strokes. Stroke-order variations is tackled by maintaining, whenever required, multiple definitions of the stroke sequences for a single symbol. Several spatial relations between a pair of symbols are identified online. However, the existence of such relations gets confirmed under an offline processing where the entire expression is reconstructed.

A. Symbol Recognition

For recognition of symbols, our algorithm tries to exploit the neuromotor characteristics of handwriting. Consider the way in which a child learns to write. S/he is advised to down the pen at some position, make straight/curved pen movement in a particular direction, create loops when needed and lift the pen at some other position. Pen down position for the next stroke is also mentioned and s/he follows such instructions until the character is complete. Apart from pen up/down positions and the direction of pen movement, the children are also taught the relative lengths of different parts of a stroke. These aspects are captured and used as features.

Feature Extraction: To elaborate our feature extraction process, let the digitizer output be represented in the format of $(\mathrm{pt}[l])_{l=1}^{N} \in \mathcal{R}^2 \times \{0,1\}$, where $\mathrm{pt}[l]$ is the pen position having x-coordinate $(\mathrm{pt}[l] \cdot x)$ and y-coordinate $(\mathrm{pt}[l] \cdot y)$. Additionally, pen up and pen down information is captured to distinguish a stroke. At first, we extract angle variation information as follows:

$$v[i] = \mathrm{pt}[i] - \mathrm{pt}[i-1], \quad i = 1, 2, \ldots, N-1$$
$$\phi_i = \pi - \mathrm{sign} \times \cos^{-1}\left(\frac{v[i] \cdot x}{\delta l_i}\right) \tag{1}$$

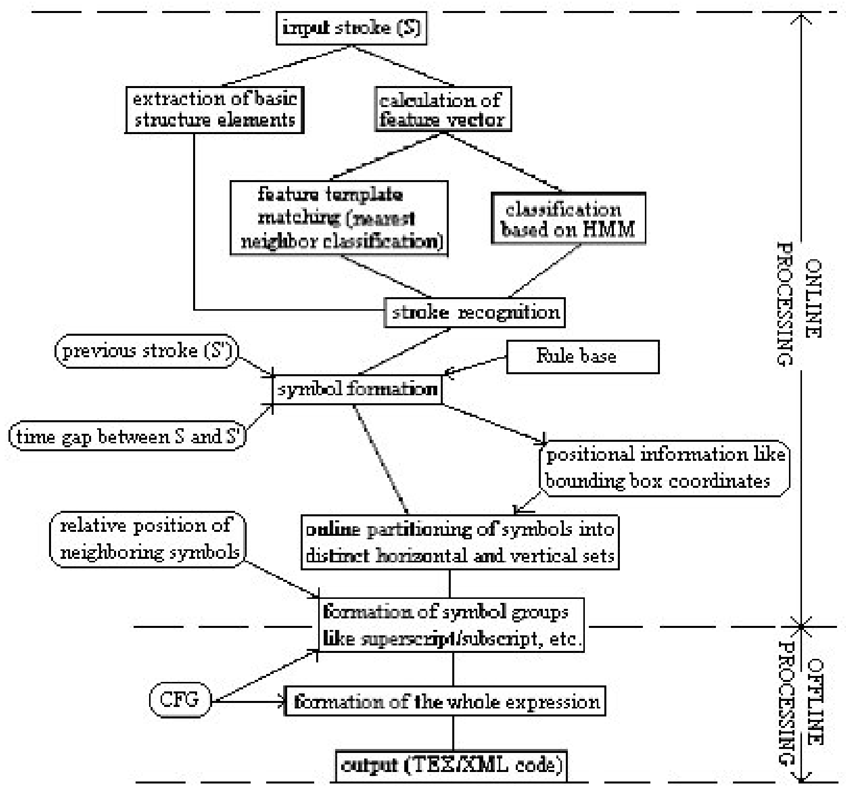

input stroke (S)
extraction of basic structure elements
calculation of feature vector
feature template matching (nearest neighbor classification)
classification based on HMM
stroke recognition
previous stroke (S')
Rule base
symbol formation
time gap between S and S'
positional information like bounding box coordinates
relative position of neighboring symbols
online partitioning of symbols into distinct horizontal and vertical sets
formation of symbol groups like superscript/subscript, etc.
CFG
formation of the whole expression
output (TEX/XML code)
ONLINE PROCESSING
OFFLINE PROCESSING

Fig. 1. Architecture of the proposed system.

where $r[i]\cdot cx = \mathrm{pt}[i]\cdot x - \mathrm{pt}[i-1]\cdot x$, $r[i]\cdot cy = \mathrm{pt}[i]\cdot y - \mathrm{pt}[i-1]\cdot y$ and $\mathrm{sign} = -1$ if $r[i]\cdot cy < 0$, otherwise $\mathrm{sign} = 1$. The δl_i is the Euclidean distance between two consecutive points and measured as $\sqrt{(r[i]\cdot cx)^2 + (r[i]\cdot cy)^2}$.

However, in reality, instead of angle variation information, direction change information is taught to a child. Therefore, (2) converts ϕ_i into a direction code (an integer) following a eight-direction Freeman coding as shown in Fig. 2.

$$d_i = \left(\left((\mathrm{int})\left(\frac{8\phi_i}{\pi}\right) + 1\right) \bmod 16\right) \div 2 \tag{2}$$

where mod is the modulus operator and int returns integer part of a real number. Next, we normalize δl_i which represents local trajectory length as follows:

$$\delta l_i = \frac{\delta l_i}{\sum_{j=1}^{N-1} \delta l_j}. \tag{3}$$

1) Description of the Classifiers: Two different classifiers are used in our system. *Classifier* 1 involves feature template matching approach and employs a nearest neighbor classification scheme, whereas, *classifier* 2 uses HMM [21] for classification. The details of the classifiers are described as follows.

- **Classifier 1**: Feature vector used by this classifier is defined by the tuple $(d_i, \delta l_i)$, $i = 1, 2, \ldots, N-1$. Nearest neighbor classification is implemented by a distance measure as follows assuming that the feature vectors for T (stroke to be recognized) and S (stored prototype) are given by $f_T = (d_k^T, \delta l_k^T)_{k=1}^K$ and $f_S = (d_j^S, \delta l_j^S)_{j=1}^J$, where $\sum_k \delta l_k^T = \sum_j \delta l_j^S = 1$.

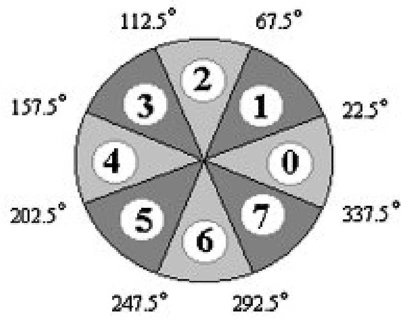


Fig. 2. Eight-directional Freeman-Chain coding.

In reality, K is rarely equal to J, hence, implementation of any direct distance measure is difficult and therefore, the following modification is done

$$\Delta L_k^T = \sum_{i=1}^{k} \delta l_i^T \quad \text{and} \quad \Delta L_j^S = \sum_{i=1}^{j} \delta l_i^S. \tag{4}$$

Next, we sort the union of $(\Delta L_k^T)_{k=1}^K$ and $(\Delta L_j^S)_{j=1}^J$ together in an increasing sequence. Let this sequence of numbers be $(\Delta L_r)_{r=1}^R$ where $\Delta L_R = 1$. Now, it is clear that the direction codes (d_k^T) of T and (d_j^S) of S are constant over $\Delta L_r - \Delta L_{r-1}$.

The feature vectors, f_T and f_S are re-defined as $(d_r^T, \delta l_r)$ and $(d_r^S, \delta l_r)$, respectively, and the distance between them is measured as follows:

$$J(T,S) = \sum_{r=1}^{R} \delta l_r \times \left(1 - |1 - |d_r^T - d_r^S||\right). \tag{5}$$

Note that in 8-directional coding (see Fig. 2) the maximum difference between two successive direction codes can be 4. Moreover, it may be noted that since $\sum \delta l_r = 1$, the metric property of J is retained in the (5). A formal proof can be found in [22]. For any input stroke T, $J(T, S_i)$ is measured against all stored prototypes S_i and T is classified as S_i if $J(T, S_i) < J(T, S_j)\ \forall j \neq i$. However, in our implementation, classes are ranked based on the J values and the class with lowest J gets the highest rank.

- **Classifier 2**: This classifier uses a left-to-right Hidden Markov Model (HMM) for recognition of symbols. HMM parameters are estimated as follows:
  - The states of HMM: For each stroke, we consider its own HMM. Since different writers write a particular stroke in different manners, a number of observation sequences are used to train the model. Let there be η number of such sequences available and each sequence consists of M observation symbols, $O = O_1, O_2, \ldots, O_M$, where (O_i) is presented in a 2-D vector, $(\phi_i, \delta l_i)_{i=1}^{M}$ where ϕ_i and δl_i are given by (1) and (3), respectively. Next, each of the ηM observation vectors is mapped into one of the N clusters. For this purpose, ϕ_i is given a direction code d_i based on the Freeman-chain coding scheme given in (2) and each segment length δl_i is labeled as short, medium or long, based on two predefined thresholds. As d_i can take one of the eight values (0–7) and ϕ_i can have one of three descriptions (short, medium, or long), any observation O_i will be mapped into one of the 24 clusters. Each cluster forms a state.
  - Training and recognition of strokes: To train the model, we follow the segmental K-means Algorithm [23]. The initial $(\hat{\pi}_i)$ and transition probabilities $(\hat{a}_{ij})$ are calculated as follows:

    For $1 \leq i \leq N$
$$\hat{\pi}_i = \frac{\#\text{occurrences of } \{O_1 \in i\}}{\#\text{occurrences of } O_1 \text{ i.e., } \eta}. \tag{6}$$

    For $1 \leq i \leq N$ and $1 \leq j \leq N$
$$\hat{a}_{ij} = \forall t \frac{\#\text{occurrences of } \{O_t \in i \text{ and } O_{t+1} \in j\}}{\#\text{occurrences of } \{O_t \in i\}}. \tag{7}$$

    The observation matrix $(B = \{b_i(k)\})$, the probability of observing the symbol O_k given that the model is in ith state is estimated by calculating the symbol probability distributions (assumed Gaussian) for each training vector for each state as follows:

    For $1 \leq i \leq N$
$$\hat{b}_i(O_t) = \frac{1}{2\pi|\hat{C}_i|^{\frac{1}{2}}} \exp\left[-\frac{1}{2}(O_t - \hat{\mu}_i)\hat{C}_i^{-1}(O_t - \hat{\mu}_i)^T\right] \tag{8}$$

    where $\hat{\mu}_i$ and $\hat{C}_i$ are the mean vector and the covariance matrix for each state, i.

Viterbi Algorithm [24] is used to find optimal state sequence for each training sequence. Once the training is over, well-estimated HMMs are obtained for strokes (for some strokes more than one corresponding HMMs are maintained). To recognize a stroke, the observation sequence which configures the stroke is considered and the probabilities of occurrence of the observation sequence against each HMM, λ_i are calculated. In actual implementation, the classes (w_i) are ranked based on the $P(O \mid \lambda_i)$ values and the class with the maximum $P(O \mid \lambda_i)$ gets the highest rank.

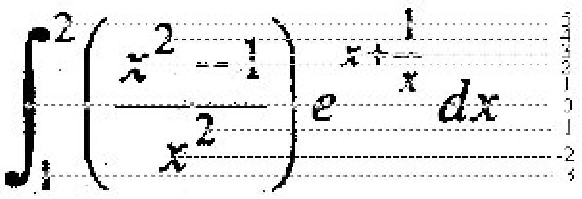


Fig. 3. Baseline and other horizontal lines in an expression.

Fusion of the Classifiers: Before looking for a suitable combination method to integrate the classifiers described above, similarity between classifiers is, at first, studied by measuring the agreement between their decisions. Similarity is measured using the index discussed in [25]. It is to be noted that the minimum of the similarity index is equal to 0 (when the classifiers always disagree to each other) and the maximum is equal to 1 (when the classifiers always provide the same response).

Next, the classifiers are combined following three combination methods:

1) highest rank method;
2) borda count [27];
3) logistic regression [28].

All these methods attempt to improve the rank of the correct class. The work by Ho *et al.* [26] discusses about the use of these three methods for classifier combination. In our experiment, relative merits and demerits of each combination method are studied and results are presented in Section IV.

B. Analysis of ME Structure

The approach for analysis of an expression's structure consists of three stages, namely, online interpretations, offline processing, and compilation of TEX string.

- Online Interpretations: As soon as a symbol (S_i) is drawn, its bounding box (BB_i), center $(C_i(y))$ (i.e., bounding box's center) are noted. Each symbol is tagged with a *Level*, (L) having the following properties 1) L-value of any symbol belonging to the expression baseline is 0 2) L-values of symbols increases upward and decreases downward with respect to the baseline. Fig. 3 shows L-values of individual symbols of an expression.

  For easy detection of symbol levels writing of any expression starts with its left-most baseline symbol. Symbols having nearly the same (determined by computing the variance) center $C(y)$ get the same L-value. Basically, $C(y)$ values of symbols determine a set of horizontal lines on which symbols are arranged in an ME. For a pair of symbols, their bounding box coordinates and the L-values help to determine any spatial relationship exists within the pair. Several structures like first level *superscripts, subscripts, limit, square root*, etc. are identified online and corresponding TEX strings are generated for them.

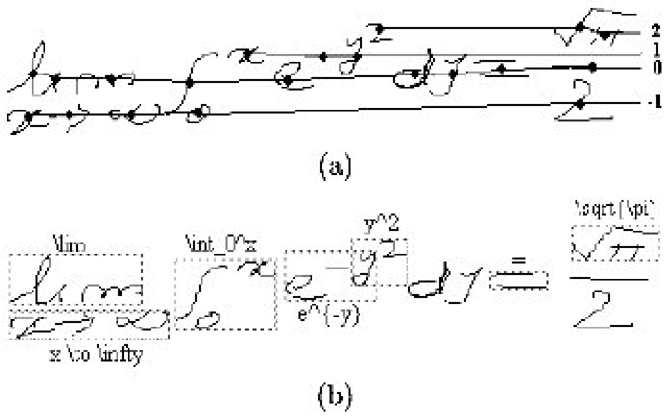


Fig. 4. Online processing: Recognition of several structures.

Meaning of certain ambiguous symbols (e.g., *dot, horizontal line*, etc.) is also understood online. For example, a dot ("$\cdot$") symbol appears in different context like as a decimal point, an accent marker $(\dot{a})$, etc. Therefore, meaning of such an ambiguous symbol like *dot* is interpreted by looking at its neighboring symbols (i.e., symbols left, right, and below to the symbol. As the function words (e.g., sin, log, exp, etc.) maintain linear one–dimensional (1-D) structures, they are recognized online. A finite automata is maintained to spot occurrence of any such function word in an input ME.

Fig. 4 demonstrates the steps described above. Fig. 4(a) shows how symbols are given their L-values based on their center $C(y)$ values. The structures identified online are shown in Fig. 4(b). In the present system, writing of a *root* sign imposes a restriction that *root* $(\sqrt{})$ symbol is drawn first, then symbols under *root* are written. The expression in Fig. 4(a) contains four horizontal lines whose meanings are also interpreted online. Occurrence of the function word lim is also identified at this stage.

- Offline Processing: Structures unidentified during online are processed offline. Moreover, the relations identified at the first stage are also checked for final acceptance. Initially, the entire expression image is recursively segmented into vertical and horizontal stripes till no more segmentation is possible. TEX strings are generated for each segment (or stripe). Next, a bottom-up approach is followed to merge two segments to generate a new TEX string. This merging process continues until the final expression is constructed. A context-free grammar, G, presented later, guides the merging process and generation of TEX string for each segment.

  Figs. 5(a)–(c) demonstrate major processing steps. Fig. 5(a) shows the vertical segmentation of the expression into nine vertical stripes (*vStripe*). Next, each vStripe are further segmented into horizontal stripes (*hStripe*). The vStripes and hStripes are tagged with numbers to keep track of their order of generation. For example, a tag (9, 1) with the hStripe, $H_{9,1}$ indicates that it is one of the stripes generated during horizontal segmentation of the vStripe V_9.

  This vertical and horizontal segmentation go hand-in-hand until each stripe (called atomic box) contains a single symbol, or no further segmentation is possible [Fig. 5(b)]. For an atomic box containing more than one symbol [(e.g., $H_{9,1}$ of Fig. 5(b)] is processed further. In such cases, the *largest* symbol (determined by the bounding box area) is separated and the rest of the symbols are subject to further segmentation. However, in case of $H_{9,1}$, only one symbol (i.e., "π") is left after the largest symbol (i.e., "$\sqrt{}$") is separated. At this stage, a suitable production rule of G (described next) is searched to get the relation between "$\sqrt{}$" and "π" and a TEX string, "\pi" is returned.

  Next, hStripes under a vStripes are processed using G to find spatial relation, if any, among them. For example, hStripes of V_9 [Fig. 5(b)] are merged following the production rule corresponding to *fraction* stated in (10). But this does not happen for all vStripes and in the succeeding stage, hStripes in adjacent vStripes are merged in left to right manner. A hStripe of V_i is merged with a hStripe of V_{i+1} if they have same levels. Level of a hStripe is determined by the center of its enclosing bounding box. For example, $H_{5,1}$ of V_5 [Fig. 5(c)–(d)] gets connected with $H_{6,2}$ of V_6. This merging propagates from left to right till no further merging is possible. The vStripes that are involved in such a merging process get fused to form a new vStripe as shown in Fig. 5(e). The function words like sin, cos, log, etc. are formed at this stage.

  Finally, pairwise lumping of vStripes is done in a left to right manner. In this stage, a single TEX string between V_i and V_{i+1} is searched using G. For instance, V_5 and V_6 [Fig. 5(b)] together form the TEX string, $y\hat{}\{2\}$ using production rule corresponding to *superscript*. However, when a vStripe is represented by more than one TEX string, several production rules are used to merge the pair, e.g., V_2 of Fig. 5(c) is represented by two TEX strings, namely, n and x and hence, when merging of (V_1, V_2) pair is considered, relation between D and n, and between D and x are searched and a single TEX string is returned by using production rules corresponding to *superscript* and *subscript* formation. Note that processing of matrices does not require any extra effort. The same reconstruction algorithm can tackle matrix structures using production rules given in (11).
- Compilation of TEX strings and use of Contextual Information: During online or offline processing whenever a TEX string is generated, it is checked for its syntactic validity and any failure invokes immediate processing with available alternatives. For example, in Fig. 4(a), "lim" has initially been recognized as "um" which gives two possible interpretations: 1) multiplication of two identifiers, "u" and "m" or 2) the combination "um" is a function word. First interpretation is rejected looking at the *limit* expression $(x \to \infty)$ below it. In the second case, the generated TEX string "\um" does not pass through the validation check. Hence, the system uses other alternatives provided by the symbol recognizer, e.g., for "*li*" of "lim," "u" is the top choice and "*li*" as the second best choice, which forms a valid TEX string "lim." If no alternatives generates a valid string, the system generates a string with the best choices for its constituent symbols and leaves it for manual correction at a later stage.

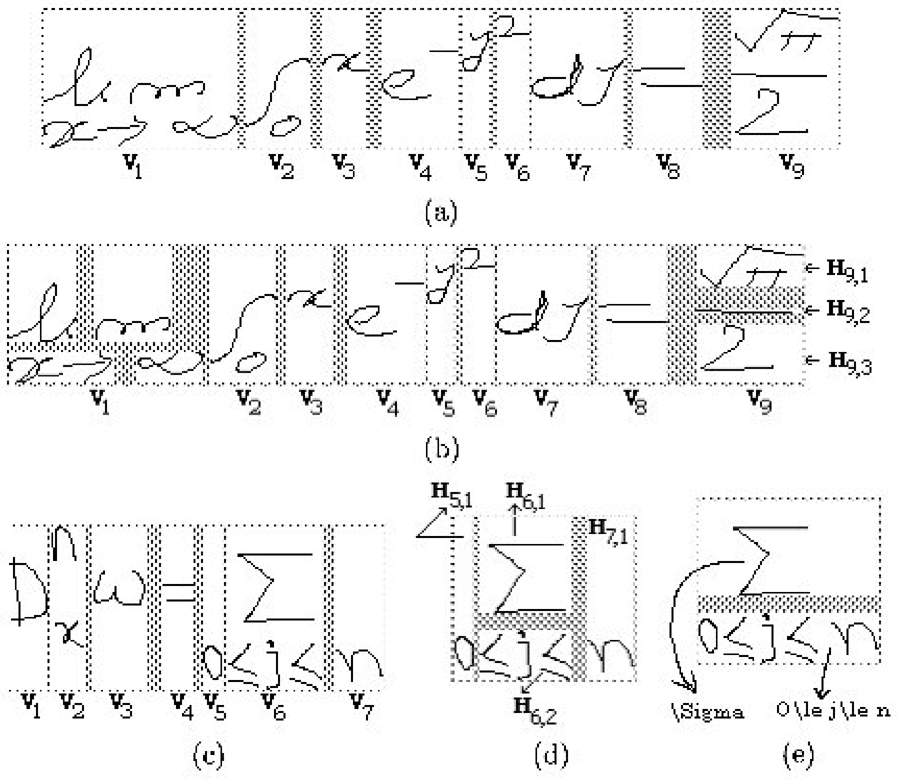


Fig. 5. Offline processing: Horizontal and vertical segmentation.

Contextual information is used while generating a new TEX string after merging of one or more stripes. For instance, vStripes, V_2 through V_7 [see Fig. 5(b)] are merged and a TEX string, "\int_0^x e^{-y^2} dy" is formed. Here, "dy" seems to a multiplication of "d" and "y." However, presence of "\int" looks for its differential and converts the string into "\int_0^x e^{-y^2}\, dy," where "dy" conveys its actual meaning

C. Grammar

The grammar (G) that has been referred in the preceding sections is a context-free one in nature. Symbols occurring in expressions are considered as terminals. In total, there are 208 terminal symbols grouped into eight categories, each represented by a *nonterminal*. The number of terminals under each category and name of the *nonterminal* (written inside braces) representing the respective category are the following:

1) Arabic Numerals (AN): 10.
2) Roman Letters (RL): 52.
3) Greek Symbols (GS): 30.
4) Mathematical Symbols (MS): 80.
5) Punctuation Marks (PM): 6.
6) Function Words (FW): 20 (e.g., "sin," "cos," "arg," "lim," "ln," etc.).
7) Hindi Numerals (HN): 10.
8) Hindi Letters (HL): 10.

In addition, other terminals like "\," "^" and underscore ("_"), etc. and 27 TEX keywords (e.g., "frac," "sqrt," "ldots," "hat," "bar," etc.) are used for convenience of generating TEX strings. In total, the grammar, G contains 239 (including *NULL*) terminal symbols. On the other hand, 16 nonterminals and a set of 276 production rules are used. The major productions are explained below.

Initially, the input ME is segmented into n vertical stripes. This is implemented by the initial productions originated at the start symbol, S as follows:

$$S \rightarrow ES \mid E \tag{9}$$

where E is a nonterminal used to produce a syntactically valid TEX string for a *vStripe*. The productions originated at E are given in (10), where AN, RL, GS, MS, etc. are nonterminals that finally generate terminal symbols of respective categories. Two more nonterminals, *ELLIP* and *ACCENT* are used to generate four types of ellipses (e.g., $\ldots, \cdots$, etc.) and different accents (e.g., $\bar{~}, \acute{~}$, etc.), respectively:

$$\begin{aligned} E \rightarrow\ & S\text{\textasciicircum}\{S\} \mid S\_\{S\} \mid \backslash\text{frac}\{S\}\{S\} \mid \backslash\text{sqrt}\{S\} \mid \\ & \backslash\text{stackrel}\{S\}\{S\} \mid \backslash\text{overline}\{S\} \mid \backslash\text{underline}\{S\} \mid \\ & \backslash\text{overbrace}\{S\} \mid \backslash\text{underbrace}\{S\} \mid \\ & \backslash\text{mathcal}\{RU\} \mid \backslash\text{ELLIP} \mid \backslash\text{ACCENT} \mid \\ & \backslash\text{begin}\{\text{array}\}\ \text{MAT}\ \backslash\text{end}\{\text{array}\} \mid \\ & \text{AN} \mid \text{RL} \mid \text{GS} \mid \text{MS} \mid \text{PM} \mid \text{FW} \mid \text{HN} \mid \text{HL} \mid \epsilon. \end{aligned} \tag{10}$$

The nonterminal MAT takes care of matrix structures. Further expansion of MAT is given in (11). Note that it does not retain the alignment information (e.g., left, right, or center) for matrix columns:

$$\begin{aligned} \text{MAT} &\rightarrow \text{ROW}\ \backslash\backslash\ \text{MAT} \mid \text{ROW} \\ \text{ROW} &\rightarrow S\ \&\ \text{ROW} \mid S. \end{aligned} \tag{11}$$

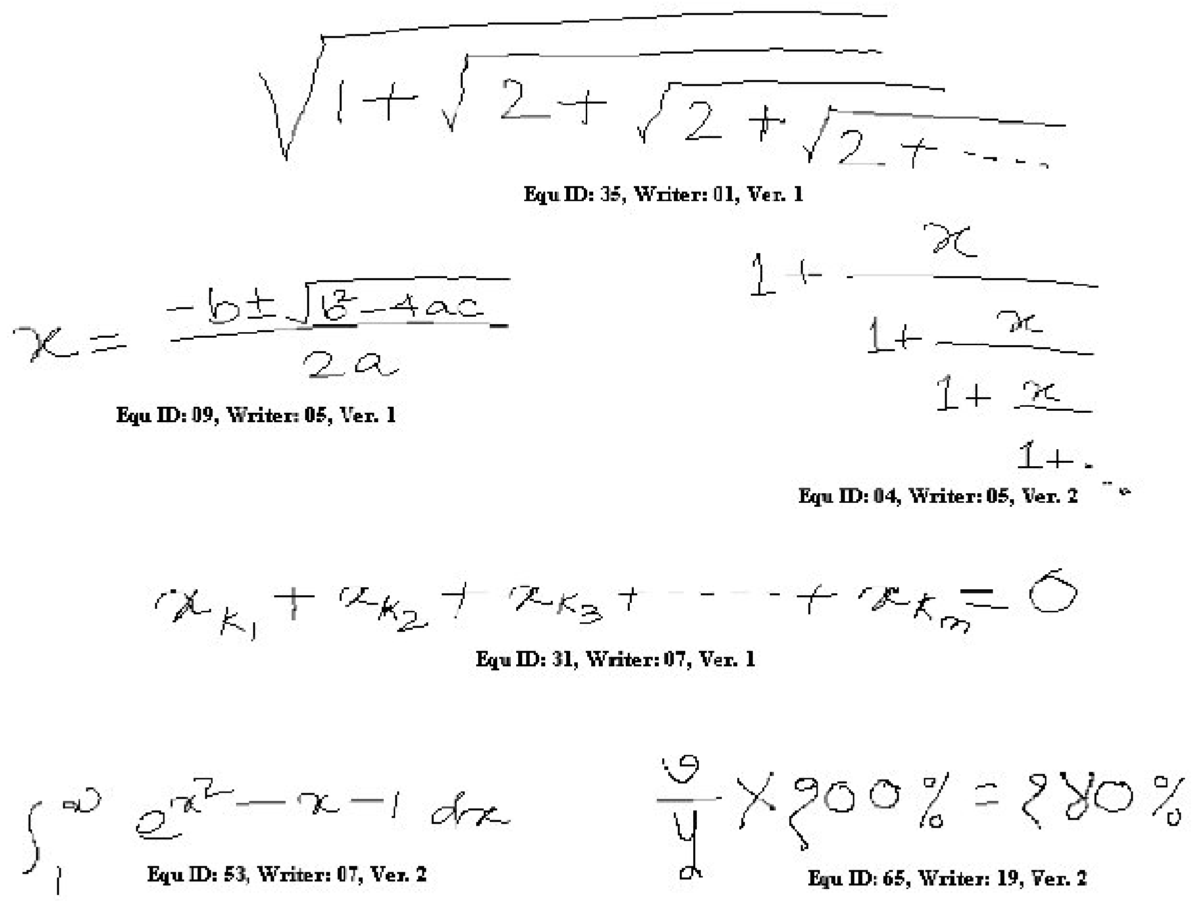

Equ ID: 35, Writer: 01, Ver. 1
Equ ID: 09, Writer: 05, Ver. 1
Equ ID: 04, Writer: 05, Ver. 2
Equ ID: 31, Writer: 07, Ver. 1
Equ ID: 53, Writer: 07, Ver. 2
Equ ID: 65, Writer: 19, Ver. 2

Fig. 6. Some handwritten MEs used in the experiment.

IV. Experimental Results and Discussions

At first, a database of handwritten MEs is constructed to test the proposed system. A Genius-make electronic tablet (Wacom Intuos 12×12 Serial Tablet) attached with a 733 MHz IBM PC is used to capture data. 175 expressions are taken from different branches of science. Some MEs from the database are shown in Fig. 6. Handwritten samples for these MEs are divided into three parts. In Part-I, 100 are selected from various topics covered in science (mainly mathematics) books of school and college standard and written twice by 20 students studying at the corresponding standards. This resulted in $4\ 000\ (100\times20\times2$ $4000)$ samples for the 100 MEs.

In Part-II, 50 MEs are taken from the dataset used by Raman [29]. This dataset considers different types of expressions and freely available in the Internet. Ten writers engaged in research and teaching at the university level have been asked to write two versions for each ME to generate $1000\ (50 \times 10 \times 2 = 1\ 000)$ samples. Part-III of the database contains 25 MEs selected from Hindi medium high school level science books and contain digits and letters of Devanagri (Hindi) scripts. Ten native Hindi writers are asked to write each of these 25 MEs twice resulting in a set of $500\ (25 \times 10 \times 2 = 500)$ samples.

Each expression is given a unique identification key `exp-id`. MathML[1] presentation tags are used to encode the expression contents. A few user-defined tags also are used to record several aspects, for example, each expression is tagged with its geometric complexity (**GC**) measured by the number of horizontal lines (on which expression symbols are arranged) found in that expression. Note that, **GC** is 9 for the expression in Fig. 3. Each expression symbol is tagged with `level` value as shown in Figs. 3 and 4(a). Online data for each sample is recorded strokewise. Number of strokes, number of points for each stroke and time gap between two successive strokes are registered.

[1]See W3C Math Home at http://www.w3.org/Math/.

A. Recognition of Expression Symbols

The total number of distinct symbols present in the database expressions is 3176. Number of handwritten samples for these 3176 symbols is 98 060. Details are given in Table I. It is to be noted that ME symbols present in our database generate 198 distinct classes and are partitioned into seven different categories as shown in Table II. The category "*Math. Symbols*" contains the largest number of symbols including binary and relation operators, arrow symbols, bracket symbols and several miscellaneous symbols like *prime* ("$\prime$"), *for all* ("$\forall$"), *there exists* ('$\exists$'), etc. The classification results obtained from two classifiers are outlined in Table II that considers only the top ranked class returned by a classifier.

Analysis of the classification results shows that the classifiers are of comparable power but behave differently for different types of symbols. Classifier-I shows slight better performance

TABLE I
DISTRIBUTION OF DATABASE SAMPLES

Source	#Distinct Symbols	#Writers (Versions)	#Total Samples (Training/Test)
Part-I	1,727	20 (2)	69,080 (51,807/17,273)
Part-II	1,072	10 (2)	21,440 (17,142/4,298)
Part-III	377	10 (2)	7,540 (6,037/1,503)
Total	3,170		98,060 (74,986/23,074)

TABLE II
TRAINING AND TESTING OF THE CLASSIFIERS

Symbol Type (#Classes)	#Sample Training / Testing	Correct Recognition on Test Set by Classifier I	Classifier II
Arabic Numeral (10)	8,998 / 2,769	2,453 (88.59%)	2,551 (92.13%)
Roman Letters (52)	19,496 / 6,192	5,676 (91.67%)	5,524 (89.21%)
Greek Symbols (30)	5,993 / 1,845	1,702 (92.25%)	1,654 (89.66%)
Math. Symbols (80)	28,494 / 8,568	7,747 (90.42%)	7,898 (92.19%)
Punctuation Marks (6)	1,249 / 692	603 (87.18%)	638 (92.24%)
Hindi Numeral (10)	7,460 / 2,107	1,868 (88.17%)	1,970 (93.49%)
Hindi Letters (10)	3,287 / 901	840 (93.26%)	822 (91.12%)
Total (198)	74,986 / 23,074	20,889 (90.53%)	21,057 (91.26%)

TABLE III
COMBINATION OF CLASSIFIERS

Classifiers and Combinations	% Correct in Top N Choices 1	2	3	5	10
Classifier-I	90.53	93.60	96.25	96.96	96.98
Classifier-II	91.26	94.12	97.03	97.69	97.70
Highest Rank	93.18	96.73	98.02	98.92	98.96
Borda Count	91.92	94.87	97.41	98.03	98.15
Logistic Regre.	93.77	96.82	98.09	98.97	99.12

for complex shaped symbols (e.g., Roman letters, Greek symbols, Hindi Letters, etc.) whereas Classifier-II responds better for symbols with less structural complexity (e.g., Numerals, Math operators, punctuation marks, etc.). Next, the classifiers are combined and details of combination results are presented in Table III that shows that the highest rank method and the logistic regression give comparable performance, the latter being slightly better. On the other hand, as the Borda count method does not consider the differences in the individual classifier capabilities, its performance is less attractive.

B. Interpretation of Structures

As the detection of *level* (L) of a symbol plays an important role in its final placement, the accuracy for detection of L-value for each symbol is initially measured to test the module analyzing ME structures. Next, placement of a symbol is checked by looking at the interpretation result for its immediate parent structure that contains the symbol. For example, placement of "2" in e^{x^2} is assessed by checking the recognition result of the superscript structure "x^2." Results obtained for detection of the level and placement for each symbol are shown in Table IV.

TABLE IV
INTERPRETATION OF SYMBOL POSITIONS

Number of Symbols	Accuracy: Detection of Level	Placement
98,060	95,531 (97.42%)	92,049 (93.87%)

TABLE V
STRUCTURE LEVEL ACCURACY

Structure Types	Total	Correct Recognition
Scripts	13,240	13,101
Limit	2,588	2,528
Fraction	494	484
Root	248	230
Overline	156	148
Underline	34	31
Overbrace	118	111
Underbrace	52	47
Accent	1,562	1,459
Matrix	38	36
Stacking of Symbols	398	361
Ellipses	2,152	2,066
Function words	1,494	1,449
Parenthesis	1,928	1,893
Other 1-D Structures	10,168	9,986
Summary	34,676	33,939 (97.87%)

MEs in the database show fourteen elementary structures among them twelve are 2-D in nature. They are

1) superscript;
2) subscript;
3) fraction;
4) root;
5) overline;
6) underline;
7) overbrace;
8) underbrace;
9) ellipses;
10) accent;
11) matrix;
12) stacking of symbols.

On the other hand, 1-D structures are grouped into two classes, namely, *Function word* (e.g., "sin," "log," "etc.) and other 1-D structures (e.g., "−," "−," parentheses, etc). The recognition accuracy for individual structures is presented in Table V. Errors encountered in structure analysis stage are mainly attributed to the following:

- Handwriting Style: The prime reason behind errors in symbol placement is casual handwriting style that create confusion while determining spatial relationship between a pair of symbols. Some meaningful restrictions on writing style will definitely help to resolve several ambiguities that arise at the structural analysis stage.
- Ambiguous role of a symbol: Several symbols like *dot*, *horizontal line segment*, etc. represent different meaning in different context and lead to ambiguous parsing. For example, parsing of the expression in Fig. 7 gives different results due to incorrect interpretation of the *underline* of

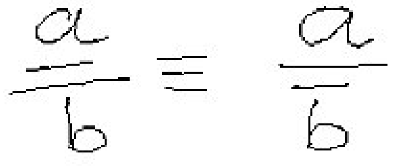


Fig. 7. Expression with symbols having ambiguous role.

"a," *overline* of "b," and the *fraction* lines. Use of a *probabilistic* version of the proposed grammar may help us a lot to resolve many of such ambiguities.

- Effect of geometric complexity: Test results show that the number of errors in symbol placement increases with higher complexity of expressions. Figures in Table VII also attest this observation. For example, the expression in Fig. 8 shows a high geometric complexity and its recognition is incorrect due to wrong placement of symbols which are far apart from the dominant baseline.
- Error in Input: Another reason for causing errors during structural analysis is input of an incorrect ME. The user sometimes make mistake while writing the MEs and hence, an ability to detect such errors would be an added advantage.

Finally, recognition accuracy at the whole ME level is assessed (Table VI) and found that the accuracy at the expression level is quite low because placement error for a single symbol makes recognition of an whole expression incorrect. On the other hand, Table VI indicates that errors in placement of only a few symbols are actually responsible for wrong parsing of the majority of MEs. Therefore, such a measure does not truly evaluate the system and a better evaluation strategy as presented next can be designed.

C. Performance Evaluation

The quantitative evaluation of expression recognition results is a difficult task since recognition scheme involves two major stages: symbol recognition and structural analysis. The stages are tightly coupled and therefore, if evaluation in one stage is done independent of the other, then it may not reflect true performance of the system. This calls for an integrated evaluation mechanism for judging the performance of an expression recognition system.

Chan and Yeung [16] proposed an integrated performance measure consisting of two independent measures: one for recognition of symbols and another for recognition of operators. These two measures are combined with equal weights. Later on, Okamoto *et al.* [30] presented an automatic approach for evaluating their structure analysis method. They attempted to evaluate the performance by checking whether typical structures like scripts, limits, fractions, etc. are recognized correctly. However, both the methods count only the number of properly recognized structures and therefore, an error in recognizing a simple structure gets the same weight as that of an error in a complex nested structure. More recently, Zanibbi *et al.* [18] presented another automatic way of evaluating the performance where an expression is visualized as a set of symbols appearing on different baselines. The performance is assessed by separately counting the number of 1) correctly recognized baselines and 2) properly placed (w.r.t. the corresponding baseline) symbols. Such an assessment presents more in-depth analysis of the recognition results but does not provide any single figure of merit for overall performance evaluation.

In our study, we formulate a new performance-index (γ) that uses geometric (or structural) complexity of an expression to measure overall performance. The structural complexity of an expression is defined by (**GC**) i.e., the number of horizontal lines on which constituent symbols are arranged. One can visualize that the error in interpreting position of any base level symbol adversely affects the layout of other immediate symbols nested on it. Moreover, as use of nonbase level symbols (e.g., script or limit expressions) increase the structural complexity of an expression, any systematic evaluation strategy is expected to consider how a system can recognize simple expressions and then to check the system's response as the complexity increases. Therefore, to evaluate the efficiency of an approach that deals with recognition of expressions, one has to take the geometric complexity of expressions into account.

In our approach, recognition result for a handwritten expression is compared with the groundtruth corresponding to that expression. If they do not match, then the result is not correct. Errors originate from two sources, namely, 1) symbol recognition errors and 2) errors in structure interpretation. Symbol recognition errors is easily computed as a ratio of number of properly recognized symbols and the total no. of symbols. However, computation of structure recognition errors is not a trivial one. This is so because parsing of an expression may not be fully correct but some of its symbol arrangements may be interpreted properly and the system should be given partial credit for it. In our method, erroneous arrangement of a symbol (s) is penalized by a factor $(1)/(|i|+1)$, where i is the `level` of the symbol, s. It is to be noted that in case of computing structure recognition errors only spatial arrangements are important. Therefore, no symbol recognition errors are counted as such errors are taken into account while computing symbol recognition accuracy.

For any test expression, let S_t be the total number of symbols, S_e be number of symbols recognized wrongly, R_i be the number of symbols in the ith level and E_i be the number of ith level symbols for which incorrect arrangement analysis is encountered. Now, the performance-index (γ) is defined as

$$\gamma = 1 - \frac{S_e + \sum_i E_i \times \frac{1}{|i|+1}}{S_t + \sum_i R_i \times \frac{1}{|i|+1}}. \tag{12}$$

Assuming a test set (T) contains N expressions, γ_i is computed for all $i = 1, 2, \ldots N$ and to rate the overall system performance, an average γ_{avg} is computed as

$$\gamma_{\text{avg}} = \frac{1}{N} \sum_k \gamma_k. \tag{13}$$

The performance of our proposed system has been judged following (12) and an average performance is computed according to (13). Table VII reports evaluation results on our dataset.

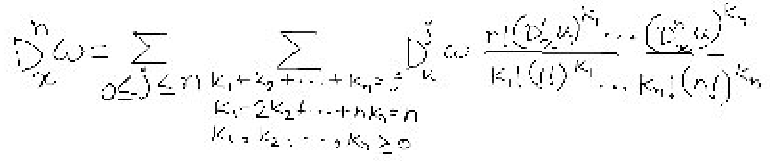


Fig. 8. Faa-de-bruno's Formula: An expression with high geometric complexity.

TABLE VI
EXPRESSION LEVEL ACCURACY

# Expression	# Correctly Parsed Expressions	# Expressions with N Symbol Placement Errors				
		1	2	3	4	≥ 5
5,500	4,121 (74.92%)	522 (37.85%)	409 (29.60%)	243 (17.62%)	122 (8.85%)	83 (6.02%)

TABLE VII
PERFORMANCE EVALUATION

Geometric Complexity	#Exp.	#Samples	Correct Recog.	γ_{avg}
1	25	780	691	0.981
2	36	1,200	997	0.974
3	44	1,500	1,158	0.959
4	31	1,040	716	0.927
5	8	220	154	0.919
6	14	400	251	0.902
7	7	160	91	0.926
8	5	100	39	0.893
9	2	40	10	0.897
10	2	40	8	0.862
11	1	20	6	0.894
Summary	175	5,500	4,121	0.951

V. CONCLUSION

In this paper, a system for online recognition of mathematical expressions is presented. The recognizer uses a multiple-classifier approach. The classifiers have been designed to capture wide variations in shape and size of the large number of symbols that occur in writing mathematics. The proposed technique for structural analysis of MEs uses both online and offline data analysis. A context free grammar is used to parse the input expression. Contextual information has been used at different levels to increase the system efficiency. A new performance measure has been presented in this paper.

Because of this study, online entry of mathematics into electronic documents will become more user-friendly. Also, such a study is useful in developing electronic chalkboard [20], pen-based calculator programs [31], etc. The proposed system supports entry of Devanagri (Hindi) online text and it eventually helps to prepare scientific and technical documents in Indian language [32]. Our future work plan includes designing of a more robust error detection and correction scheme to improve structural analysis of input expressions. Moreover, it would be advantageous for visually impaired people if the recognized expressions were converted into corresponding speech form [33].

ACKNOWLEDGMENT

The authors wish to thank S. K. Parui and J. Mitra for some useful discussions on this work.

REFERENCES

[1] *Handbook of Character Recognition and Document Image Analysis*, H. Bunke and P. S. P. Wang, Eds., World Scientific, Singapore, 1997, pp. 557–582. D. Blostein, A. Grbavec, "Recognition of mathematical".

[2] K.-F. Chan and D.-Y. Yeung, "Mathematical expression recognition: A survey," *Int. J. Document Anal. Recognit.*, vol. 3, no. 1, pp. 3–15, 2000.

[3] R. H. Anderson, "Syntax-directed recognition of hand-printed two-dimensional mathematics," Ph.D. dissertation, Dept. Eng. Appl. Phys., Harvard Univ., Cambridge, MA, 1968.

[4] ——, "Two-dimensional mathematical notations," in *Syntactic Pattern Recognition Applications*, K. S. Fu, Ed. New York: Springer-Verlag, 1977, pp. 147–177.

[5] A. Belaid and J. Haton, "A syntactic approach for handwritten mathematical formula recognition," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-6, pp. 105–111, Jan. 1984.

[6] M. Koschinski, H.-J. Winkler, and M. Lang, "Segmentation and recognition of symbols within handwritten mathematical expressions," in *Proc. IEEE Int. Conf. Acoustics, Speech, Signal Processing (ICASSP)*, vol. 4, Detroit, MI, 1995, pp. 2439–2442.

[7] H.-J. Winkler, H. Fahrner, and M. Lang, "A soft-decision approach for structural analysis of handwritten mathematical expressions," in *Proc. ICASSP*, vol. 4, Detroit, MI, 1995, pp. 2459–2462.

[8] A. Kosmala, G. Rigoll, S. Lavirotte, and L. Pottier, "On-line handwritten formula recognition using hidden Markov models and context dependent graph grammars," in *Proc. of Int. Conf. Document Analysis Recognition (ICDAR)*, Bangalore, Karnataka, India, 1999, pp. 107–110.

[9] S. Lavirotte and L. Pottier, "Optical formula recognition," in *Proc. Int. Conf. Document Analysis Recognition (ICDAR)*, Ulm, Germany, 1997, pp. 357–361.

[10] Z. Xuejun, L. Xinyu, Z. Shengling, P. Baochang, and Y. Tang, "On-line recognition handwritten mathematical symbols," in *Proc. Int. Conf. Document Analysis Recognition (ICDAR)*, Ulm, Germany, 1997, pp. 645–648.

[11] Y. Sakamoto, M. Xie, R. Fukuda, and M. Suzuki, "On-line recognition of handwriting mathematical expression via network," in *Proc. 3rd Asian Technol. Conf. Mathematics (ATCM)*, Tsukuba, Japan, 1998, http://www.atcminc.com/mPublications/EP/EPATCM98/.

[12] R. Fukuda, F. Tamari, X. Ming, and M. Suzuki, "A technique of mathematical expression structure analysis for the handwriting input system," in *Proc. Int. Conf. Document Analysis Recognition (ICDAR)*, Bangalore, Karnataka, India, 1999, pp. 131–134.

[13] K. Inoue, R. Miyazaki, and M. Suzuki, "Optical recognition of printed mathematical documents," in *Proc. 3rd Asian Technol. Conf. Mathematics (ATCM)*, Tsukuba, Japan, 1998, [Online] Abailable http://www.atcminc.com/mPublications/EP/EPATCM98/.

[14] K.-F. Chan and D.-Y. Yeung, "Recognizing on-line handwritten alphanumeric characters through flexible structural matching," *Pattern Recognit.*, vol. 32, pp. 1099–1114, 1999.

[15] ——, "An efficient syntactic approach to structural analysis of on-line handwritten mathematical expressions," *Pattern Recognit.*, vol. 33, pp. 375–384, 2000.

[16] ——, "Error detection, error correction, and performance evaluation in on-line mathematical expression recognition," *Pattern Recognit.*, vol. 34, pp. 1671–1684, 2001.

[17] K. Toyozumi, T. Suzuki, K. Mori, and Y. Suenaga, "A system for real-time recognition of handwritten mathematical formulas," in *Proc. Int. Conf. Document Analysis Recognition (ICDAR)*, Seattle, WA, 2001, pp. 1059–1063.

[18] R. Zanibbi, D. Blostein, and J. R. Cordy, "Recognizing mathematical expressions using tree transformation," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 24, pp. 1455–1467, Nov. 2002.

[19] R. Zanibbi, K. Novins, J. Arvo, and K. Zanibbi, "Aiding manipulation of handwritten mathematical expressions through style-preserving morphs," in *Proc. Graphics Interface*, 2001, pp. 127–134.

[20] E. Tapia and R. Rojas, "Recognition of on-line handwritten mathematical formulas in the E-chalk system," in *Proc. Int. Conf. Document Analysis Recognition (ICDAR)*, Edinburgh, U.K., 2003, pp. 980–984.

[21] L. R. Rabiner and B. H. Juang, "An introduction to hidden Markov models," *IEEE Trans. Acoust. Speech, Signal Processing*, vol. ASSP-36, pp. 4–16, June 1986.

[22] U. Garain and B. B. Chaudhuri, "Compound character recognition by run number based metric distance," in *Proc. IS&T/SPIE's 10th Int. Symp. Electronic Imaging: Science Technology, SPIE*, vol. 3305, San Jose, CA, 1998, pp. 90–97.

[23] B. H. Juang and L. R. Rabiner, "The segmental k-means algorithm for estimating the parameters of hidden Markov models," *IEEE Trans. Acoust. Speech, Signal Processing*, vol. 38, pp. 1639–1641, Sept. 1990.

[24] G. D. Forney, Jr., "The Viterbi algorithm," *Proc. IEEE*, vol. 61, pp. 263–278, Mar. 1973.

[25] L. Bovino, G. Dimauro, S. Impedovo, M. G. Lucchese, R. Modugno, G. Pirlo, A. Salzo, and L. Sarcinella, "On the combination of abstract-level classifiers," *Int. J. Document Anal. Recognit.*, vol. 6, no. 1, pp. 42–54, 2003.

[26] T. K. Ho, J. J. Hull, and S. N. Srihari, "Decision combination in multiple classifier systems," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 16, no. 1, pp. 66–75, 1994.

[27] D. Black, *The Theory of Committees and Elections*, 2nd ed. London, U.K.: Cambridge Univ. Press, 1963.

[28] D. R. Cox and E. J. Snell, *Analysis of Binary Data*, 2nd ed. London, U.K.: Chapman & Hall, 1989.

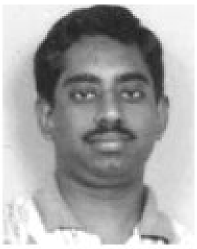
[29] T. V. Raman, "Audio System for Technical Readings," Ph.D. dissertation, Comput. Sci. Dept., Cornell Univ., Ithaca, NY, 1994.

[30] M. Okamoto, H. Imai, and K. Takagi, "Performance evaluation of a robust method for mathematical expression recognition," in *Proc. 6th Int. Conf. Document Analysis Recognition*, Seattle, WA, 2001, pp. 121–128.

[31] K.-F. Chan and D.-Y. Yeung, "PenCalc: A novel application of on-line mathematical expression recognition technology," in *Proc. Int. Conf. Document Analysis Recognition (ICDAR)*, Seattle, WA, 2001, pp. 774–778.

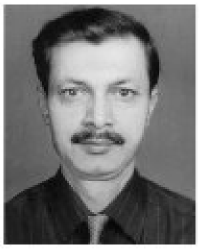
[32] U. Garain and B. B. Chaudhuri, "Input of handwritten mathematical expressions into machine coded Indian language documents," in *Proc. Indo European Conf. on Multilingual Technologies (IECMT)*, R. K. Arora, M. Kulkarni, and H. Darbari, Eds. Pune, Maharashtra, India, 2002, pp. 3–12.

[33] B. Hayes, "Speaking of mathematics," *Amer. Sci.*, vol. 84, no. 2, pp. 110–113, 1996.



Utpal Garain was born at Suri, West Bengal, India, in 1972. He received the B.E. and M.E. degrees in computer science and engineering from Jadavpur University, Kolkata, India, in 1994 and 1997, respectively.

He began his career as a Software Professional in industry and later on, as Research Personnel in the Indian Statistical Institute, Kolkata, India, where he is a full-time faculty now. He is one of the key scientists involved in the development of a bilingual (Devanagri and Bangla) OCR system, first of its kind in India. He has published several technical papers in reputed international journals and conferences. His areas of interest include digital document processing including optical character recognition for Indian language scripts, online character recognition, document data compression, etc.



B. B. Chaudhuri (F'00) received the B.Tech. and M.Tech. degrees from Calcutta University, India, in 1972 and 1974, respectively, and the Ph.D. degree from the Indian Institute of Technology, Kanpur, India, in 1980.

He is currently the head of the Computer Vision and Pattern Recognition Unit of Indian Statistical Institute, India, Kolkata, India. His research interests include pattern recognition, image processing, computer vision, natural language processing, and digital document processing. He has published more than 250 research papers in reputed journal/conferences and has authored three books. He is Associate Editor of *Pattern Recognition*, *Pattern Recognition Letters* and *VIVEK*

Dr. Chaudhuri has received many awards and prizes including Sir J. C. Bose Memorial Award (1986), M. N. Saha Memorial Award (twice: 1989, 1991), Homi Bhabha Fellowship award (1992), Dr. Vikram Sarabhai Research Award (1995), and C. Achuta Menon Prize (1996), Homi Bhabha award (2003) and Jawaharlal Nehru Fellowship (2004) for his contribution in the field of engineering sciences, Indian language processing, computer applications, etc. He is a Fellow of IAPR, IETE (India), National Academy of Sciences, and National Academy of Engineering (India).