Sādhanā Vol. 39, Part 1, February 2014, pp. 63–79. © Indian Academy of Sciences

A string matching based algorithm for performance evaluation of mathematical expression recognition

P PAVAN KUMAR*, ARUN AGARWAL and CHAKRAVARTHY BHAGVATI

School of Computer and Information Sciences, University of Hyderabad, Hyderabad 500 046, India
e-mail: pavan.ppkumar@gmail.com; aruncs@uohyd.ernet.in; chakcs@uohyd.ernet.in

MS received 17 April 2013; revised 26 July 2013; accepted 30 July 2013

Abstract. In this paper, we have addressed the problem of automated performance evaluation of Mathematical Expression (ME) recognition. Automated evaluation requires that recognition output and ground truth in some editable format like LaTeX, MathML, etc. have to be matched. But standard forms can have extraneous symbols or tags. For example, <mo> tag is added for an operator in MathML and \begin{array} is used to encoded matrices in LaTeX. These extraneous symbols are also involved in matching that is not intuitive. For that, we have proposed a novel structure encoded string representation that is independent of any editable format. Structure encoded strings retain the structure (spatial relationships like superscript, subscript, etc.) and do not contain any extraneous symbols. As structure encoded strings give the linear representation of MEs, Levenshtein edit distance is used as a measure for performance evaluation. Therefore, in our approach, recognition output and ground truth in LaTeX form are converted to their corresponding structure encoded strings and Levenshtein edit distance is computed between them.

Keywords. Mathematical expression recognition; performance evaluation; linear representation; Levenshtein edit distance.

1. Introduction

Recognition of mathematical expressions (MEs) is currently a challenging pattern recognition problem. ME recognition is a process of converting offline or online MEs into an editable encoding format like LaTeX (Lamport 1986), MathML (W3C 2010) etc. In offline recognition, MEs take the form of scanned images whereas for online, they are written using data tablets.

Several approaches are proposed in literature for the recognition of MEs and a recent survey on existing works is found in (Zanibbi & Blostein 2012). Some offline recognition systems are

*For correspondence

discussed in (Lee & Lee 1994; Lee & Wang 1997; Chaudhuri & Garain 2000; Suzuki *et al* 2003) and online systems in (Zanibbi *et al* 2002; Garain & Chaudhuri 2004; Vuong *et al* 2008). There is an imperative need to evaluate and compare the performance of different systems. It requires that the recognition output and ground truth in standard editable form be matched in an automated and generic manner. An obstacle to straight-forward matching is that the standard forms for representing MEs, such as LaTeX and MathML, use extraneous *tags* and *commands* in addition to the regular mathematical symbols and units. For example, in LaTeX, extraneous keywords or symbols like **\begin**{**array**} and **\end**{**array**} are used to encode matrices, enumerated functions, etc. In MathML also, extraneous tags or symbols like <mo>, <mi>, etc. are added to encode operators, identifiers, etc. These extraneous symbols are also involved in the matching process if standard forms are used. Performance evaluation can still be done by considering these additional symbols, but it is not intuitive.

In this paper, we have proposed an approach that allows performance evaluation to be done using standard forms and yet tackles the issues mentioned above. For that, we have introduced structure encoded strings which are analogous to the intermediate representation of compilers. Structure encoded strings retain the structure (spatial relationships like superscript, subscript, etc.) but do not contain any extraneous symbols. Recognition output and ground truth in LaTeX form are converted into their corresponding structure encoded strings. Levenshtein edit distance (Richard *et al* 2000) between the structure encoded strings is then used as a measure for performance evaluation. As structure encoded strings retain the linear representation of MEs, standard Levenshtein edit distance can be applied. This approach is intuitive as extraneous symbols are not involved in the matching process.

The paper is organized as follows. Existing approaches for performance evaluation are discussed in section 2. Proposed approach to evaluate performance is presented in section 3. In section 4, experimental results are discussed, summary and discussion was provided in section 5 and conclusions are provided in section 6.

2. Related work

The complexity of performance evaluation task has been discussed in (Lapointe & Blostein 2009) and several issues such as levels of representation used for MEs, quality of performance metrics, etc. have been identified. Garain & Chaudhuri (2005) have reported a performance index that depends on their recognition procedure and it is not useful as different researchers use different recognition approaches. In (Lin *et al* 2012), the authors have focused on performance evaluation of mathematical formula identification in the documents. Chan & Yeung (2001) have proposed an integrated measure for symbol recognition and structural analysis. This integrated recognition rate is given by the ratio of the number of correctly recognized symbols and operators to the total number of symbols and operators. Jin *et al* (2004) have proposed a bottom-up and top-down performance evaluation method based on tree matching and they have presented triplet tree for ME representation. Some studies (Okamoto *et al* 2001; Ashida *et al* 2006) have measured accuracy manually by separating different structures like *fraction, limit, root*, etc. and the number of such structures that are recognized correctly is computed.

For the first time, (Sain *et al* 2011) have proposed an approach (called as EMERS approach) for an automated evaluation in a generic manner (using editable format). In their approach, the authors have adopted tree distance based algorithm (Zhang & Shasha 1989) to find distance between two MathML (tree-like) representations. As tree matching (Tai 1979) is a non-trivial problem in terms of implementation complexity, the authors have approximated MathML trees

by their corresponding Euler strings (Aratsu *et al* 2009) and tree matching algorithm based on dynamic programming (Zhang & Shasha 1989) is used. Euler string (Aratsu *et al* 2009) is obtained by the depth-first traversal of MathML tree and in the traversal, if a node is encountered second time, complement of the node's label is added. In addition, Sain *et al* (2011) have suggested that edit cost should take the level of symbol into consideration as symbols in the superscript and subscript are smaller in size than their base symbols and the current recognition engines cannot be expected to handle characters irrespective of sizes. So, they have designed the cost functions in terms of levels of tags in MathML tree. For example, let b^i and a^{x^i} be recognized as bi and a^{xi}, respectively. Both of them are erroneous. In the first case, the error occurs in the first level script whereas for the second case, in the second level script. It would be judicious to give more penalty to the first error than the second one. As dynamic programming based tree distance algorithm has been used, their algorithm complexity is $O(n^4)$ (Sain *et al* 2011).

In EMERS approach, extraneous tags are also involved in matching that is not intuitive. Consider an example given in (Sain *et al* 2011): $\int dx/x$ is wrongly recognized as $1dx/x$. MathML representation for $\int$ is <mo>#x222b;</mo> and for 1 is <mn>1</mn>. By intuition, it needs only one edit (substitute 1 with $\int$) operation. But the edit operations are given by: Substitute <mn> with <mo> and Substitute 1 with #x222b;. The total edit cost is 2 as the extra tag <mn> is also involved in the matching.

Zanibbi *et al* (2011) have proposed performance metrics based on bipartite graphs at stroke level. In this approach, recognition output and ground truth interpretations are converted to bipartite graphs on which metrics based on hamming distances are defined. A competition (called as CROHME competition) has been conducted on the recognition of online handwritten mathematical expressions (CROHME 2011). Four research groups have submitted their ME recognition systems to the CROHME competition Mouchère *et al* 2011 and each of them is evaluated based on the following four aspects: stroke-level classification rate, symbol segmentation rate, symbol recognition rate and expression-level recognition rate. In this competition, some restrictions like only one symbol in superscript/subscript of function names, no recursive fraction, etc. have been imposed on the datasets. Recently, the second version of CROHME competition, CROHME2012 (Mouchère *et al* 2012) has been organized. In CROHME2012, seven systems are evaluated using augmented datasets that have less restrictions than those of the CROHME2011 competition.

In (Álvaro *et al* 2012), a method for unbaised evaluation has been proposed as different MathML structures can draw the same ME. In this method, a constrained parsing of an ME is performed to obtain a different MathML tree equivalent to the ground truth tree according to the model representation criteria. For evaluation, recognized tree and the obtained constrained parsed tree are compared using EMERS approach.

3. Proposed approach

Our approach for performance evaluation uses LaTeX format for illustration. It works for MathML also. To eliminate extraneous symbols, we propose the use of structure encoded strings. Recognition output and ground truth in LaTeX form are converted into their corresponding structure encoded strings. Levenshtein edit distance (Richard *et al* 2000) between the structure encoded strings is used as a measure for performance evaluation.

As suggested in (Sain *et al* 2011), we have also incorporated level information of symbols in the cost functions of Levenshtein edit distance measure. As Levenshtein edit distance is used, time complexity of the proposed approach is $O(n^2)$ (Cormen *et al* 1989). Our approach is shown

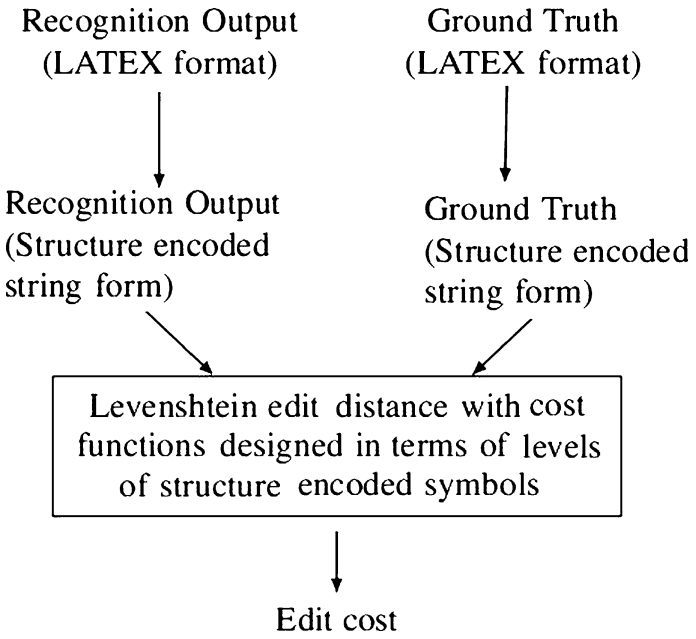

Figure 1. Proposed approach for performance evaluation.

in the form of a block diagram in figure 1. Structure encoded strings and the algorithm for performance evaluation are discussed in the next sections.

3.1 *Structure encoded strings*

Mathematical symbols can have spatially related symbols in its surrounding six regions namely, top-left, above, top-right, bottom-left, below and bottom-right regions, which are shown in figure 2. But in most of the MEs, for any symbol, the entire top-left, above and top-right regions form a single subexpression and similarly, the entire bottom-left, below and bottom-right regions form a single subexpression. The combined top-left, above and top-right (bottom-left, below and bottom-right) regions is called as *Northern (Southern)* region (shown in figure 2).

For example, consider the following ME:

$$\int\limits_{x-x_0<0} f(x)\,dx. \tag{1}$$

Here, subscript expression ($x - x_0 < 0$) of Integral symbol ($\int$) spans its bottom-left, below and bottom-right regions (entire southern region). For FRACTION symbol, numerator and denominator present in its above and below regions and so they form its northern and southern regions, respectively. For SQUAREROOT symbol, its *degree* is considered to be in the northern region and the contained expression in the southern region. For other symbols, northern and southern regions give superscript and subscript expressions, respectively. That means, atmost two different subexpressions are present around any symbol in most of the MEs. Identity also called as *label* (a positive integer) information of a symbol can be used to resolve its northern and southern regions.

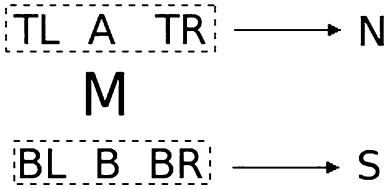

Figure 2. Possible spatial regions around a mathematical symbol **M**. Here, TL, A, TR, BL, B and BR refer to top-left, above, top-right, bottom-left, below and bottom-right regions, respectively. N and S refer to northern and southern regions, respectively.

There may be unusual cases, where two different subexpressions are present in the same region like ${}_nC_r$. In ${}_nC_r$, C has two different subexpressions n (bottom-left) and r (bottom-right) in its southern region. There can also be rare cases where more than two different subexpressions are present around a symbol. In some other cases, symbols enclosed in the accent symbols (present in above or below region) may have subexpressions in the other regions. For example, in $\hat{a}_i^2$, a is enclosed in the HAT accent symbol (above of a) and has superscript 2 (top-right subexpression) and subscript i (bottom-right subexpression). In this example, both HAT symbol and 2 are present in the northern region of a. All of these cases are considered in converting LaTeX strings into structure encoded strings.

Generation of structure encoded strings: Structure encoded strings are generated by scanning LaTeX strings from left to right. Let symbols in the structure encoded string be called as *Structure encoded symbols*. As symbols in LaTeX or any other encoding forms are not atomic, a mapping table that maps LaTeX symbols to structure encoded symbols is created. Each structure encoded symbol is given by a unique integer label. We have listed LaTeX keywords of different categories namely, Roman letters, numerals, Greek letters, Sum-like operators, Binary operators, Log-like symbols, Delimiters, etc. in a sequence and these keywords are consecutively assigned labels (starting from 1). For example, \frac (*fraction* in LaTeX) has four symbols and is made atomic by mapping it to an integer label (say, 219). Similarly, \sqrt ($\sqrt{\ }$ in LaTeX) has five symbols and is given an integer label, 140. In the process of scanning LaTeX string, keywords are captured and mapped to their corresponding labels using mapping table. Extraneous symbols are ignored while scanning and so this mapping not only creates atomic symbols but also eliminates extraneous symbols (that is similar to tokenization in compilers).

To maintain structure information, four special structure symbols are used to designate start and end of northern and southern subexpressions. That means, start structure symbol for northern (southern) region is used to designate start of northern (southern) subexpression. Similarly, end structure symbol for northern (southern) region is used to designate end of northern (southern) subexpression. Therefore, start and end structure symbols form the boundaries of northern and southern subexpressions so that these subexpressions can be unambiguously determined in the structure encoded string. Structure symbols also have unique labels, but do not refer to any mathematical symbol. Let N_s and N_e (S_s and S_e) designate start and end structure symbols for northern (southern) subexpression, respectively.

Therefore, conversion is a simple mapping process that maps LaTeX encoded symbols to the corresponding structure encoded symbols by scanning the input LaTeX string. In the process of scanning, extraneous keywords or symbols are discarded as well as start and end structure symbols of northern and southern subexpressions for different symbols (like *fraction, squareroot*, etc.) are inserted recursively to store structure information.

For example, consider an ME: a_2^{i+1} and its LaTeX string given by: a^{i+1}_2. In LaTeX, ^ and _ designate superscript (northern subexpression) and subscript (southern subexpression), respectively. Start and end structure symbols are added before and after scanning these subexpressions, recursively. Its structure encoded string is given by: a, N_s, i, +, 1, N_e, S_s, 2, S_e. As it is difficult to follow integer labels, here and in all the subsequent examples in the paper, we have shown symbols instead of their integer labels in the structure encoded string. It can be observed that N_s and N_e are written before and after superscript $(i+1)$ and similarly, S_s and S_e are written before and after subscript (2).

Consider a slightly unusual case where accented symbols have subexpressions. For example, $\hat{a}_i^2$. Its LaTeX string is given by: \hat{a}^2_i. Here, HAT symbol is considered to be northern subexpression of a. Logically, other subexpressions are meant for the composite symbol (accent symbol along with its enclosed symbol), not for the symbol alone and hence they appear after the composite symbol in the structure encoded string representation. Continuing the example, super and subscripts, 2 and i form the northern and southern subexpressions of $\hat{a}$ respectively and appear after it. Therefore, its structure encoded string is written as: a, N_s, ^, N_e, N_s, 2, N_e, S_s, i, S_e.

Wide accent symbols can have enclosed expressions in their northern or southern regions. For wide accent symbols having enclosed expression in the northern region, subscript if any present, is considered to be in the southern region. For example, in $\underbrace{a+b+\cdots+z}_{26}$, enclosed expression $a+b+\cdots+z$ is in the northern region of UNDERBRACE and the subscript 26 is in its southern region. Its structure encoded string representation is given by: UNDERBRACE, N_s, a, +, b, +, CDOTS, +, z N_e, S_s, 2, 6, S_e. Here, UNDERBRACE and CDOTS denote integer labels of *underbrace and ellipsis* $(\cdots)$, respectively. Similarly, for wide accent symbols with southern enclosed expression, superscript if any present, is considered to be in the northern region. For example, in $\overbrace{n(n-1)\cdots(n-m+1)}^{\text{m factors}}$, enclosed expression $n(n-1)\cdots(n-m+1)$ is in the southern region of OVERBRACE and superscript **m factors** is in its northern region.

Handling Multi-Line MEs: For Multi-Line MEs (MLMEs, defined in our earlier work (Pavan Kumar *et al* 2011)) like matrices, enumerated functions, etc. structure encoded strings are generated for all the elements (note that each element is again a ME) in all the rows in a row-major order. Special element and row delimiter symbols (which also have unique labels) are added after each element and row, respectively. Let D_e and D_r designate labels of the element and row delimiters of MLMEs respectively.

Consider the following MLME example (a determinant):

$$\left|\begin{array}{cc} a_0 & b_0 \\ a_1 & b_1 \end{array}\right| = 0. \tag{2}$$

Its LaTeX string is given by: \left| \begin{array}{cc} a_{0} & b_{0} \\ a_{1} & b_{1} \\ \end{array} \right| = 0. In LaTeX, element and row delimiters are given by & and \\, respectively and they are accordingly converted to D_e and D_r, respectively. Its structure encoded string is given by: |, a, S_s, 0, S_e, D_e, b, S_s, 0, S_e, D_r, a, S_s, 1, S_e, D_e, b, S_s, 1, S_e, D_r, |, =, 0. It has two rows with two elements in each row and the inserted delimiters, D_e and D_r. It can be observed that **\begin{array}** and **\end{array}** are removed in the structure encoded string.

Handling math fonts: Mathematical symbols can be typeset in different fonts namely, Calligraphic (\mathcal or {\cal }), Blackboard Bold (\mathbb), Fraktur (\mathfrak), etc. For example, Calligraphic form of A is given by {**\cal A**} ($\mathcal{A}$). To handle math fonts, we have defined start and end special symbols (that have unique labels) for each font. These start and end font symbols enclose the actual symbol. Let C_s and C_e denote integer labels of start and end Calligraphic font symbols, respectively. Structure encoded string representation for the above example is given by: C_s, A, C_e.

Consider an example: $\mathcal{A} \setminus \mathcal{B}$. Its LaTeX string is given by: {\cal A} \setminus {\cal B}. Its structure encoded string is given by: C_s, A, C_e, \, C_s, B, C_e.

Completeness: As structure encoded strings need to handle all types of symbols including rare cases, we have introduced two special symbols, *EMPTY* and *STACK*. *EMPTY* and *STACK* are dummy symbols that also have unique labels and are used to handle unusual and rare cases as discussed below:

(i) Consider an unusual case where two different subexpressions are present in the same (northern or southern) region for a given symbol. In this case, an *EMPTY* symbol is added and one of the subexpressions of the given symbol is handled by it. Consider ${}_nC_r$ for example. Here, pre-subscript n is considered the southern subexpression of *EMPTY* symbol and r is the southern subexpression of C. That is, *EMPTY* symbol is written first followed by its southern subexpression, followed by C and its southern subexpression. Structure encoded string for this example is thus given by: *EMPTY*, S_s, n, S_e, C, S_s, r, S_e.

(ii) Consider rare cases with more than two subexpressions around a symbol. Such symbols do not occur almost in any ME, but LaTeX can generate them. For such cases also, *EMPTY* symbols are added to handle the extra subexpressions as discussed below.

- Consider ${}_q^p\sum_r^s$. Here, $\sum$ has four different subexpressions around it. Here, one EMPTY symbol is added to handle pre-super and pre-subscripts (top-left and bottom-left). That means, p and q are considered as the northern and southern subexpressions for the *EMP* symbol and the super and subscripts, s and r, are considered for $\sum$ symbol. Its structure encoded string is given by: *EMPTY*, N_s, p, N_e, S_s, q, S_e, $\sum$, N_s, s, N_e, S_s, r, S_e.
- Consider ${}_b^a\prod_{i=1}^{n}{}_c^d$. Here, $\prod$ has six different subexpressions around it. In this case, two *EMPTY* symbols are added, one to handle top-left and bottom-left subexpressions and the other one to handle above and below subexpressions. Actual symbol $\prod$ handles the remaining two subexpressions. Its structure encoded string is given by: *EMPTY*, N_s, a, N_e, S_s, b, S_e, *EMPTY*, N_s, n, N_e, S_s, i, =, 1, S_e, $\prod$, N_s, d, N_e, S_s, c, S_e.

(iii) There can be rare cases where two or more symbols are vertically stacked one over another. For example, in $\overset{\alpha}{\omega}$, α is above ω and in $\underset{\mu}{\nu}$, μ is below ν. To handle these cases, we have introduced start and end *STACK* symbols that are used to enclose the actual stacked symbols. Let T_s and T_e denote integer labels of start and end *STACK* symbols, respectively.

In LaTeX, keywords **\overset**, **\underset**, **\stackrel**, etc. are used to generate vertically stacked symbols (like <mover> and <munder> in MathML). In **\overset** {x}{y} or **\stackrel** {x}{y}, x is above (northern region) y (whose structure encoded string can be written as y, N_s, x, N_e). As y and x are vertically stacked, they are enclosed between T_s and T_e. Its structure encoded string representation is thus given by: T_s, y, N_s, x, N_e, T_e. Similarly, in **\underset** {x}{y}, x is below (southern region) y and its structure encoded string is given by: T_s, y, S_s, x, S_e, T_e.

Consider a complex example: $\overset{\beta}{\underset{\Delta}{\tau}}$. Its LaTeX string is given by: \overset{\beta}{\underset {\Delta }{\tau}}. Here, β is above (northern region) $\underset{\Delta}{\tau}$ and in $\underset{\Delta}{\tau}$, Δ is below (southern region) τ. Structure encoded string for $\underset{\Delta}{\tau}$ is given by: T_s, τ, S_s, Δ, S_e, T_e and for the entire expression is thus given by: T_s, T_s, τ, S_s, Δ, S_e, T_e, N_s, β, N_e, T_e.

Uniqueness: For performance evaluation, structure encoded strings should give a unique representation. That is, if two structure encoded strings are the same, both should refer to the same ME. Structure encoded strings give a unique representation due to the following reasons:

(i) Symbols in an expression or any subexpression (northern or southern) are ordered in a left to right manner.
(ii) Northern and southern subexpressions for any symbol are enclosed between their corresponding start and end structure symbols recursively.
(iii) Southern subexpression is always handled after the northern subexpression (if present).
(iv) *EMPTY* and *STACK* symbols are used to handle rare cases.

Length: As structure symbols are added, the length of structure encoded string is more than the number of symbols in the given ME. Let n be the number of symbols in the given ME. Let n_1 be the number of symbols (out of n) having only one of the northern or southern subexpressions and n_2 be the number of symbols (out of n) having both the subexpressions. Two structure symbols (start and end of one of the subexpressions) are added for each of the n_1 symbols and for each of the n_2 symbols, four structure symbols (start and end of both the subexpressions) are added. Therefore, total number of structure symbols is $2n_1 + 4n_2$ and the total length of the string is given by $n + 2n_1 + 4n_2$. As $n_1 < n$ and $n_2 < n$, total length of structure encoded string is $O(n)$.

Levels: Levels give nested depths of the symbols in an ME. In the conversion process, levels of the symbols in the structure encoded string are also found. Symbols of any expression (or subexpression) which are not present in the northern or southern region of any symbol are called its *Baseline* symbols. All the baseline symbols of a ME are at the same level '0'. For example, a is the baseline symbol of ME $a^{i^2_j}$ and is at level 0. Any symbol present in the northern or southern regions of a symbol (called a base symbol) is one level greater than that of the base symbol. i is the baseline symbol of the northern subexpression i^2_j, of a and so is at level 1. 2 and j are the northern and southern subexpressions of i, respectively and are at level 2.

Levels of start and end structure symbols of northern and southern subexpressions are assigned the same values as those of the baseline symbols of the subexpressions. For example, structure encoded string of a^2 is: a, N_s, 2, N_e. Here, 2 is the baseline symbol of northern subexpression of a and is at level 1. Levels of N_s and N_e are assigned the same value 1. Similarly, levels of element and row delimiters of MLMEs are assigned the same values as those of baseline symbols of the elements.

Intuitiveness: In structure encoded string representation, as northern and southern regions are considered standalone, it appears to be non-intuitive. But in most of the cases, mathematical symbols have atmost two arguments (numerator/denominator for fraction, degree/contained expression for squareroot, super/subscripts for others, etc.). Hence proposed approach that considers standalone northern and southern subexpressions is not far from intuition.

If sub-regions are considered separately so that each of the them has its corresponding start and end structure symbols, uniqueness property may be violated. For example, both $\sum_i$ and $\sum\limits_{i}$ represent the same ME (as i is the lower limit of $\sum$ semantically). In the first ME, i is in the bottom-right region whereas in the second one it is in the below region. Let BR_s and BR_e (B_s and B_e) denote start and end structure symbols for bottom-right (below) region. Structure encoded strings for the above expressions if sub-regions are considered separately are respectively given by: $\sum, BR_s, i, BR_e$ and $\sum, B_s, i, B_e$ (both are different). If southern subexpression is considered standalone, both the above MEs are given by the same structure encoded string: $\sum, S_s, i, S_e$.

In view of the above reasons, northern and southern regions are considered standalone, and *EMPTY* and *STACK* symbols are used for rare cases in our structure encoded string representation. If tight performance evaluation (that gives more importance to spatial locations than semantics) is needed, sub-regions have to be considered separately. Tight evaluation is also feasible in our representation by defining and using start and end structure symbols for all the sub-regions.

Generality: The idea of the structure encoded string can be applicable to other similar structures like Indian language scripts and chemical equations. Indian language scripts have vowel and consonant modifiers in the northern and southern regions of base consonants, respectively. Hence proposed approach can be used for performance evaluation of the OCR systems for Indic scripts. Similarly, chemical equations have ionic information (like oxidation state) and the number of instances of an atom in the northern and southern regions, respectively. Hence matching of chemical structures can be performed using the proposed approach.

3.2 *Algorithm for performance evaluation*

As mentioned earlier, proposed approach for performance evaluation takes LaTeX strings of the recognition output and ground truth and converts them into their corresponding structure encoded strings. Levenshtein edit distance (Richard *et al* 2000) between these two strings is then used as a measure for performance evaluation. All types of errors such as recognition, structural errors; etc. can be handled using edit distance on structure encoded strings. For example, if $a^2 + b^2$ is recognized as a^2tb^2, it is a recognition error where $+$ is misrecognized as t. Similarly, if it is recognized as $a2 + b^2$, superscript relationship between a and 2 is lost, and it is a structural error.

Levenshtein edit distance (Richard *et al* 2000) between two strings gives the minimum number of edit operations (substitution, insertion and deletion) required to transform one string to the other. The dynamic programming approach (Cormen *et al* 1989) uses a matrix to hold partial distances between all prefixes of the first string and all prefixes of the second. The final distance between the given two strings is computed using it. The time complexity of the dynamic programming approach to computing Levenshtein Edit Distance is $O(n^2)$ (Cormen *et al* 1989). In terms of time complexity, this string edit distance is better than the tree edit distance (which is $O(n^4)$).

Let G and R be structure encoded strings of the ground truth and recognition output respectively and let p and q be their respective lengths. Let $G[i]$ and $R[i]$ be the i^{th} elements of the strings G and R, respectively. Let E be the dynamic programming matrix, which is of size $(p+1) \times (q+1)$. Each cell in E, $E[i, j]$, contains the edit distance by considering the first i elements in G and the first j elements in R and $E[p, q]$ gives the final edit cost. Distance between empty strings, the trivial case, is zero. This implies, $E[0, 0] = 0$. Each cell $E[i, j]$, is computed as follows:

$$E[i, 0] = E[i-1, 0] + I(i) \tag{3}$$

$$E[0, j] = E[0, j-1] + D(j) \tag{4}$$

$$E(i, j) = \min \begin{cases} E(i-1, j-1) + S(i, j) \\ E(i-1, j) + I(i) \\ E(i, j-1) + D(j). \end{cases} \tag{5}$$

Here, $S(i, j)$ denotes the cost of substituting $R[j]$ with $G[i]$. $I(i)$ denotes the cost of inserting $G[i]$ and $D(j)$ denotes the cost of deleting $R[j]$. We have used the same cost function as given in (Sain *et al* 2011), which is $1/(l+1)$ where l is the level of a symbol. As the level of a symbol increases, its cost decreases and this cost function penalizes errors in symbols more than their nested (northern and southern) ones.

If all the errors need to be given same cost, irrespective of levels, it is still possible by setting l to 0 in the cost function which gives a cost of 1 to all the errors. Cost functions that are designed in terms of levels of structure encoded symbols are given by equations (6), (7) and (8). $L(X)$ denotes the level of symbol X. For substitution cost, level of ground truth symbol is used as it is being substituted. For start and end structure symbols (as well as for start and end font and *STACK* symbols), only half of the computed costs are assigned as each start and end pair constitute a single (northern or southern) region.

$$S(i, j) = \begin{cases} 0 & \text{if } G[i] = R[j] \\ \frac{1}{L(G[i])+1} & \text{otherwise} \end{cases} \tag{6}$$

$$I(i) = \frac{1}{L(G[i]) + 1} \tag{7}$$

$$D(j) = \frac{1}{L(R[j]) + 1}. \tag{8}$$

Consider the example discussed in section 2. LaTeX strings of the ground truth and recognition output are given by: \int \frac{dx}{x} and 1 \frac{dx}{x}. Using proposed approach, structure encoded strings of ground truth and recognition output are respectively given by: $\int$, FRACTION, N_s, d, x, N_e, S_s, x, S_e and 1, FRACTION, N_s, d, x, N_e, S_s, x, S_e. FRACTION denotes the integer label for \frac. Levels of structure encoded symbols of ground truth in order are: 0, 0, 1, 1, 1, 1, 1, 1, 1 and those of structure encoded symbols of recognition output are: 0, 0, 1, 1, 1, 1, 1, 1, 1, 1.

Hence using the proposed approach, only one edit operation is needed: *Substitute 1 with* $\int$ *at position 1 at level 0 with cost 1*. The total edit cost is 1 which is as per intuition. Here, positions (start from 1) refer to structure encoded string of the recognition output as edit operations are assumed to be performed on the recognition output.

In our implementation, the edit operations can have further interpretations. For example, the edit operation like insertion of structure symbol N_s at position x can be interpreted in the following manner: Subsequent symbols (after the inserted symbol N_s) form the northern subexpression of the symbol at position, $x-1$. Northern and southern sub-expressions for different symbols are

also accordingly interpreted as discussed in section 3.1. For instance, northern sub-expression of FRACTION symbol is interpreted as its numerator while that of SQUAREROOT symbol is interpreted as its degree.

It is to be noted that edit distance is not normalized between 0 and 1. As edit distance gives an estimate of the minimum number of edits to correct an ME, it reflects the correction efforts of a proof reader. Theoretically, its value ranges from 0 to ∞. If there are no edits (perfect recognition), edit distance is zero. More number of edits results in a greater edit distance. Hence the lesser the value of edit distance, the lesser the correction effort and the more efficient the ME recognition system. Hence edit distance values can be used to relatively compare correction efforts for different systems and so their normalization is not needed.

4. Experimental results and discussion

In this section, some experimental results are presented. As EMERS approach (Sain *et al* 2011) is the only one in the literature where performance evaluation is done using standard forms in a generic manner, proposed approach is compared with this approach. For comparison, we have used MEs from the publicly available Infty database (Infty 2009).

One detailed comparative example is given and table 1 summarizes results on more number of MEs from Infty database. All the recognition outputs are obtained using InftyReader (Suzuki *et al* 2003), an OCR system for MEs (that is also publicly available). Both the proposed and EMERS approaches are compared under similar conditions. For both of them, edit operations are assumed to be performed on the recognition output and the notion of *level* as well as the cost functions (defined in terms of levels) are same. Consider the following example.

Ground truth: $j \in P = \cup_{-1 \leq i \leq 2^{s-1}} P(k; i)$

Recognition output: $j \epsilon P = \cup_{-1 \leq i \leq 2s-1} P(k; i)$

Here, $\in$ is misrecognized as ϵ (symbol recognition error) and superscript relationship between 2 and $s - 1$ is lost (structural error).

4.1 *EMERS approach*

MathML for ground truth:

```
1. <math> <mi>j</mi> <mo>#x03f5</mo>
2. <mi>P</mi> <mo>=</mo> <msub>
3. <mo>#x222a</mo> <mrow> <mo>-</mo>
4. <mn>1</mn> <mo>#x2264</mo> <mi>i</mi>
5. <mo>#x2264</mo> <msup> <mn>2</mn>
6. <mrow> <mi>s</mi> <mo>-</mo>
7. <mn>1</mn> </mrow> </msup> </mrow>
8. </msub> <mi>P</mi> <mo maxsize="1">
9. (</mo> <mi>k</mi> <mi>;</mi>
10. <mi>i</mi> <mo maxsize="1">)</mo> </math>
```

MathML for recognition output:

```
1. <math> <mi>j</mi> <mi>#x03b5</mi>
```

Table 1. Edit costs of EMERS and Proposed approaches on different types of MEs from Infty database (Infty 2009). Recognition outputs are obtained using InftyReader (Suzuki *et al* 2003).

S.No	Ground truth	Recognition output	Edit cost	
			EMERS approach	Proposed approach
1	$W_{1,\epsilon} = \{W_{1,\epsilon}(p) : p \in \Lambda\}$	$W_{1,\epsilon} = \{W_{1,\epsilon}(p) : pe\Lambda\}$	2	1
2	$g : (x, y) \rightarrow (x, g_y x)$	$g : (x, y) -, (x, g_y x)$	3	2
3	$\left\lvert \int_{u(\xi_1+i\eta_0)}^{u(\xi_2+i\eta_0)} \frac{dc}{\Theta_D(c)} \right\rvert \leq \frac{1}{\pi} \log \frac{\xi_1}{\xi_2} + \frac{\pi}{4}$	$\left\lvert \int_{u(\xi_1+i\eta_0)}^{u(\xi.+i\eta_0)} \frac{dc}{\Theta_D(c)} \right\rvert \leq \frac{1}{\pi} \log \frac{\xi_1}{\xi_2} + \frac{\pi}{4}$	1	0.67
4	$\coprod_q \Lambda_b^q = \Lambda_{bR}^0$	$\coprod_q \Lambda_b^q = \Lambda_{bR}^o$	1	0.5
5	$k^*(z) \leqq k^*(w_0) + \epsilon/4^{2n+2}$	$k^*(z) \leqq k^*(w_0) + \epsilon[4^{2n+2}$	2	1
6	$S_{0R} = S_0^+ \cup S_0^-$	$S_{0R} = S_0^+ u S_0^-$	2	1
7	$\sum(g) = D^{k+1} \times S^q \cup_g S^k \times D^{q+1}$	$\sum(g) = D^{k+1} \times S^q U_g S^k \times D^{q+I}$	3	1.5
8	$\Lambda_{2R}^0 = \left(\cup_{q=0,1} \sum_{2,0}^{0,q,+}\right)$	$\Lambda_{2R}^o = \left(\cup_{q=0,1} \sum_{2,0}^{0.q,+}\right)$	2	1
9	$D_1(S)I_{(n)} + S_{(n)}K = (I_{(n)} - S_{(n)}K + S_{(n)}K = K$	$D_1(S)I_{(n)} + S_{(n)}K = (I_{(n)} - S_{(n)}K + S_{tn)}K = K$	1	0.5
10	$\lim \mid P_n - f \mid_{U_0} = 0$	$\lim \mid P_n - f \mid_{U_O} = 0$	0.67	0.33
11	$\int_0^\epsilon \log \gamma(1/\delta) d\delta = \int_{1/\epsilon}^\infty \frac{\log \gamma(t)}{t^2} dt$	$\int_0^\epsilon \log \gamma(1/\delta) d\delta = \int_{1'\epsilon}^\infty \frac{\log \gamma(t)}{t^2} dt$	1	0.5
12	$\frac{\partial \rho}{\partial t} + \nabla j = 0$	$\frac{\partial \rho}{\partial t} + \nabla \cdot j \rightarrow = 0$	4	3
13	$\int_{u(\zeta)}^{u(\xi')} \frac{dc}{\Theta_R(c)} + \int_{u(\xi')}^{u(\xi)} \frac{dc}{\Theta_R(c)}$	$\int_{u(\zeta)}^{u(\xi\gamma} \frac{dc}{\Theta_R(c)} + \int_{u\langle\xi')}^{u(\xi)} \frac{dc}{\Theta_R(c)}$	1.5	1
14	$K_i^*(p) = \int \mid K_i(p, e) \mid^{n/i} d\sigma$	$K_i^*(p) = \int \mid K_i(p, e) \mid n/ * d\sigma$	2	1
15	$\check{H}^0(G, \check{A}(L)) \rightarrow H^1(G, A(L))^*$	$\check{H}^0(G, \check{A}(L)) \rightarrow H^f(G, A(L))^*$	1	0.5
16	$g(z) = \frac{1}{\Omega_{2n-1}} \int_{\lvert\lvert w\rvert\rvert=1} g(w) \frac{1-\lvert\lvert z\rvert\rvert^2}{\lvert\lvert w-z\rvert\rvert^{2n}} \omega_{2n-1}$	$g(z) = \frac{1}{\Omega_{2n-1}} \int_{\}\lvert w\rvert I=1} g(w) \frac{1-\lvert\lvert z\rvert\rvert^2}{\lvert\lvert w-z\rvert\rvert^{2n}} \omega_{2n-1}$	2	1
17	$p = (0, +dx_1) \in T^*V$	$p = (0, +dx_1)eT^* \triangledown$	4	2
18	$a \times (bc) = (a \cdot c)b - (a \cdot b)c$	$a \times (bxc) = (a \cdot c)b - (a \cdot b)c$	2	1

Table 1. (continued)

S.No	Ground truth	Recognition output	Edit cost	
			EMERS approach	Proposed approach
19	$\varphi^2(r) = \int_{C_r} \mid f \mid^2 dv = \int_{C_r} (U^2 + V^2) dv$	$\varphi^2(r) = \int_{C_r} \mid f \mid^2 dv = \int_{C_r} (U^2 + \nabla^2) dv$	2	1
20	$= \frac{1}{(z-\zeta)^2}[Tvz + Tv\zeta - 2\frac{Pvz - Pv\zeta}{z-\zeta} + o(1)]$	$= \frac{1}{(z-\zeta)^2}[Tvz + Tv\zeta - 2\frac{Pvz - pV\zeta}{z-\zeta} + 0(1)]$	4.5	3.5
21	$\begin{vmatrix} x & y & 1 \\ x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \end{vmatrix} = 0$	$\begin{vmatrix} + & y & 1 \\ x_0 & y0 & 1 \\ x_1 & y1 & 1 \end{vmatrix} = 0$	3	2
22	$G_x = g \in G \mid gx = x$	$G_X = g \in G \mid gx = x$	0.5	0.5
23	$\xi_i = T(G_r \times G_s D(\sigma^s(e_i)))$	$\xi_i = T(G_r \times C_S D(\sigma^s(e_i)))$	1.5	1.5
24	$D(S) \rightarrow D(S_0)$	$D(S) \rightarrow D(S_0\rangle$	1	1
25	$N = (M^*/Ss)^* \simeq (M^*/St)^*$	$N = (M^*/Ss)^* = (M^*/St)^*$	1	1
26	$N_1 \cup_\psi N_2 = 2SV_j$	$N_1 \cup \psi N_2 = 2SV_j$	0.5	0.5

```
2. <mi>P</mi> <mo>=</mo> <msub>
3. <mo>#x222a</mo> <mrow> <mo>-</mo>
4. <mn>1</mn> <mo>#x2264</mo> <mi>i</mi>
5. <mo>#x2264</mo> <mn>2</mn> <mi>s</mi>
6. <mo>-</mo> <mn>1</mn> </mrow> </msub>
7. <mi>P</mi> <mo maxsize="1">(</mo>
8. <mi>k</mi> <mi>;</mi> <mi>i</mi>
9. <mo maxsize="1">)</mo> </math>
```

Edit operations (costs are shown upto two decimal points):

1 Substitute <mi> and </mi>(that encloses #x03b5 (ϵ)) with <mo> and </mo> at level 0 with cost 1 (in line 1).
2 Substitute #x03b5 with #x03f5 (ϵ with $\in$) at level 0 with cost 1 (in line 1).
3 Insert <msup> before <mn>2</mn> (in line 5) and </msup> after <mn>1</mn> (in line 6) at level 2 with cost 0.33.
4 Insert <mrow> before <mn>s</mn> (in line 5) and </mrow> after <mn>1</mn> (in line 6) at level 2 with cost 0.33.
Total cost of edit operations = 2.66.

Here, line numbers refer to MathML of recognition output as edit operations are assumed to be performed on recognition output. First two edit operations correct the recognition error and #x03b5 and #x03f5, denote MathML symbols for ϵ and $\in$ respectively. First edit operation considers substitution of the extraneous tags <mo> and </mo>, and so an additional cost of 1 is incurred which is not intuitive. Last two insert operations correct the structural error. If a super or subscript expression has more than one symbol, it should be enclosed between <mrow> and </mrow> in MathML. As such, last edit operation considers insertion of these extraneous tags with an additional cost of 0.33. It can be seen that edit operations on the opening and closing tags (like <mi> and </mi>, <msup> and </msup> etc.) are combined in this approach (Sain *et al* 2011).

4.2 *Proposed approach*

LaTeX for Ground Truth: j \in P = \cup_{-1 \leq i \leq 2^{s-1}}P(k;i).
LaTeX for Recognition output: j \epsilon P = \cup_{-1 \leq i \leq 2s-1}P(k;i).
Structure encoded string for Ground Truth: j, $\in$, P, =, $\cup$, S_s, -, 1, $\leq$, i, $\leq$, 2, N_s, s, -, 1, N_e, S_e, P $(k; i)$.
Levels of structure encoded symbols of Ground Truth: 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 1, 0, 0, 0, 0, 0, 0.
Structure encoded string for Recognition output: j, ϵ, P, =, $\cup$, S_s, -, 1, $\leq$, i, $\leq$, 2, s, -, 1, S_e, P $(k; i)$.
Levels of structure encoded symbols of Recognition output: 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0.
Edit operations (costs are shown upto two decimal points):

1. Substitute ϵ with $\in$ at position 2 at level 0 with cost 1.00.
2. Insert N_s at position 13 (before s) at level 2 with cost 0.17.
3. Insert N_e at position 16 (after 1) at level 2 with cost 0.17.
4. Total cost of edit operations = 1.34.

Here, positions (start from 1) refer to structure encoded string of the recognition output. First substitute operation corrects the symbol recognition error and the next two insert (of structure symbols) operations correct the structural error. As mentioned earlier, cost of the insertion operations for each of the structure symbols, N_s and N_e, is 0.17 (half of 0.33, obtained using equation (7)). As extraneous symbols are not involved, edit cost is reduced to 1.34.

If level information is considered, it is to be noted that half of the edit costs are considered for each of the start and end structure symbols. Hence the edit costs for the start and end structure symbols as well as for the opening and closing MathML tags, at a given level are equal.

Ignoring level information (setting $L(x) = 0\ \forall x$, so that a cost of one is assigned to all the errors), edit costs for EMERS and the proposed approaches are given by 7 and 3 respectively. In our approach, edit operations on start and end structure symbols are considered separately unlike EMERS approach. To facilitate comparison, we have treated the operations on opening and closing tags separately due to which an edit cost of 7 is obtained for EMERS approach.

Our approach is compared with EMERS approach by taking more examples that cover different fields like Algebra, Calculus, Geometry etc. from Infty database. Edit costs for these examples using both the approaches, are shown in table 1.

5. Summary and discussion

It can be observed from table 1 that EMERS approach shows higher edit cost values than for ours for most of the MEs as extraneous tags are also involved in the EMERS approach. Both EMERS and the proposed approaches give same cost for the last five entries in table 1. In these examples, extraneous tags are not involved in EMERS approach and the cases where this phenomenon occurs, are discussed below.

(i) Symbol mis-recognition occurs within the same class like operators, identifiers, numerals, etc. For example, if j is misrecognized as J, their tags (<mi>) are matched as both are identifiers. Identifier tag matching can be seen for the entries 22 and 23, in table 1. In 24$^{\text{th}}$ entry of the table, operator) is misrecognized as $\rangle$ and in 25$^{\text{th}}$ entry, operator $\simeq$ is misrecognized as =. For these two, operator (<mo>) tags are matched.

(ii) Superscript (subscript) relationship is lost and the superscript (subscript) has a single symbol. In this case, <mrow> tags are not needed to group symbols in the

Table 2. EMERS vs proposed approaches.

Characteristic	EMERS approach	Proposed approach
Matching type	Tree based	String based
Matching units	MathML trees	Structure encoded strings
Extraneous symbols	Involved except for the cases discussed in section 5	Not involved
Edit cost	Always greater than or equal to that of proposed approach (equal if extraneous tags are not involved)	Always less than or equal to that of EMERS approach
Time complexity	$O(n^4)$	$O(n^2)$

super/subscripts. This case is shown in the 26th entry of table 1 (where subscript relationship between $\cup$ and ψ is lost).

In other cases, EMERS approach shows higher edit costs. EMERS approach can be modified to avoid contribution of the extraneous tags to the total edit cost by assigning zero costs to the edit operations involving these tags. But still, as tree edit distance is used, its time complexity is $O(n^4)$ (Sain *et al* 2011). Differences between EMERS and the proposed approaches are highlighted in table 2.

6. Conclusions and future work

In this paper, we have addressed the problem of an automated performance evaluation of ME recognition using standard forms like LaTeX. For this, we have proposed structure encoded strings that retain the structure and eliminate extraneous symbols. Recognition output and ground truth in LaTeX format are converted to structure encoded strings and Levenshtein edit distance is used to find distance between them. This approach is intuitive as only required symbols are involved in the matching process. Cost functions of edit distance are designed in terms of levels of structure encoded symbols. Length of structure encoded string is $O(n)$ and as string edit distance is used, time complexity is $O(n^2)$ which is better than $O(n^4)$ for the earlier EMERS approach. Although it is illustrated using LaTeX in this paper, note that it is a general approach and can be used for MathML or any other encodings. It is also applicable to other domains like chemical structures and OCR systems of Indic scripts.

As structure encoded string representation is unique and complete, in future, it can be incorporated into ME recognition systems so that structure encoded string representation can be generated instead of LaTeX, MathML, etc.

References

Álvaro F, Sánchez J-A and Benedí J-M 2012 Unbiased evaluation of handwritten mathematical expression recognition. In *International Conference on Frontiers in Handwriting Recognition (ICFHR).*

Aratsu T, Hirata K and Kuboyama T 2009 Approximating tree edit distance through string edit distance for binary tree codes. In *Proceedings of the 35th Conference on Current Trends in Theory and Practice of Computer Science,* SOFSEM '09, pages 93–104, Berlin, Heidelberg. Springer-Verlag

Ashida K, Okamoto M, Imai H and Nakatsuka T 2006 Performance evaluation of a mathematical formula recognition system with a large scale of printed formula images. In *Second International Conference on Document Image Analysis for Libraries* 320–331

Chan K-F and Yeung D-Y 2001 Error detection, error correction and performance evaluation in on-line mathematical expression recognition. *Pattern Recognition* 34(8): 1671–1684

Chaudhuri B B and Garain U 2000 An approach for recognition and interpretation of mathematical expressions in printed document. *Pattern Analysis and Applications* 3(2): 120–131

Cormen T H, Leiserson C E and Rivest R L 1989 *Introduction to Algorithms.* The MIT Press and McGraw-Hill Book Company

CROHME 2011 Competition on recognition of online handwritten mathematical expressions. http://www.isical.ac.in/ crohme/

Garain U and Chaudhuri B 2005 A corpus for OCR research on mathematical expressions. *International Journal on Document Analysis and Recognition* 7(4): 241–259

Garain U and Chaudhuri B B 2004 Recognition of online handwritten mathematical expressions. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics* 34(6): 2366–2376

Infty 2009 Printed mathematical expressions. http://www.inftyproject.org/download/InftyMDB-1.zip

Jin J-M, Jiang H-Y, Wang K and Wang Q-R 2004 Automatic performance evaluation of mathematical expression recognition. *Proceedings of International Conference on Machine Learning and Cybernetics* 6: 3595–3599

Lamport 1986 *LaTeX: A Document Preparation System.* Addison-Wesley

Lapointe A and Blostein D 2009 Issues in performance evaluation: A case study of math recognition. In *Proceedings of International Conference on Document Analysis and Recognition,* 1355–1359

Lee H J and Lee M C 1994 Understanding mathematical expressions using procedure-oriented transformation. *Pattern Recognition* 27(3): 447–457

Lee H J and Wang J S 1997 Design of a mathematical expression understanding system. *Pattern Recognition Lett.* 18(3): 289–298

Lin X, Gao L, Tang Z, Lin X and Hu X 2012 Performance evaluation of mathematical formula identification. *IAPR International Workshop on Document Analysis Systems* 287–291

Mouchère H, Viard-Gaudin C, Kim D H, Kim J H and Garain U 2011 CROHME2011: Competition on recognition of online handwritten mathematical expressions. In *2011 International Conference on Document Analysis and Recognition* 1497–1500

Mouchère H, Viard-Gaudin C, Kim D H, Kim J H and Garain U 2012 ICFHR 2012 - Competition on recognition of online mathematical expressions (CROHME2012). In *International Conference on Frontiers in Handwriting Recognition (ICFHR)*

Okamoto M, Imai H and Takagi K 2001 Performance evaluation of a robust method for mathematical expression recognition. In *Proceedings of International Conference on Document Analysis and Recognition (ICDAR)* 121–128

Pavan Kumar P, Agarwal A and Bhagvati C 2011 A rule-based approach to form mathematical symbols in printed mathematical expressions. *Multi-disciplinary Trends in Artificial Intelligence, MIWAI 2011, LNCS/LNAI* 7080: 181–192

Richard D O, Peter H E and David S G 2000 *Pattern Classification (2nd Edition).* Wiley

Sain K, Dasgupta A and Garain U 2011 EMERS: A tree matching-based performance evaluation of mathematical expression recognition systems. *Int. J. on Document Analysis and Recognition* 14(1): 75–85

Suzuki M, Tamari F, Fukuda R, Uchida S and Kanahori T 2003 Infty- an integrated OCR system for mathematical documents. In *Proceedings of ACM Symposium on Document Engineering 2003* 95–104. ACM Press

Tai K-C 1979 The tree-to-tree correction problem. *J. ACM,* 26(3): 422–433

Vuong B-Q, Hui S-C and He Y 2008 Progressive structural analysis for dynamic recognition of on-line handwritten mathematical expressions. *Pattern Recognition Lett.* 29(5): 647–655

W3C 2010 Mathml. http://www.w3.org/Math/

Zanibbi R and Blostein D 2012 Recognition and retrieval of mathematical expressions. *International Journal on Document Analysis and Recognition* 15(4): 331–357

Zanibbi R, Blostein D and Cordy J R 2002 Recognizing mathematical expressions using tree transformation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24(11): 1455–1467

Zanibbi R, Pillay A, Mouchere H, Viard-Gaudin C and Blostein D 2011 Stroke-based performance metrics for handwritten mathematical expressions. In *2011 International Conference on Document Analysis and Recognition (ICDAR)* 334–338

Zhang K and Shasha D 1989 Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Computing* 18(6): 1245–1262