11

OCR of Printed Mathematical Expressions

Utpal Garain and Bidyut B. Chaudhuri

11.1 Introduction

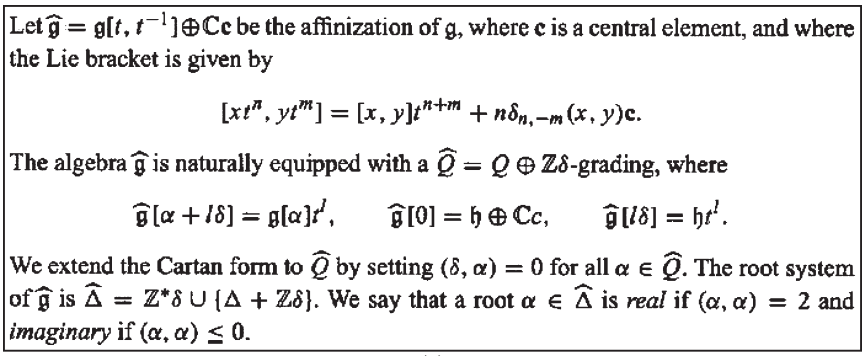
Automatic recognition of mathematical expressions (hereafter, referred as expressions) is one of the challenging pattern recognition problems of significant practical importance. Such a recognition task is required while converting printed scientific documents into electronic form or to aid the visually impaired persons in reading scientific documents.

This chapter discusses the key issues involved in the development of a system for recognition of printed documents containing expressions. In fact, studies on this topic date back to 1960s when Anderson [1] proposed a syntax-directed scheme for recognition of hand-printed expressions. Several studies have been reported so far and surveyed in [4, 5, 13].

However, the present discussion starts with showing limitations of existing optical character recognition (OCR) systems in converting scientific papers into corresponding electronic form. Figure 11.1 demonstrates one such example obtained from a popular OCR system, namely ABBYY Fine Reader 6.0.[1] The limitation arises because such a system fails to recognize expressions that often appear in scientific documents.

Instead of developing new OCR systems for scientific documents, we put emphasis on upgrading the existing ones by additional processing modules for expressions in documents. Since the presence of expressions disturbs typical OCR system (not trained for expression recognition), the identification and extraction of expression zones, therefore, could be the first step in this module. It permits an existing OCR engine to process the normal text portion as usual, whereas the extracted expressions can be tackled by a system specially designed for expression recognition.

[1] www.abbyy.com.



Let $\widehat{\mathfrak{g}} = \mathfrak{g}[t, t^{-1}] \oplus \mathbb{C}\mathbf{c}$ be the affinization of $\mathfrak{g}$, where $\mathbf{c}$ is a central element, and where the Lie bracket is given by

$$[xt^n, yt^m] = [x, y]t^{n+m} + n\delta_{n,-m}(x, y)\mathbf{c}.$$

The algebra $\widehat{\mathfrak{g}}$ is naturally equipped with a $\widehat{Q} = Q \oplus \mathbb{Z}\delta$-grading, where

$$\widehat{\mathfrak{g}}[\alpha + l\delta] = \mathfrak{g}[\alpha]t^l, \qquad \widehat{\mathfrak{g}}[0] = \mathfrak{h} \oplus \mathbb{C}c, \qquad \widehat{\mathfrak{g}}[l\delta] = \mathfrak{h}t^l.$$

We extend the Cartan form to $\widehat{Q}$ by setting $(\delta, \alpha) = 0$ for all $\alpha \in \widehat{Q}$. The root system of $\widehat{\mathfrak{g}}$ is $\widehat{\Delta} = \mathbb{Z}^*\delta \cup \{\Delta + \mathbb{Z}\delta\}$. We say that a root $\alpha \in \widehat{\Delta}$ is *real* if $(\alpha, \alpha) = 2$ and *imaginary* if $(\alpha, \alpha) \leq 0$.

(a)

Lefg = $Q[I, t^1]$ j ©Cc be the affinization of g, where c is a central element, and where the Lie bracket is given by

$$[xt^n, yt^m] = (X_{>}y]t^{n+m} + n\delta_{n,-m}(x, y)C.$$

The algebra $\hat{g}$ is naturally equipped with a $Q = Q \odot Z5$-grading, where

$$f[a + IS] = Q[a]t', \qquad f[0] = fj \odot Cc, \qquad ?[/\$] = Ij^{*'}.$$

We extend the Cartan form to Q by setting (5, a) = 0 for all a e Q. The root system off is A = Z*5 U {A + Z<5}. We say that a root « 6 A is rea/ if $(a,$ a) = 2 and *imaginary* if (a, a) < 0.

(b)

Fig. 11.1. OCR output of scientific documents: an example (a) image (b) recognition results.

Mathematical expressions typically appear in documents, either as (a) displayed (isolated) expressions or (b) expressions embedded into (i.e. mixed with) the text lines. As far as automatic identification of expressions is concerned, displayed and embedded expressions impose different level of complexities. The displayed ones are typed in separate lines and exhibit several image-level features that distinguish them from normal text lines. On the other hand, embedded expressions are generally small expression fragments, which are difficult to isolate from the text portion mixed with expressions. These issues have been discussed in Section 11.3.

Once expressions in a document are identified, recognition of them involves two major components namely (i) symbol recognition and (ii) structure interpretation. Symbol recognition is difficult because a large character set (roman letters, Arabic digits, Greek letters, Operator symbols, etc.) with a variety of typefaces (regular, italic, bold), and a large number of different font sizes may be used to generate the expressions. Moreover, certain symbols (e.g. *integration*, *summation*, *product*, *brackets*, etc.) are elastic in nature and have a wide range of possible scales.

Interpretation of structure is particularly another non-trivial problem due to the subtle use of space that often defines the relationship among symbols. For instance, unlike plain text (which is written linearly from left to right), symbols in an expression can be written above, below, and one nested inside another. Therefore, understanding of the spatial relationship among symbols is crucial to the interpretation of structure of an expression. This means that even if all the characters are correctly recognized, there still remains the non-trivial problem of interpreting the two-dimensional structure of an expression. Moreover, several symbols (e.g. *horizontal line*, *dot*, etc.) have multiple meanings depending on the context and such ambiguous role of symbols makes the interpretation task more difficult.

Work on an expression recognition system needs another problem to be addressed, namely quantitative evaluation of the system. Evaluation of such a system is non-trivial 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. Moreover, error in the symbol recognition stage affects the structure analysis result. This calls for an integrated evaluation mechanism for judging the performance of a system dealing with expression recognition.

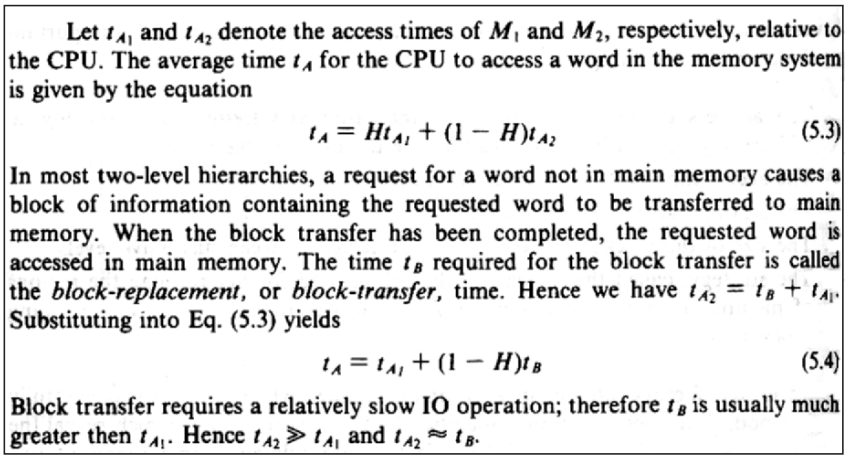
The rest of this chapter is organized as follows. Identification of expression zones in a document is discussed in Section 11.2. Section 11.3 presents several methods for recognition of expression symbols. Geometric interpretation of structure of expressions is presented in Section 11.4. Section 11.5 deals with the issues related to performance evaluation of an expression recognition system. Need for generation of standard benchmarked data for a set of scientific documents is outlined and availability of such data is mentioned in Section 11.5. Section 11.6 concludes with outlines of the aspects requiring further research attention in future.

11.2 Identification of Expressions in Document Images

Since embedded and displayed expressions are of different complexities, separate techniques have so far been proposed for identification of embedded and displayed expressions. Some reviews are provided in the next section (Figure 11.2).

11.2.1 Techniques for Identification of Expression Zones

Studies dealing with identification of expression zones are few in number. Most of the previous works assume that expressions are available in isolated form. Among the existing techniques, method proposed by Lee and Wang [29] labels text lines in a document as either *TEXT* (to denote normal text) or *EXP* (to denote displayed expression) based on two properties



Let t_{A_1} and t_{A_2} denote the access times of M_1 and M_2, respectively, relative to the CPU. The average time t_A for the CPU to access a word in the memory system is given by the equation

$$t_A = Ht_{A_1} + (1 - H)t_{A_2} \tag{5.3}$$

In most two-level hierarchies, a request for a word not in main memory causes a block of information containing the requested word to be transferred to main memory. When the block transfer has been completed, the requested word is accessed in main memory. The time t_B required for the block transfer is called the *block-replacement*, or *block-transfer*, time. Hence we have $t_{A_2} = t_B + t_{A_1}$. Substituting into Eq. (5.3) yields

$$t_A = t_{A_1} + (1 - H)t_B \tag{5.4}$$

Block transfer requires a relatively slow IO operation; therefore t_B is usually much greater then t_{A_1}. Hence $t_{A_2} \gg t_{A_1}$ and $t_{A_2} \approx t_B$.

Fig. 11.2. A sample page containing *embedded* as well as *displayed* expressions.

(i) isolated expressions are taller compared to the normal text lines and (ii) the line spaces above and below them are larger than those between text lines that contain no mathematical expressions.

The technique [29] for locating embedded expressions initially recognizes characters in a text line from left to right direction and then converts them to a stream of tokens. A token is decided to belong to an embedded expression according to some basic expression forms which considers presence of special mathematical symbols (e.g. horizontal line, summation, product, etc.), scripting or matrix structures. Symbols that are adjacent to the above tokens are heuristically attached to form an embedded expression.

Fateman [12] presented a three-pass algorithm that initially recognizes all connected components in a scanned document and separates them into two bags, math and text. The text bag contains all roman letters, italic numbers and the math bag includes punctuations, special symbols, italic letters, roman digits and other marks (e.g. horizontal lines, dots). Next, components in the math bag are grouped into zones according to their proximity. Symbols that are left ungrouped and appeared to be too far from other math symbols are moved to the text bag. Symbols in the text bag are similarly joined up into groups according to proximity. Text words (hopefully include words like "sin", etc.) that are relatively isolated from other text but within any previously identified math zone are moved to the math bag. Manual intervention is employed to review the segmentation result to correct errors, if any.

The method proposed by Inoue [24] isolates expressions contained in Japanese scientific document by assuming that the OCR recognizes Japanese characters with high confidence, whereas expression symbols are either rejected or recognized (rather misrecognized) with low confidence.

In another approach, Toumit [37] locate embedded expressions by finding special symbols like "=", "+", "<", ">", etc. and some specific context propagation from these symbols is done. For example, for parenthesis and brackets, symbols between them are checked; for horizontal bars, symbols above and below them are investigated, etc.

Later on, Kacem [26] proposed a two-pass scheme that does not put much emphasis on symbol recognition. Initially, expressions are separated from the text lines using a primary labelling which uses fuzzy logic based model to identify some mathematical symbols. Later on, a secondary labelling uses some heuristics to reinforce the results of the primary labelling and locates super- and subscripts inside the text. An evaluation strategy has been presented to judge the expression extraction technique and a success rate of about 93% has been reported on a combined test set of 300 displayed and embedded expressions. A similar technique is used in [34] to locate mathematical expressions in printed documents.

Recently, Chowdhury et al. [10] proposed a recognition-free approach that exploits the usual spatial distribution of the black pixels in math zones. Experimental results show that the method works well for segmenting displayed expression. In another recognition-free technique reported by Jin [25], embedded expressions are extracted based on the detection of two-dimensional structures. However, the authors of [10, 25] concluded that the extraction of embedded expressions is quite difficult without doing character recognition.

11.2.2 Ways to Improve Identification Results

On reviewing the existing methods, it is understood that identification of displayed expressions does not pose much difficulty compared to the task of locating embedded expressions. We also have experienced this while designing an expression recognition system [8]. There the module for detection of displayed expressions initially considers several image level features to suspect text lines containing isolated expressions. Some of these features are (i) wider white space above and below an expression line, (ii) height of the expression line, (iii) non-linear arrangement of symbols in an expressions, etc. Presence of one or more frequently occurring mathematical symbols has been checked to validate the identification results.

These features are formulated to give values in [0,1]. For example, feature representing white space property is defined as $f_{ws} = 1 - e^{(-\frac{r}{r_\mu})}$ where r denotes the average of the white space (measured in number pixel rows) above and below a text line and r_μ denotes the mean of the white space between two consecutive text lines. In case of the first line, only the line below it is considered to measure $\boldsymbol{r}$ (similarly, for the last text line its preceding line is considered).

Similarly, scatteredness of symbols inside a displayed expression is measured as, $f_{ms} = 1 - e^{-\sigma_y}$ where σ_y denotes the standard deviation among

the y-coordinates of the lower-most pixels of the symbols (i.e. connected components) of a text line. Feature related to height of a line containing displayed expression is formulated as $f_{mh} = 1 - e^{(-\frac{h}{h_\mu})}$ where h is the height of a text line in terms of pixel rows and $\boldsymbol{h}_\mu$ is the mean of all $\boldsymbol{h}$-**values**. The feature, $\boldsymbol{f}_{mo}$, keeps track of the occurrence of a few mathematical operator symbols that often appear in expressions. In our experiment [8], only 13 operators are considered for computation of $\boldsymbol{f}_{mo}$ that is defined as $f_{mo} = 1 - \exp\left\{-K \sum_{i=1}^{K} p_i\right\}$ where K denotes the number of operators identified in a text line and p_i denotes the probability of occurrence of the ith symbol.

Next, these features are computed on line level and then integrated (linearly/non-linearly) to produce a single score for individual line. Based on these scores, lines containing displayed expressions are selected.

However, these features are not sufficient (some are relevant at all) in case of embedded expressions. On the other hand, we did explore some aspects that seem interesting for this purpose. For example, if commercial OCRs are invoked to recognize pages containing expressions, OCR output shows interesting patterns:

- Sentences without expressions are recognized with almost no error.
- Also, high recognition accuracy is obtained for normal text words in sentences with expressions.
- Some of the expression symbols (e.g. roman letters, digits, symbols like "+", "−", "=", punctuation marks, etc.) are often recognized properly. However, for majority of these symbols, the OCRs, on recognition, associate suspicion marks with them to indicate that either these symbols in isolation (excepting characters like "a", etc.) do not form any valid word (e.g. isolated characters, characters with scripts, words like "sin", "log", etc.) or to reveal poor OCR confidence during their recognition (sometimes due to italic or bold characters).
- Other expression symbols (mostly Greek letters, majority of mathematical operators, special symbols, etc.) are either rejected (signalled by some special symbol) or mis-recognized with a suspicion mark.

Another interesting linguistic property is also observed. Let us divide sentences in scientific documents into two categories namely, sentences with and without expressions. Next, if word level n-grams are computed for these two categories then word n-gram based category profiles markedly differ from one another. This linguistic property can be used in spotting lines containing embedded expressions.

This technique has been explored in one of our studies [19]. It is experimented with about 4000 sentences appearing in scientific documents. More than 3000 sentences (number of sentences without any expressions: 2655 and number of sentences with embedded expressions: 870) were used to generate word n-gram based category profile. Later on, a test involving 877 sentences showed that on an average in 95% cases these category

profiles can distinguish a sentence with expression fragments from the one that does not contain any expression.

11.3 Recognition of Expression Symbols

Symbol recognition in mathematical expressions is difficult because there is a large character set (roman letters, Greek letters, operator symbols, etc.) with a variety of font styles (regular, bold, italic) and a range of font sizes (scripts, limit expressions, etc.). Certain symbols have a wide range of possible scales (e.g. brackets, parentheses, symbols like "$\int$", "$\sum$", "$\prod$", "$\cup$", etc.). Symbols also vary substantially in their shape characteristics. Therefore, recognition of mathematical symbols is considered as an important pattern recognition problem.

Review of existing studies shows that the studies dealing with recognition of symbols are a few. Most of the published papers have put emphasis on analysis of two-dimensional structures appearing in expressions. In several experiments, an error-free symbol recognition is assumed before formulating any method for structure analysis. In a controlled research environment, it is possible to bypass the symbol-recognition step and concentrate on structure analysis phase. However, design of a symbol-recognition module is essential to realize a complete expression recognition system.

11.3.1 Existing Methods for Symbol Recognition

The approaches proposed in previous studies on recognition of printed mathematical symbols can be broadly classified into two categories namely, (i) template matching and (ii) feature extraction and classification. Okamoto et al. [33] followed template matching approach where two sets of dictionaries for normal and script type symbols are maintained. Symbols are normalized to the predefined size prior to classification. Based on this method, an accuracy of 98.96% has been reported.

Fateman et al. [13, 14] proposed another template matching technique where a symbol template is represented by a vector of features. Bounding box of grey-level character image is divided into 5×5-rectangular grids and percentage of grey values in each grid is computed. The feature vector is made up of this set of grey-values along with two more data items, the height-to-width ratio and the absolute height in pixels of the bounding box. During classification of symbols, the authors used a Euclidean metric to define the distance between characters.

Lee and Lee [28] adopted a feature extraction based classification scheme where 13 features are utilized to represent each symbol. Next, a coarse classification algorithm is applied to reduce the number of candidates. For each input symbol, the character with the highest similarity is selected as the candidate symbol. The recognition accuracy reported in [28] is 84.80%.

The method presented by Lee and Wang [29] initially divides the symbol set into three classes based on the aspect ratio of bounding box of symbols. For recognizing a symbol within a class, the symbol image is divided into 4×4 non-uniform blocks and a four-dimensional direction feature vector is computed from each image block. It gives a 64-dimensional feature vector representation for each symbol. The authors achieved an accuracy of 96.18%. Ha et al. [22] also adopted a feature extraction based approach, but their classification is done through neural network.

Suzuki et al. [36] designed a recognition engine that tries to distinguish 564 symbol categories. The number of classes is so large because the authors considered several categories for a single character to tackle font and style variation. For instance, six categories have been considered for the character "B" to take care of its *regular*, *italic*, *bold*, *calligraphic*, etc. versions. A three-step coarse-to-fine classification strategy has been employed for recognition of symbols. The features like aspect ratio, crossing counts, as well as directional, peripheral and mesh features have been used for classification of symbols. Experiments conducted on a set of 476 scientific pages showed an accuracy of 95.18% for recognition of expression symbols.

11.3.2 Ways to Improve Symbol Recognition Accuracy

It is noted that symbol recognition accuracy is often lowered due to presence of touching characters in expressions. In many occasions, the adjacent characters in expressions touch each other in the scanned image and presence of such touching characters causes recognition errors. This is because the expression is typically segmented by connectivity analysis that considers touching characters as a single unit, which the recognition engine cannot properly tackle.

The main reasons of getting touching characters are (i) poor printing technology, (ii) inferior paper quality, (iii) photocopied documents, (iv) digitization errors, etc. Expressions contained in documents printed in ancient times impose a serious problem due to touching characters. Figure 11.3 shows a small set of touching characters found in printed expressions.

Lee and Wang [29] analysed the reasons behind the error in recognizing expression symbols and found that depending on document quality 12–28%

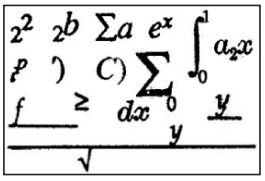


Fig. 11.3. Some touching characters found in mathematical expressions.

errors (out of all types of recognition errors) are due to touching characters. Suzuki et al. [36] also found nearly 2% touching character images in their database of 11,194 expressions. Therefore, addition of extra module for processing of touching characters is needed to improve the symbol recognition accuracy.

So far, only a few studies have addressed the issue of segmentation and recognition of touching characters appearing in expressions. Lee and Lee [28] proposed a dynamic programming algorithm where the segmentation is performed on a one-dimensional sequence of curve segments representing a connected component. The approach presented by Okamoto et al. [34] is based on the projection profiles of a given binary image of a pair of touching characters and minimal points of the blurred image obtained by applying the Gaussian kernel to the original image. This segmentation method is restricted for cases where only two characters touch each other.

Very recently, Nomura et al. [31] proposed an approach for detection as well as segmentation of touching characters in expressions. In their approach, touching characters are detected by looking at the deviation of the feature values (computed on an image of touching characters) from the standard feature values pre-computed for isolated characters. The segmentation is achieved by comparing a touching character image with a set of images synthesized from two single character images. Here also touching of only two characters is assumed. Since the number of ways by which two single characters touch each other is quite large, synthesis of all possible touching character images and comparison of a given image with all of these synthesized images is computationally not very much attractive. Moreover, such a scheme may fail to tackle variations in size, style and typefaces used to print expressions.

Segmentation of Touching Symbols: Considering the limitations of the previous approach we proposed a different technique described in one of our recent papers [17]. Since the number of touching characters is limited in an expression, no separate module for detection of touching characters is implemented in our system. Rather, any pattern rejected by the recognizer is initially suspected as touching character and segmentation is attempted for its recognition.

Several features contributing towards finding the cut-positions are identified. Selection of features is dictated by some observations like: (1) Though an image of touching characters mostly contains two characters, three or more characters touching each other are not rare. In our database [16], we found that among 2853 images of 6144 touching characters images consisting of 2, 3, and 4 characters are 2501 (87.66%), 266 (9.32%), 86 (3.02%), respectively. (2) Adjacent characters may touch each other in horizontal, vertical or diagonal directions. (3) If black runs (or crossing counts) are computed along the touching direction, a single run is, in general, encountered at the touching position. (4) The thickness of black blob at the

touching position is usually small compared to the thickness of other parts. (5) The character parts generate uncommon (quite a few in number) stroke patterns above and below the touching points.

These observations are captured through computation of a few features. For example, one feature i.e. $f_{ic} = \mathrm{c}^{-1}$ is formulated to obtain inverse crossing-count, where c is the crossing-count (number of white to black transitions) computed for an object (column, row, etc.). On the other hand, the vertical thickness of the black blob at the touching point is always small compared to the thickness of the other character parts, so another feature is formulated as $f_{mt} = 1 - e^{\frac{w_\mu}{t}}$, where t is the number of black pixels encountered in one scan and w_μ is the mean thickness of the character strokes.

Other two features concentrate on finding shape patterns above and below the touching blobs. Features are computed along four directions (vertical, horizontal and two diagonals namely, $+45°$ and $-45°$). In each direction, features are evaluated for each scan (e.g. each column for vertical direction; similarly, rows are considered as objects for horizontal direction, etc.). Next, the feature values in each scan are combined to give a scalar, which is used as a prediction for that scan to be a cut-candidate for separation of touching characters. The method does not assume anything about the number of characters that may be present in a touching character image.

Use of Multiple Classifier System: As mentioned earlier, the symbols appearing in mathematical expressions are quite large in number and show wide variety in shape, size and style. Symbols like dot, comma, colon, etc. are small in size and they have little shape signature. Symbols like equal to, plus, minus, fraction bar, vertical bar, greater than, less than, brackets, etc. are not much complex in shape and recognition of such strokes is not very difficult. On the other hand, symbols like roman and Greek letters, etc. involve relatively complex stroke patterns and recognition of these symbols needs some amount of extra effort.

Based on these observations, a multiple classifier system is proposed [18] for recognition of symbols. A group of four classifiers of different capabilities are arranged hierarchically in two levels as shown in Figure 11.4. The classifier used at the top level employs stroke-based classification technique to recognize symbols having high occurrence frequencies. Symbols not recognized at the first level are passed to the second level that employs a combination of three classifiers. The classifiers placed at this level make use of different feature descriptors namely, run-number or crossing-counts, density of black pixels and wavelet decomposition. Different combination techniques have been attempted to integrate the second level classifiers to achieve high recognition accuracy.

From this study, it is experienced that a single classifier is not capable enough to tackle the shape, size, style variations observed in expression symbols. Rather, a set of classifiers each working on subset of symbols

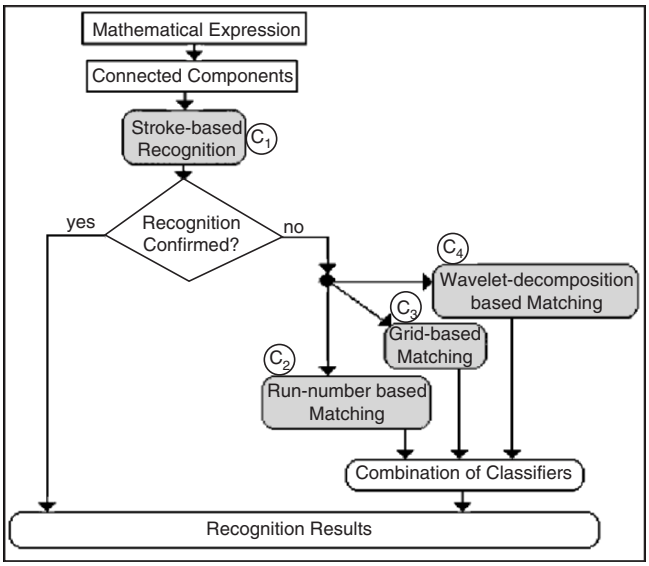


Fig. 11.4. Multiple classifier system.

can achieve better recognition accuracy provided they are combined in a suitable way. For example, in Figure 11.4 the classifiers show different performance in our experiment [15,18]. Classifier C_1 gives 98.3% accuracy, whereas, classifiers C_2, C_3 and C_4 give accuracies 88.1%, 85.8% and 90.5%, respectively, when tested in isolation. However, logistic regression method (i.e. logistic regression) combining C_2, C_3 and C_4 gives accuracy about 93.77%.

11.4 Interpretation of Expression Structure

The geometric structure of an expression can be significantly more complex than that of normal text lines. For example, plain text is written linearly from left to right, but mathematical symbols are written above, below and one inside another. The spatial relationship among symbols is crucial to the interpretation of the expression. This means that even if all the characters are correctly recognized, there still remains the non-trivial problem of interpreting the two-dimensional structure of an expression. Ambiguities arise in areas like (1) *The semantic role of symbols*: Several symbols (e.g. *horizontal line*, *dot*, etc.) have multiple meaning depending on the context; (2) *Relative symbol proximity and position*: Expression symbols use spatial relationship to indicate the logical relationship among them. For instance, structures like *superscript*, *subscript*, *implied multiplication*, *matrix*, etc. are indicated implicitly by the geometric layout of operands.

Several researchers have studied this problem and various approaches to solve this are reviewed next. However, these studies reveal that additional research is needed for automatic understanding of mathematical expressions to achieve acceptable accuracy.

11.4.1 Previous Methods on Interpretation of Expression Symbols

One of the earliest contributions in this area is by Anderson [1, 2]. His syntax-directed technique is essentially a top-down parsing approach based on a co-ordinate grammar. Though the partitioning strategy used there does not show satisfactory results in many cases, this work is regarded as a pioneering one. Afterwards, Chang [7] proposes an algorithm based on operator precedence and operator dominance, but did not clearly explain the algorithm as well as its performance in practical scenario.

Among other studies, some consider expressions in printed form whereas others assume handwritten input. In a few cases (e.g. [40]), structure analysis in both printed as well as handwritten data have been attempted by a single approach. In 1989, Chou [9] presented a stochastic context-free grammar to understand two-dimensional (2D) structure of expressions. A 2D probabilistic version of Cocke–Younger–Kasami parsing algorithm is proposed to find the most likely parse of the observed image. However, very little is discussed about the construction of the training set and the experimental results. Among other related approaches, Hull [23] computed probabilities by which two sub-expressions are in a particular relationship. The algorithm attempts to enumerate all possibilities by using an A-star search and prunes away the unlikely ones. Later on, Miller and Viola [30] tried to limit the number of potentially valid interpretations by decomposing the expressions into a sequence of compatible convex regions.

Twaakyondo and Okamoto [38] presented a technique that uses notational conventions in typing expressions. Structure of an expression is analysed by *projection-profile cutting* and a top-down strategy is used to analyse the horizontal and vertical relation between sub-expressions. To analyse nested structure such as subscripts, superscripts, etc., a bottom-up strategy is invoked that begin with the smaller symbols. The authors also provided an automatic approach [38] for evaluating their method. An accuracy of 98.04% is reported for recognition of 4701 elementary expression structures like *scripts*, *limit*, *fraction*, etc.

Lee and Lee [28] uses a procedure-oriented bottom-up approach to translate a 2D expression into a 1D character string. Initially, smaller symbol groups are formed around seven special mathematical symbols (i.e. "$\sum$", "π", *fraction*, etc.) which may deviate from the typographic centre of an expression. Next, symbol groups are ordered from left to right based on their centre y-coordinates. Matrices are handled separately [29]. The authors consider 105 expressions for training and 50 expressions for testing.

An error rate of about 2% is reported but the approach for computing the error rate is not presented.

Fateman et al. [3,14] described a *recursive decent* parser where an additional stage (called *linearization*) is used in between lexical analysis and the conventional parsing. In the *linearization* phase, adjacency relations among the tokens are detected. Several data-dependent heuristics are used. The experiments put emphasis on parsing of integrals. In another study [22], Ha et al. outlined a system that uses a recursive X–Y decomposition to understand the geometric layout of an expression.

Toumit et al. [37] assigned different priority levels to symbols in order to present a tree representation of the input expression. An alternative approach based on graph representation of an expression image is proposed by Grbavec and Blostein [21]. Nodes in the graph represent symbols or sub-expressions and edges represent relationship between the sub-expressions. Later on, Lavirotte and Pottier [27] used context-sensitive graph grammar technique which attempts to add context in the graph-rewriting rules for replacing one sub-graph by another.

Eto and Suzuki [11] proposed a concept of virtual link network to recognize expression structures. The proposed technique was tested using 123 expressions, of which 110 are properly recognized. The same was incorporated in [36] and experiment conducted on a larger dataset containing 12,493 expressions reported a structure recognition accuracy of 89.6%.

Garcia and Couasnon [20] proposed a generic method, DMOS (Description and MOdification of Segmentation) for recognition of musical scores, tables, forms, etc. The proposed method was tested using 60 formulae but details of the test results were not available. More recently, Zanibbi [40] described an algorithm consisting of multiple passes for (1) constructing a baseline structure tree describing the 2D arrangement of symbols, (2) grouping tokens comprised of multiple symbols and (3) arranging symbols in an operator tree which describes the order and scope of operations in the input expressions. Experiment using 73 expressions of UW-III database [35] showed a recognition (at expression level) accuracy of 38% (at most). However, the authors presented an in-depth analysis of the performance of their proposed method for structure analysis.

11.4.2 Further Research on Interpretation of Expression Structure

Studying the expression corpus [16], we noted a number of structural properties inherent in these expressions. Use of these properties make reconstruction algorithm more robust. A few such properties are mentioned below.

Bounding box of a symbol: For a symbol s, we define its bounding box $B(s)$ to be the smallest upright rectangle enclosing the symbol. We denote

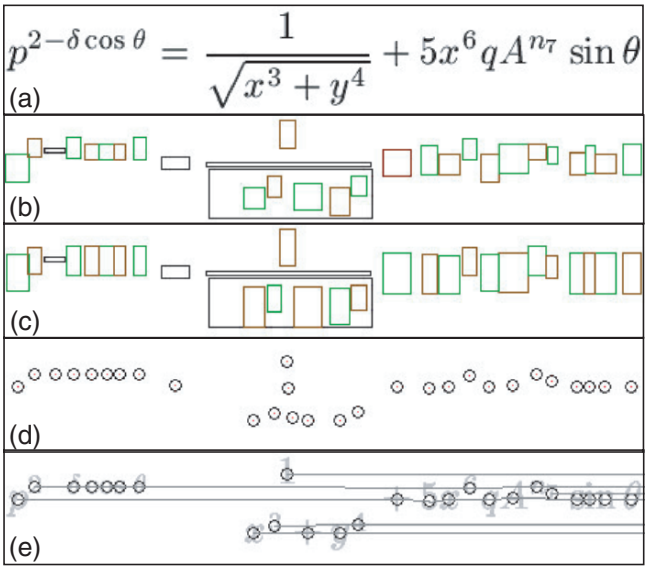


Fig. 11.5. Detection of horizontal lines and symbols' L-values: (a) expression image, (b) bounding boxes for expression symbols, (c) extended bounding boxes for symbols, (d) symbol centres and (e) horizontal lines (excluding the *HEES* symbols).

it by a four-tuple (*xl, xr, yt, yb*), where (*xl, yt*) and (*xr, yb*) represent the top-most left corner and the bottom-most right corners of B, respectively. Figure 11.5(b) shows bounding boxes for symbols of expression in (a). We define the height (*yh*) and width (*xw*) of a symbol by $yh = yb - yt + 1$ and $xw = xr - xl + 1$.

Enclosing Zone (EZ), Extended Bounding box (EB) and Centre (C) of a Symbol: Characters like roman letters, Greek symbols, etc. exhibit three zones in a text line. Let *enclosing zone* (*EZ*) refers to all the three zones and is **represented** by a pair of y-coordinates (*eyt*, *eyb*), *eyt* and *eyb* are being the y-coordinates of the top and bottom rows of the zone, respectively. If the symbols "A", "a" and "p" occur within the same *EZ*, then usually "A" and "a" will have *yb*-values very close to each other, but the *yt*-values will vary. Similarly, for symbols 'a' and 'p', *yt*-values are close but not the *yb*-values. We normalize *EZ* by assuming that an *EZ* starts at $yt = 0$ and ends at $yb = 1$. With respect to a normalized *EZ*, values of (*yt*, *yb*)-pair for all symbols are computed. However, there are some symbols (*elastic symbols* as discussed next), which are treated as exceptions for computing their *EZ*.

Once $EZ(s) = (eyt, eyb)$ is obtained for a symbol (s), its extended bounding box (*EB*) is computed as $EB = (xl, xr, eyt, eyb)$. For different classes of symbols, the rules for estimating *EB* from B (bounding box)

are formulated from the rules for normalizing *EZ*. Figure 11.5(c) shows the extended bounding boxes for symbols of expression in (a).

The centre of a symbol s is given by the y-centre of its $EB(s)$ and is computed as the arithmetic mean of the *eyt* and *eyb* values. Let EB (xl, xr, eyt, eyb) denote the extended bounding box for the symbol, s. Therefore, centre of s, $C(s)$ is given by $C(s) = (eyt + eyb)/2$. Figure 11.5(d) shows the centre positions for expression symbols in (a).

Elastic Symbols: There are some symbols that vary in size in different context. For example, horizontal lines may appear in expression in different lengths. Some other elastic symbols appear for *sum*, *integration*, *arrows*, *brackets*, etc. For all these symbols, $eyt = yt$, $eyb = yb$ and $C = (yt + yb)/2$. Among these symbols, certain symbols are horizontally elongated and denoted as *HEES* (horizontally elongated elastic symbol) symbols. An elastic symbol is identified as *HEES* if its aspect ratio (i.e. xw/yh) is more than a pre-defined threshold whose value is more than 1.

Level (L) of a Symbol: Computation of symbol L-values initially requires determination of horizontal lines on which symbols are arranged in an expression and identification of one of the lines as the dominant baseline [40] of the expression. The algorithm for finding symbol L-values initially sorts the symbols in ascending order of their centres (C-values). Then symbols having nearly the same C-values are grouped together to have the same L-values. This grouping starts with the first symbol and with $L = 0$. L-value increases for subsequent groups. Symbols labelled *HEES* are ignored for assigning any L-value.

Symbols of same L-values define a horizontal line (*HL*) in the expression. Let l be the number of such lines, $HL_1, HL_2, \ldots, HL_l$, where HL_i is defined by its member symbols having $L = i$. This l basically determines the geometric complexity (GC) of an expression. Centre of an *HL* ($C(HL)$) is computed as an average of its members' centre values. Figure 11.5(e) shows the *HL*s found for expressions in (a). Next, symbols detected as *HEES* are added to different *HL*s. An *HEES* is added to an *HL* by looking at their positional proximity.

Once all the symbols are arranged in different *HL*s, dominant baseline is selected as the *HL* that contains the left-most expression symbol. L-values of the symbols belong to this HL line is re-assigned to 0 and L-values of other symbols are changed with respect to this new HL_0. L-values of the symbols belonging to the *HL* just above this line (i.e. the new HL_0) are assigned to 1 and similarly, L-values of the symbols belonging to the *HL* just below the dominant baseline are assigned to -1. In this way, L-values increases above and decreases below the dominant baseline.

Reduction Ratio: Consider an expression A^{A}. The size (characterized by the height) of symbol reduces from the base level to the script (superscript or subscript) level. The ratio by which the height of a symbol (say, s) decreases from the base level to script level will be called the *reduction*

ratio of that symbol s, and denoted by $RR(s)$. In general, the *reduction ratio* of a symbol s is defined as, (height of s as super/subscript) $\div$ (height of s as base).

The RR-values of different symbols have been studied and the mean and standard deviation were found to be 0.60222 and 0.0237, respectively. Thus, the 3Σ-limit for RR-values is given by the interval (0.53109, 0.67335) and all the observed RR-values to lie in this interval. In other words, there is a reduction in height of a symbol by at least 32% (approx.) and at most 47% (approx.) at the script (super/sub) level with respect to its height at base level. This feature helps us to determine a *script* relation between a pair of symbols (or symbol groups) deferring in their L-values.

Space Between Symbols: The average space between two symbols, occurring in the same line, side by side, was measured as percentage of the height of their (normalized) enclosing zone (EZ). This measure is denoted by *sp* and shows a mean and variance of 0.05392 and 0.00050, respectively. This measure is used to identify function words (e.g. *sin*, *log*, *lim*, etc.) in expressions.

Expression Reconstruction: Let S be the set of pre-processed symbols (i.e. each symbol is tagged with its identity, bounding box info (B), enclosing zone (EZ), extended bounding box (EB), centre (C), level (L-value), etc.). The symbols S are sorted in ascending order of their *xl* values. Next, expression structure is interpreted and coded into a *TeX* string as follows.

Initially, the expression image is divided into n vertical stripes (called *vStripe*) based on the white space between horizontally adjacent symbols. Figure 11.6(a) shows the *vStripe*s for the image in Figure 11.5(a), where $n = 19$. Next, symbols under each *vStripe* are further segmented into horizontal stripes (called *hStripe*) based on the white space between vertically adjacent symbols. This vertical and horizontal segmentation continues until each stripe contains a single symbol or no further segmentation is possible. For the former case *TeX* equivalent of the symbol is returned and in the later case, further processing is invoked. For example, the *vStripe*, V_9 of Figure 11.6(a) is partitioned into three *hStripe*s (see Figure 11.6(b)), of which both $H_{9,1}$ and $H_{9,2}$ contain a single symbol. In case of $H_{9,3}$, no further segmentation is possible. In such cases, the largest (based on bounding box area) symbol (i.e. *square root* symbol) is separated (see Figure 11.6(c)) and further segmentation is invoked recursively on the rest of the symbols (see Figure 11.6(d)).

Once segmentation is over for all symbols, merging of symbols takes place. Merging of two/more symbols is guided by a context-free grammar similar to one in [2]. The statistics computed before are used to determine which grammar rule to be applied. For example, application of rule related to *script*s makes use of information like extended bounding box (EB), reduction ratio (RR), etc. Rule pertaining to *fraction* uses L-values of symbols.

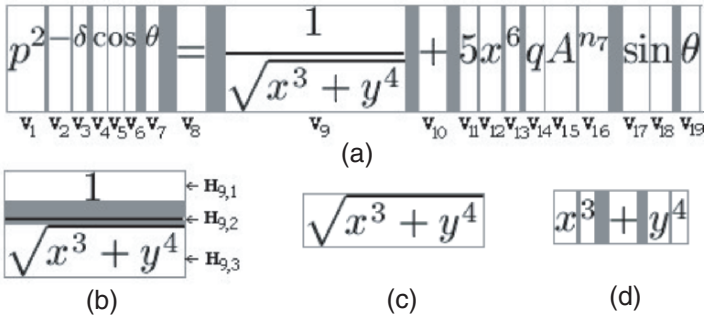


Fig. 11.6. Structural analysis of an expression image: vertical and horizontal segmentation.

Experiment with the database described in [16] shows that out of 5560 expressions 4348 are properly reconstructed giving an accuracy of 78.2%. Error analysis reveals that error increasing with the increase of geometric complexity of the expressions. However, this way of measuring accuracy is not judicious because placement error for a single symbol makes recognition of an expression incorrect. Therefore, a more reasonable performance measure is presented in the next section where a performance index is computed to provide a more meaningful evaluation of accuracy for interpretation of expression structures.

11.5 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. Errors in the symbol recognition stage affect the structure analysis results. This calls for an integrated evaluation mechanism for judging the performance of system dealing with expression recognition.

Chan and Yeung [6] 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. The proposed evaluation is based on manual effort. Later on, Okamoto et al. [32] have presented an automatic approach for evaluating their structure analysis method. They attempted to evaluate the performance by checking whether each typical structure such as scripts, limits, fractions, roots, and matrices, is recognized correctly. In their approach, expressions against which a system is evaluated are groundtruthed into MathML format. More recently, Zanibbi et al. [40] presented another

automatic way of evaluating the performance. In that approach, 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.

As the methods proposed in [6, 32] count only the number of properly recognized structures, an error in recognizing a simple structure gets the same weight as that of an error in a complex nested structure. On the other hand, the technique proposed in [40] presents more in-depth analysis of the recognition results, but does not provide a single figure of merit for overall performance evaluation.

11.5.1 Basic Requirements for Performance Evaluation

For evaluating the performance of an expression recognition system, at first, we need a corpus of scientific documents on which the system will be tested. Unavailability of a suitable corpus of expressions has prompted the researchers to define their own data set for testing their algorithms. As a result, replication of experiments and comparison of performance among different methods has become a difficult task.

Therefore, a database of expression images is needed that would facilitate research in automatic evaluation of expressions. The only relevant database available so far is the University of Washington English/Technical Document Database III (UW-III) [35]. However, the database is mainly constructed for general OCR (optical character recognition) research and contains 25 groundtruthed (into TeX) document pages containing about 100 expressions. Therefore, it does not seem to be a representative corpus for the respective research. Another drawback of this data set is that groundtruthing of expressions into TeX only does not support an in-depth analysis of recognition performance [6, 32, 40].

Recently, Uchida et al. [39] developed a database of a large collection of mathematical documents and analysed from several viewpoints for the development of practical OCR for mathematical and other scientific documents. The database contains 690,000 manually ground-truthed characters and available in public domain. The result of their analysis shows the difficulties of recognizing mathematical documents and at the same time, suggests several promising directions to overcome them.

Development of a Corpus of Scientific Documents: In one of our projects [15] we took up this problem and attempted to generate a database of printed scientific documents. The proposed database contains 400 document images containing 2459 displayed and 3101 embedded expression fragments. Both real (297 pages) and synthetic (generated by TeX or Microsoft Word) documents (103 pages) are present in the data set. Real documents are collected from (1) books of various branches of science,

(2) science journals, (3) proceedings of technical conferences, (4) question papers (College/University level examination), etc.

Synthetic documents are selected from sources that are available in Microsoft Word or TeX format. Several electronically available journals, conference proceedings, Internet sites related to various science subjects were considered for this purpose. A few pages were selected from the technical articles published by the members of our research unit. Several factors like (1) frequency of expressions in pages, (2) variations in typeset, (3) aging effect and other degradations, etc. influence the choice of materials.

Groundtruthing of Scientific Documents: The article in [16] describes the groundtruth format in details. Groundtruthing is done at the page level. However, it is assumed that page segmentation is done and only text parts (along with expressions) are labelled for Groundtruthing. Both embedded and displayed expressions are considered. For a given page, embedded expressions are truthed into *.emb* file whereas displayed expressions are stored in *.dis* file. Embedded expressions are recorded along with the sentence containing it. A sentence is said to have one or more embedded expressions if it would need the use of *math mode* had the sentence been prepared using *TeX*. On the other hand, displayed expressions are coded in isolation.

MathML format is followed to code the expressions. Some additional tags are used to encode bounding box information that pinpoints the expression zones in a page. Geometric complexity of an expression is coded and it is defined as the number of horizontal lines (on which expression symbols are arranged) found in the expression. For individual symbols, MathML tags are extended to include information like symbols' bounding box, type styles and *level* values. For example, if t is an identifier used in the expression, its MathML representation (i.e. $<$mi$>$ t $<$/mi$>$\$) is extended as follows:

$<$mi$>$
$<$level$>$ ... $<$/level$>$
$<$style$>$ ... $<$/style$>$
$<$truth $>$ ... $<$/truth$>$
$<$/mi$>$

The *level* indicates symbol L-values (discussed in Section 11.4.2). The *style* indicates the type style (n: normal, b: bold, i: italic, bi: bold-italic) of the symbol. The identity of the symbol is given within the $<$truth$>$ $<$/truth$>$ tag pairs. Some sample images and their corresponding groundtruth can be found at the following site: **http://www.isical.ac.in/∼utpal**.

Validation of the Truthed Data: While generating the truthed data, manual intervention was involved at several stages and therefore, prone to contain errors. On the other hand, for different purposes like performance

evaluation etc., it is important to have the truthed data free from errors. Therefore, we took some attempt in this direction.

Initially, some tools are developed to generate an upper level description of the page content from the ground-truthed data. This description is manually compared with the original page content. The elements that undergo checking are (1) expression itself (when MathML description is converted into two-dimensional layout), (2) expression positions, (3) their geometric complexity and (4) expression symbols are checked for their type style, level and identity. Bounding box for individual symbols are also checked for displayed expressions. Any error detected during this checking is corrected in the corresponding ground-truthed files.

We do consider the above validation check as the first level task and it is yet to finish for all the pages. Our plan is to check the truthed data (say, T) is to be manually checked and corrected by two different groups in isolation to produce two versions of the same truthed data namely, T_1 and T_2. Later on, T_1 and T_2 are to be compared by automatic file comparison utilities like cmp, diff, etc. Differences are to be marked and corrected to produce the final version of the truthed data.

11.5.2 Performance Evaluation Strategy

We formulate a performance evaluation strategy for expression recognition using the Groundtruthing format as discussed before. The recognition result for an expression (printed or handwritten) is compared with the groundtruth corresponding to that expression. If they do not match, then the result is not correct. This matching can be done by comparing two document object model[2] (DOM) trees, one generated from the groundtruth data for the expression and another generated from the recognition output for that expression. Let G_D be the DOM tree obtained from the ground-truthed data for an expression E and R_D be the DOM tree corresponds to the recognition result for E. A comparison between G_D and R_D detects the errors for recognition of E.

Since matching of two trees is itself a long-standing subject of research,[3] we at present do not explore much in this area (rather we treat this issue for our specific purpose as a future research problem). In our current approach, matching of two DOM trees is centred on the leaf-nodes only and parsing proceeds in a left to right order. At first, the left-most leaf node of G_D is picked up and corresponding leaf node in R_D is matched. Matching considers (1) identities (symbols) of the nodes and (2) the paths found from the leaf-nodes to the root-nodes in two trees. A mis-match in the first

[2] http://www.w3schools.com/dom/ and for further reference see http://www.w3.org/TR/MathML2/chapter8.html.

[3] We would like to refer to a recent article on this topic: Philip Bille, "Tree Edit Distance, Alignment Distance and Inclusion", available at "citeseer.ist.psu.edu/bille03tree.html".

case reports a symbol recognition error, whereas mis-match in the second case indicates symbol placement error.

Leaf-nodes generating mis-matches are marked in G_D as it represents the groundtruth. Manual intervention is required because the above DOM-matching approach identifies the symbols suffering from placement errors but at the same time it may mark certain other symbols which truly speaking do not suffer from placement errors. Actually, this matching method pinpoints the structures (i.e. group of symbols that impose a 2D structure like scripts, etc.) for which the arrangements of some constituent symbols are incorrect.

Errors originating from two sources (1) symbol recognition errors and (2) errors in structure interpretation. Symbol recognition errors is easily computed as

$$S_e = \frac{\text{No. of wrongly recognized symbols}}{\text{Total no. of symbols}} \tag{11.1}$$

For computation of structure recognition errors, the symbols with improper placement are identified. In our method, the erroneous arrangement of a symbol (s) is penalized by a factor $\frac{1}{|i|+1}$, where i is the level of the symbol, s.

It may be noted that in case of computing structure recognition error only spatial arrangements are important. For example, no structure recognition error is reported if x^m is recognized as x^{rn}. This is so because such symbol classification errors are accounted for by computing symbol recognition accuracy. In the foregoing example, structure recognition error is detected only if identification of *superscript* structure fails.

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

$$\gamma = 1 - \frac{S_e + \sum\limits_i O_i \times \frac{1}{|i|+1}}{S_t + \sum\limits_i R_i \times \frac{1}{|i|+1}} \tag{11.2}$$

Assuming a test set (T_s) contains Z expressions, γ_k is computed for all $\mathrm{k} = 1, 2, \ldots, N$ and to rate the overall system performance, an average γ_{avg} is computed as:

$$\gamma_{avg} = \frac{1}{Z} \sum_k \gamma_k \tag{11.3}$$

11.6 Conclusion and Future Research

Major steps needed for OCR of scientific documents have been discussed. The state of the art and different approaches available for implementation

of each step are outlined. Several schemes to achieve further improvement have been presented from our research experience. A new format for groundtruthing of scientific documents has been outlined and a performance index based on this groundtruthing format has been presented to evaluate an expression recognition system.

In spite of good amount of research effort as described in this chapter, commercial OCR systems are still not process expression with acceptable accuracy. Possible reason could be (1) most of the studies have dealt with isolated expressions and therefore, problems to deal with a whole page containing expressions have not been studied well and (2) non-availability of a representative dataset on which a proposed method can be tested to judge its confidence while working in real situation instead of being tested on limited dataset.

Recent research efforts are motivated by these needs. For example, the research group led by Prof. M. Suzuki has put a ground-truthed database along with their software in the Internet[4] so that the resources are available in the public domain. We too have made our groundtruthing method available[5] for public comments. These efforts will simulate interaction among the peer research groups.

Apart from these, we view that further research is needed in at least two areas (1) identification of expression zones in document images and (2) a unified method for performance evaluation of expression recognition systems. In fact, number of studies in these two areas is quite less. In Sections. 11.2 and 11.5, we have tried to discuss these issues. A robust approach that locates expressions in document images will enable the existing OCR systems to be enhanced to process scientific documents with better accuracy. On the other hand, a uniform performance evaluation method is very much needed to evaluate recognition results.

References

1. Anderson, R.H. (1968). Syntax-directed recognition of hand-printed two-dimensional mathematics. Doctoral dissertation. Department of Engineering and Applied Physics, Harvard University.
2. Anderson, R.H. (1977). Two-dimensional mathematical notations. In: K.S. Fu, (Ed.). *Syntactic Pattern Recognition Applications*. New York: Springer, pp. 147–177.
3. Berman, B.P. and Fateman, R.J. (1994). Optical character recognition for typeset mathematics. *ACM Proceedings of International Symposium on Symbolic and Algebraic Computation (ISSAC), Oxford, UK*, pp. 348–353.
4. Blostein, D. and Grbavec, A. (1997). Recognition of mathematical notation. In: H. Bunke and P.S.P. Wang (Eds.). *Handbook of Character Recognition and Document Image Analysis*. Singapore: World Scientific, pp. 557–582.

[4] http://www.inftyproject.org/en/index.html.
[5] http://www.isical.ac.in/∼utpal (for the link called "Resources").

5. Chan, K.-F. and Yeung, D.-Y. (2000). Mathematical expression recognition: a survey. *International Journal on Document Analysis and Recognition, 3*, pp. 3–15.
6. Chan, K.-F. and Yeung, D.-Y. (2001). Error detection, error correction and performance evaluation in on-line mathematical expression recognition. *Pattern Recognition, 34*, pp. 1671–1684.
7. Chang, S.-K. (1970). A method for the structural analysis of two-dimensional mathematical expressions. *Information Sciences, 2*, pp. 253–272.
8. Chaudhuri, B.B. and Garain, U. (2000). An approach for recognition and interpretation of mathematical expressions in printed document. *Pattern Analysis and Applications, 3*, pp. 120–131.
9. Chou, P.A. (1989). Recognition of equations using a two-dimensional stochastic context-free grammar. *Proceedings of the SPIE, Visual Communication and Image Processing IV, 1199*, pp. 852–863.
10. Chowdhury, S.P., Mandal, S., Das, A.K., and Chanda, B. (2003). Automated segmentation of math-zones from document images. *Proceedings of the Seventh International Conference Document Analysis and Recognition (ICDAR), Edinburgh, Scotland*, pp. 755–759.
11. Eto, Y. and Suzuki, M. (2001). Mathematical formula recognition using virtual link network. *Proceedings of the Sixth International Conference Document Analysis and Recognition (ICDAR), Seattle, USA*, pp. 762–767.
12. Fateman, R.J. (1999). How to find mathematics on a scanned page. *Proceedings of the SPIE, San Jose, California, USA, 3967*, pp. 98–109.
13. Fateman, R.J. and Tokuyasu, T. (1996). Progress in recognizing typeset mathematics. *Proceedings of the SPIE, San Jose, California, USA, 2660*, pp. 7–50.
14. Fateman, R.J., Tokuyasu, T., Berman, B.P., and Mitchell, N. (1996). Optical character recognition and parsing of typeset mathematics. *Journal of Visual Communication and Image Representation, 7*, pp. 2–15.
15. Garain, U. (2005). Recognition of printed and handwritten mathematical expressions. PhD thesis. Indian Statistical Institute.
16. Garain, U. and Chaudhuri, B.B. (2005). A corpus for OCR of printed mathematical expressions. *International Journal of Document Analysis and Recognition (IJDAR), 7(4)*, pp. 241–259.
17. Garain, U. and Chaudhuri, B.B. (2005). Segmentation of touching symbols for OCR of printed mathematical expressions: an approach based on multifactorial analysis. *Proceedings of the Eighth International Conference on Document Analysis and Recognition (ICDAR), Seoul, Korea, I*, pp. 177–181.
18. Garain, U., Chaudhuri, B.B., and Ghosh, R.P. (2004). A multiple classifier system for recognition of printed mathematical symbols. *The Seventeenth International Conference on Pattern Recognition (ICPR), Cambridge, UK*, pp. 380–383.
19. Garain, U., Chaudhuri, B.B., and Ray Chaudhuri, A. (2004). Identification of embedded mathematical expressions in scanned documents. *Seventeenth International Conference on Pattern Recognition (ICPR), Cambridge, UK*, pp. 384–387.
20. Garcia, P. and Couasnon, B. (2002). Using a generic document recognition method for mathematical formulae recognition. In: D. Blostein and Y.-B. Kwon (Eds.). *Proceedings of International Workshop on Graphics Recognition (GREC) LNCS*. Berlin, Heidelberg: Springer, 2390, pp. 236–244.

21. Grbavec, A. and Blostein, D. (1995). Mathematics recognition using graph rewriting. *Proceedings of the Third International Conference on Document Analysis and Recognition (ICDAR), Montreal, Canada*, pp. 417–421.
22. Ha, J., Haralick, R.M., and Phillips, I.T. (1995). Understanding mathematical expressions from document images. *Proceedings of the Third International Conference on Document Analysis and Recognition (ICDAR), Montreal, Canada*, pp. 956–959.
23. Hull, J.F. (1996). Recognition of mathematics using a two-dimensional trainable context-free grammar. Master's thesis. Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology.
24. Inoue, K., Miyazaki, R., and Suzuki, M. (1998). Optical recognition of printed mathematical documents. *Proceedings of Asian Technology Conference in Mathematics (ATCM)*. New York: Springer, pp. 280–289.
25. Jin, J., Han, X., and Wang, Q. (2003). Mathematical formulas extraction. *Proceedings of the Seventh International Conference Document Analysis and Recognition (ICDAR), Edinburgh, Scotland*, pp. 1138–1141.
26. Kacem, A., Belaid, A., Ben Ahmed, M. (2001). Automatic extraction of printed mathematical formulas using fuzzy logic and propagation of context. *International Journal on Document Analysis and Recognition (IJDAR), 4*, pp. 97–108.
27. Lavirotte, S. and Pottier, L. (1997). Optical formula recognition. *Proceedings of the Fourth International Conference on Document Analysis and Recognition (ICDAR), Ulm, Germany*, pp. 357–361.
28. Lee, H.J. and Lee, M.C. (1994). Understanding mathematical expressions using procedure-oriented transformation. *Pattern Recognition, 27*, pp. 447–457.
29. Lee, H.J. and Wang, J.-S. (1997). Design of a mathematical expression understanding system. *Pattern Recognition Letters, 18*, pp. 289–298.
30. Miller, E.G. and Viola, P.A. (1998). Ambiguity and constraint in mathematical expression recognition. *Proceedings of the National Conference of Artificial Intelligence. American Association of Artificial Intelligence, Madison, Wisconsin*, pp. 784–791.
31. Nomura, A., Michishita, K., Uchida, S., and Suzuki, M. (2003). Detection and segmentation of touching characters in mathematical expressions. *Proceedings of the Seventh International Conference Document Analysis and Recognition (ICDAR), Edinburgh, Scotland*, pp. 126–130.
32. Okamoto, M., Imai, H., and Takagi, K. (2001). Performance evaluation of a robust method for mathematical expression recognition. *Proceedings of the Sixth International Conference Document Analysis and Recognition (ICDAR), Seattle, USA*, pp. 121–128.
33. Okamoto, M. and Miyazawa, A. (1992). An experimental implementation of document recognition system for papers containing mathematical expressions. In: H.S. Baird, H. Bunke, and Yamamoto (Eds.). *Structured Document Image Analysis*. New York: Springer, pp. 36–53.
34. Okamoto, M., Sakaguchi, S., and Suzuki, T. (1998). Segmentation of touching characters in formulae. *Proceedings of the Third IAPR Workshop on Document Analysis Systems (DAS), Nagano, Japan*, pp. 283–289.
35. Phillips, I. (1998). Methodologies for using UW databases for OCR and image understanding systems. *Document Recognition V, Proceedings of the SPIE, San Jose, CA, USA, 3305*, pp. 112–127.

36. Suzuki, M., Tamari, F., and Fukuda, R. (2003). INFTY – an integrated OCR system for mathematical documents. In: S. Uchida and T. Kanahori (Eds.). *Proceedings of ACM Symposium on Document Engineering (DocEng), Grenoble, France*, pp. 95–104.
37. Toumit, J.-Y., Garcia-Salicetti, S., and Emptoz, H. (1999). A hierarchical and recursive model of mathematical expressions for automatic reading of mathematical documents. *Proceedings of the Fifth International Conference Document Analysis and Recognition (ICDAR), Bangalore, India*, pp. 119–122.
38. Twaakyondo, H.M. and Okamoto, M. (1995). Structure analysis and recognition of mathematical expressions. *Proceedings of the Third International Conference on Document Analysis and Recognition (ICDAR), Montreal, Canada*, pp. 430–437.
39. Uchida, S., Nomura, A., and Suzuki, M. (2005). Quantitative analysis of mathematical documents. *International Journal on Document Analysis and Recognition (IJDAR), 7(4)*, pp. 211–218.
40. Zanibbi, R., Blostein, D., and Cordy, J.R. (2002). Recognizing mathematical expressions using tree transformation. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 24*, pp. 1455–1467.