19th Computer Vision Winter Workshop
Zuzana Kúkelová and Jan Heller (eds.)
Křtiny, Czech Republic, February 3–5, 2014

Combining Structural and Statistical Approach to Online Recognition of Handwritten Mathematical Formulas

Jan Stria[1], Daniel Průša[1], and Václav Hlaváč[2]

[1]Center for Machine Perception, Department of Cybernetics, Faculty of Electrical Engineering, Czech Technical University in Prague, Czech Republic
{striajan,prusapa1}@cmp.felk.cvut.cz

[2]Department of Theoretical Computer Science and Mathematical Logic, Faculty of Mathematics and Physics, Charles University in Prague, Czech Republic
hlavac@cmp.felk.cvut.cz

Abstract *This paper introduces a novel method for online recognition of handwritten mathematical formulas. The method is based on the combination of a structural analysis with a statistical model specifying relations of individual symbols. A description of all recognition phases is given, focusing mostly on the structural analysis stage. The recognition process following a bottom-up manner is driven by a spatial grammar, which expresses mathematical formulas unambiguously. As the grammar describes the relative positions only roughly, a statistical model learned from a database of annotated formulas is employed to refine the description. Several experimental results are also reported.*

1 Introduction

Recognition of mathematical formulas has been widely studied in past years [1, 16]. There are two main motivations for dealing with it. Firstly, there is a need of interpreting mathematical expressions found in scanned documents including books and handwritten notes. In this case, the offline source data are formed by a set of images. On the other hand, we also may want to recognize formulas captured by a mouse, a tablet or a touch screen. Here we deal with online data in a form of strokes of points ordered in a time manner. The main motivation for online formulas recognition is the ability of entering mathematical expressions to a computer using handwriting. It is far more natural then utilizing an arbitrary existing format for math description (TEX, MathML). Once such an expression gets correctly interpreted, it can be pasted to an article, evaluated by a mathematical program or plotted as a graph. Recognition of online handwritten mathematical formulas is a subject of this paper.

Algorithms for formulas recognition are originally based on conventional methods for recognition of a simple text [9]. However, recognition of mathematical expressions is a more challenging task, as they usually comprise a rich spatial structure. For this reason, even the top performing recognizers are able to correctly recognize only approximately 2 out of 3 formulas [8]. So there is still much room for improvements.

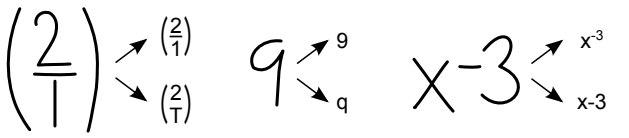

Figure 1: Ambiguities in segmentation (fraction vs. combinatorial number), symbol recognition (digit 9 vs. letter q) and structure (subtraction vs. power).

Online recognition of a handwritten mathematical formula can be usually separated into three relatively independent phases. In the first phase, the input strokes are segmented into groups potentially forming individual symbols. During the second phase, each symbol candidate is passed to a character recognizer. This is another source of errors and ambiguities, as it may be difficult to recognize the symbol properly. Finally in the third phase, spatial positions of the identified symbols are examined to reveal the whole expression. This is the last main source of ambiguities, as it may be difficult to decide what is the exact relation of two or more symbols. The sources of ambiguities are summarized in Figure 1.

There are two approaches of dealing with the presented ambiguities. The first approach finds the suboptimal solution in each of these phases [2, 4]. It identifies only one concrete segmentation of the strokes and interprets each group as one particular character. It is obvious that if this is done incorrectly then the formula cannot be successfully recognized. Thus we rather try to find all reasonable, possibly mutually conflicting, segmentations and we let each segment to be interpreted as various characters. The ambiguities are then solved during the third phase of recognition when the structure of a formula is analyzed [10, 11]. The general idea of this approach has been explicated as the structural construction paradigm in [12].

Quite common formalisms used to model the recursive character of mathematical formulas are constituted by various types of spatial grammars [3, 6]. Two-dimensional grammars are a quite powerful tool, however, they also bring a large challenge which is the computational complexity [5].

The planar placement of terminal units can easily cause that the worst-case parsing time grows exponentially (in contrast to the parsing of terminals in a linear sequence). The grammars thus require careful design and usage.

The main contribution of this paper is in four areas:

- We introduce a spatial grammar which describes mathematical formulas unambiguously. In contrast to previous approaches with productions which relate at most two nonterminals, our grammar contains production rules with many target terminal and nonterminal symbols. These more complex production rules enable more precise spatial analysis of symbols' positions.
- We present how to enhance capabilities of a spatial grammar by combining productions with statistical models refining the description of symbol alignments. The experiments prove that this leads to a significant increase of the correct rate of the structural analysis.
- We show that parsing of two-dimensional structures can be done efficiently. This is demonstrated by measuring time performance of the experiments.
- We further develop our recent work on structurally driven symbols segmentation. New methods for the creation of segmentation hypothesis are given.

The text is organized as follows. Section 2 emphasizes the most important components of our recognizer. Section 3 introduces the spatial grammar and describes how it is used to drive the structural analysis. Section 4 explains the statistical model of individual symbols relations. Section 5 provides experimental results achieved on databases of collected formulas. Finally, Section 6 summarizes presented ideas and proposes potential future improvements.

2 Overview

The input of the recognizer is formed by sequences of points in a plane, which were captured by a certain input device. The sequences are called strokes and they are ordered in a manner they were written. On the output, we provide the recognized formula in TEX and Presentation MathML format. The resulting formula is also rendered using a third-party library.

The recognizer was implemented as a web application. It can be seen in Figure 2. The application employs a HTML5 interface running in any modern web browser and a recognition engine implemented as a web service running at a server. The client part is capable of capturing the handwritten input, which is continuously being sent to the server. The received strokes are being recognized by the server in a background and their interpretation is sent back to the client. The interface is responsible for visualizing the recognized formula. The user is also allowed to erase previously written strokes or interactively correct misrecognized symbols which makes the process of recognition truly interactive. Moreover, correcting a few symbols usually leads to the successful recognition of the entered formula. More details about the web application can be found in [14].

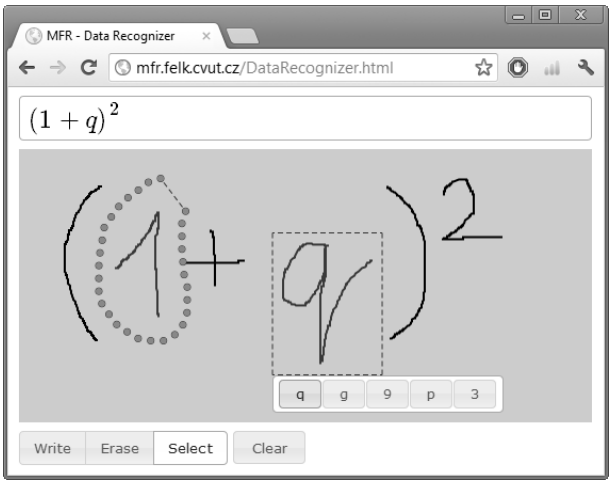

Figure 2: User interface of web application for mathematical formulas recognition. It provides a functionality of selecting and correcting a misrecognized symbol.

In the first phase of recognition, the input strokes are segmented into groups forming candidates for individual symbols. We allow one stroke to be a part of more symbol candidates. However, we allow one symbol to comprehend three strokes at most to avoid creation of too big groups. This constraint seems reasonable as perhaps no character requires more than three strokes when written in a standard way. The segmentation algorithm is based on finding two nearest neighbors of each stroke. Then each stroke forms one group by itself, two groups of two with each of its neighbors and one group of three with both of its neighbors.

The obtained groups of strokes are further checked to remove the non-perspective ones. The first heuristic algorithm relies on an empirically learned fact that strokes forming one symbol overlap horizontally or they are at least very close. The second heuristic rejects candidates whose strokes significantly intersect with strokes from other candidates. This rejects e.g. the horizontal and the vertical stroke in $+$ operator to form their own single-stroke groups.

The identified symbol candidates are then examined by two character recognizers. The first of them is the recognizer included in Microsoft Tablet PC platform. Its main advantage is the independence on the writing style. However, it is not able to handle special mathematical notation. Moreover, there is no way how to extend it. It grants up to ten characters for each candidate which are evaluated by a confidence as strong, intermediate or poor. These heuristics restrict the amount of symbol candidates significantly as shown in Figure 3.

The second recognizer is based on template matching. It searches for k-nearest classes of template symbols for each symbol candidate. One class corresponds to the unique character contained in the database of handwritten symbols, which was extracted from the database of annotated formulas. We store ten symbols per single class, i.e. character. To cover various writing styles of individual characters, all the extracted symbols belonging to the same class are hierarchically clustered and only one symbol is selected from each cluster. The distance used in the hierarchical clustering is

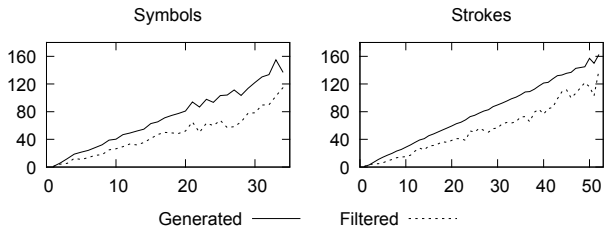

Figure 3: Average amounts of identified symbol candidates given the amount of individual symbols and strokes. Both amounts generated by the original segmentation algorithm and amounts after filtering are shown.

the same as the one used for finding the nearest classes. It is based on a so called elastic matching algorithm of the individual strokes where the optimal matching of strokes' points is found utilizing a dynamic programming approach [15].

The Tablet PC recognizer and our template matching recognizer are combined in the following way. Each symbol candidate is passed to the Tablet PC recognizer and the recognized characters are used as a filter for template matching. Moreover, characters unsupported by Tablet PC recognizer are also considered for template matching. This ensures that each symbol candidate is evaluated by the same algorithm and so there is no problem how to combine evaluations obtained from two different recognizers.

3 Structural analysis

Mathematical formulas can have very complex spatial structure. However, there are rigid constraints on a correctly formatted expression. These constraints can be naturally described using a 2D grammar. The grammar is defined in a separate text file as demonstrated on the following snippet:

```
Sum->[sum]|LowBound@B|UpBound@T|Expr@R
AddSub->AddSubOp|Term@L|Expr@R
AddSubOp->[+]
Expr->Sum
```

Each line corresponds to one production rule. There is a source nonterminal on the left side and a sequence of target nonterminal and terminal elements on the right side. The left side and the right side are separated by an arrow `->`. The elements on the right side are separated by vertical lines `|`. The terminals are enclosed in brackets and the nonterminals always start with a capital letter. In the first listed production, there is a nonterminal `Expr` representing an expression and a terminal `[sum]` representing a summation symbol. The terminals correspond to individual characters identified by the symbol recognizer. The nonterminals denote classes of terminals (e.g. `AddSubOp` or `Digit`) or logical parts of the formula (e.g. `Sum` or `Expr`) There is also a special axiom nonterminal `Formula` which represents a correct formula.

The productions are of two types. If there is only one target element on the right side, we speak about a 1-production (the last two lines of the snippet). By contrast, a n-production comprehends more than one target element on its right side (the first two lines). The 1-productions make it possible to maintain the grammar simple and readable. By contrast, the n-productions describe spatial relations of several target terminal and nonterminal elements, which form a more complex source nonterminal. The spatial relations of target elements are expressed relatively to the first target element, which is called a main element.

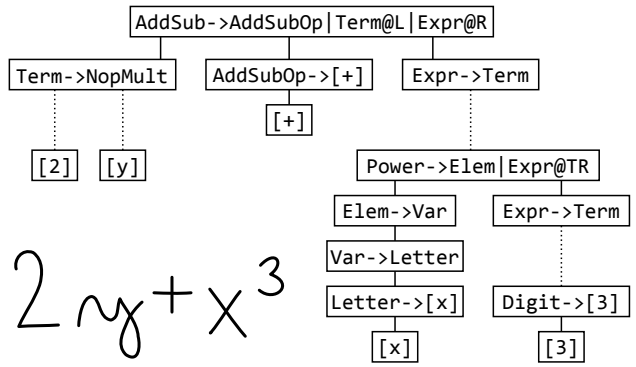

Figure 4: Handwritten formula interpreted as a derivation tree (with some nodes omitted for simplicity). Each node is assigned a terminal or a production. Nodes representing the same subset of strokes belong to the same segment.

Consider the first production describing a summation. The main element is a terminal `[sum]` denoting a summation symbol. The nonterminal `LowBound` expresses a lower bound of the summation. The suffix `@B` says that the lower bound is placed at the bottom of the main element. The `UpBound@T` expresses an upper bound of the summation placed at the top of the summation symbol. Totally, eight different relative positions are distinguished: left, right, top, bottom, top-left, top-right, bottom-right and inside.

The complete version of the grammar comprises approximately 200 terminals, 120 nonterminals and 370 productions. Approximately 100 of them are n-productions and 270 are 1-productions. There is also a reduced version of the grammar comprising approximately 100 terminals and 80 n-productions. The reduced grammar omits characters and constructs which are not supplied in the available databases of handwritten formulas. Both grammars describe the formulas unambiguously, i.e. for each formula there is only one possible description based on the grammar. The grammars were created manually by a human.

The proposed grammar drives the structural analysis of a handwritten formula, which consists of an iterated construction of derivation trees in a bottom-up manner. Figure 4 shows an example of such a derivation tree. The construction begins with an assignment of the appropriate terminals to symbols identified in the character recognition phase. Such symbols form leaf nodes of the derivation tree. All nodes sharing the same group of strokes are joined to a so called segment, i.e. derivation nodes belonging to one segment express various interpretations of that segment's strokes. Subsequently, 1-productions are repeatedly applied on each node, deriving new nodes having the original ones as their children. Consider that a node derived by 1-production belongs to the same segment as the original node.

The n-production nodes are derived by joining several existing nodes according to a certain n-production. The re-

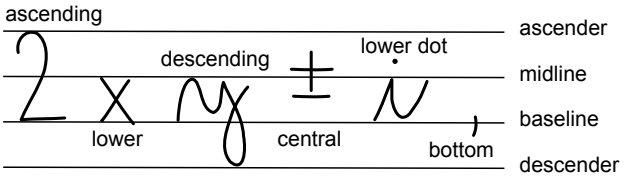

Figure 5: Various types of terminals based on their vertical positioning on four horizontal lines related to a writing line.

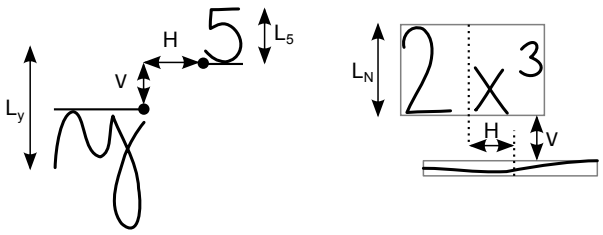

Figure 6: Relative distances and size of bind symbols. For a power y^5, the position mid segment of y and base segment of 5 are considered to determine their horizontal distance H and vertical distance V. The distance are normalized by the estimated line height L_y. The relative size is computed from L_5 and L_y. For the numerator $2x^3$ and the fraction line, their bounding rectangles are considered.

sulting node belongs to a segment created by joining segments of the original nodes. The original nodes become children of the derived node. There are several constraints on the original segments. First of all, their sets of strokes have to be pairwise disjoint because each stroke can participate only in a unique symbol of one derivation tree. Other constraints serve for reducing the amount of derived nodes. Based on the bounding rectangles of joint nodes and their relative positions, various polygons are constructed. These polygons are checked for an intersection with bounding rectangles of strokes not participating in the nodes. If the relative size of the intersection area is above a predefined threshold, the derivation is rejected.

The parsing algorithm is driven by the amount of strokes in nodes derived by n-productions. The nodes comprising k strokes are constructed in the k-th step. This systematic approach ensures derivation of more complex structures from less complex ones. It begins with derivation of nodes comprising two strokes and ends with nodes comprising all the input strokes. It should be obvious that many trees are constructed in parallel sharing common subtrees. Each derivation node is assigned a value. This value is equal to the belief obtained by the character recognizer for terminal nodes which correspond to individual symbols. For nodes derived by n-productions, the value is based on values of the child nodes and on their relative positions as described in Section 4. The derivation tree having the highest evaluated root node is chosen at the end of the parsing algorithm.

4 Statistical model

The productions describe relative positions of target terminal and nonterminal elements with respect to the main target element, as described in Section 3. However, the relative positions are only expressed as several discrete values (left, right, top, bottom, top-left, top-right, bottom-right and inside). To be able to define the spatial relations more precisely, we employ a statistical model of the expected positions. The model is extracted from the training database of 1500 handwritten formulas. Because some productions occur only sparsely in the database, it is not possible to store separate distributions for each of them. It is rather necessary to categorize derivation nodes to classes based on the properties of the involved terminals.

We distinguish various types of terminals (i.e. characters) based on their positioning on a writing line which is determined with respect to four horizontal lines: descender, baseline, midline and ascender. It can be seen in Figure 5. The ascending terminals (e.g. digit 2) are written between the baseline and ascender, the lower terminals (e.g. letter x) between the baseline and midline, the descending terminals (e.g. letter y) between the descender and midline, etc. The type of the terminal determines the base, mid and ascender segment for each terminal node. Not all segments can be determined for each terminal (e.g. there is only the mid segment for the operator $+$ because it is not aligned with the baseline or the ascender line). The vertical position of each segment is aligned with the appropriate horizontal line. The horizontal coordinates of the mid segment are coordinates of its strokes bounding rectangle. The horizontal coordinates of the base and ascender segments are determined based on the vertically local minimum and maximum of the strokes horizontal coordinates. If two of the segments are defined for the node at least, it is possible to estimate its line height.

The proposed segments and bounding rectangles of symbols are utilized to measure their relative horizontal and vertical distance and their size ratio. All these values are measured relatively to the estimated line height to deal with variously sized handwriting styles. The employed productions as well as the types of the terminals determinate which segments or bounding rectangles are considered. E.g. for the node representing a power y^5, we utilize the mid segment of the node y and the base segment of the node 5 to compute the horizontal distance H and the vertical distance V, as shown in Figure 6. The relative distances are computed by normalizing H and V with the estimated line height L_y. The ratio of the estimated line heights L_5 and L_y gives the relative height of the exponent 5 with respect to the base y. Figure 6 also shows how the bounding rectangles of a numerator and a fraction line are used to evaluate their distance. Here the vertical distance V of bounding rectangles and the horizontal distance H of bounding rectangles' centers are normalized by the estimated line height of the numerator L_N.

While evaluating nodes in left, right, top-left, top-right and bottom-right relative positions, only the segments of the neighboring side terminal nodes are considered. E.g. while evaluating positions in the addition $2y + x^3$ from Figure 4, we utilize node y which is the rightmost terminal in $2y$, node $+$, and node x which is the leftmost terminal in x^3. We call every sidemost terminal subnode of its parent node as a bind node, because it is responsible for binding its parent node to the other nodes. E.g. in the addition $2y + x^3$, the node 2

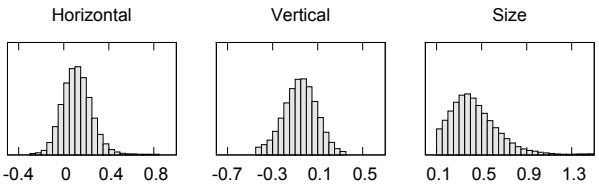

Figure 7: Examples of histograms of relative horizontal and vertical distance and relative size of a variable and its subscript. The histograms for various pairs of terminals and their relative positions form a statistical model.

is the left bind node and the node x is the right bind node because the addition cannot be bound via the exponent 3.

Totally, we distinguish 26 various categories, for which we evaluate the relative distances and the relative heights separately. The classification is based on the applied production and on the types of the terminal bind nodes. We store up to three distributions (relative horizontal and vertical distance and relative size) per each class as 1D histograms, as shown in Figure 7. It would be also possible to store the joint distribution for horizontal and vertical distance as 2D histogram. However, it would require more training data. Moreover, the relative horizontal and vertical distance are largely independent on each other, as the absolute value of the correlation coefficient is mostly around 0.1 and always under 0.2 for all 26 classes.

When evaluating a relative position of two nodes, we compute the appropriate horizontal and vertical distance and the height ratio. We also select the appropriate distributions among the aforementioned 26 classes. They give us beliefs for the computed relative values. These beliefs are combined linearly with evaluation of the individual child nodes to get the evaluation of the derived node. The derived node is constructed only if its value is over a predefined threshold. This strategy prevents deriving many poorly evaluated false nodes.

5 Experiments

To verify the proposed methods for online math recognition, we have tested them on databases of handwritten formulas containing rather challenging samples. The first of them is our own database called MfrDB which is publicly available [13]. It contains 2000 formulas, comprising 185 various expressions obtained from 232 users. The dataset was randomly split into 1500 training and 500 testing samples. The formulas were obtained using a web interface and then they were annotated semi-automatically. The semantics of formulas are described in an extended MathML format.

The second database is the one used in CROHME competitions. The CROHME 2012 [7] database contains 1366 training and 488 testing samples. Both training and testing samples are split into three parts based on the complexity of expressions (in a sense of average count of strokes per formula, number of various symbols included in formulas and complexity of a grammar describing formulas in a dataset). The latest CROHME 2013 [8] database contains 8836 training and 671 testing samples. They are even more complex than those included in CROHME 2012 database. Both training and testing samples are publicly available for CROHME 2012, however only training samples are available for CROHME 2013. That is why we do not use CROHME 2013 for a detailed evaluation.

Results	CROHME 2012			MfrDB
	Part I	Part II	Part III	
Symbol	67.6	67.8	65.6	67.3
Structure	**74.0**	**58.8**	**54.9**	**65.4**
Formula	9.7	5.8	3.9	5.2

Table 1: Recognition results on testing samples of CROHME 2012 and MfrDB databases. First row presents accuracy of isolated symbol classification. Second row gives independent performance of structural analysis given perfect segmentation and symbol classification. The third row shows rates of completely recognized formulas.

System	2012	2013		
	0 err.	0 err.	1 err.	3 err.
MFR (our system)	**3.9**	**2.7**	9.7	20.7
Tokyo Univ.	24.0	20.0	34.1	42.9
Univ. of São Paulo	6.4	9.4	18.5	27.3
Univ. of Valencia	24.9	**23.4**	37.9	47.8
Tech. Inst. Rochester	12.6	14.3	24.7	36.2
Sabanaci Univ.	11.9	8.4	19.1	26.2
Vision Objects	**70.2**	**60.4**	80.3	86.1
Univ. of Nantes	**32.9**	18.3	32.0	42.9

Table 2: Results of CROHME 2013 competition for all 8 participants. For the previous 2012 dataset, only the overall accuracy expressing rates of completely correctly recognized formulas is stated. For the latest 2013 dataset, also rates of formulas recognized with at most 1 and 3 errors are stated.

To train our system, we have used 1500 training samples from MfrDB and 8836 training samples from CROHME 2013. The experiments were performed on three CROHME 2012 testing datasets comprising 108, 301 and 488 formulas (part II comprises all formulas from part I and part III comprises both parts I and II), as well as on 500 testing formulas from MfrDB.

The recognition results are shown in Table 1. The first row shows rates of correctly recognized symbols in all three CROHME 2012 testing datasets and in MfrDB testing dataset. The symbols were extracted from formulas contained in the datasets and classified independently on their context in formulas. The symbol is considered to be correctly recognized if the symbol recognizer is able to assign a correct label to its strokes. The third row of Table 1 presents overall rates of fully recognized formulas. The formula is considered to be fully correctly recognized if the resulting derivation tree corresponds to the annotation and all the strokes are correctly assigned to their symbols. We have also tested the structural analyzer independently on the

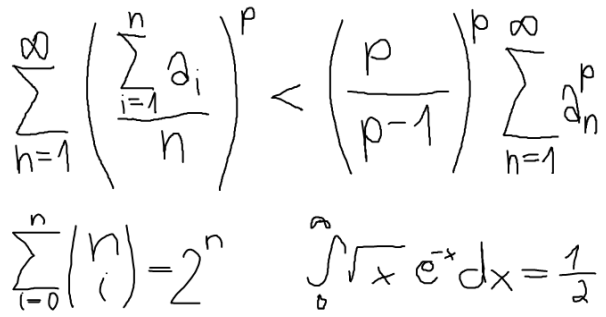

Figure 8: Examples of formulas which were correctly parsed by our structural analyzer.

Figure 9: Example of the same handwritten formula written by two various users. The left one was parsed successfully as $e^{i\pi} + 1 = 0$. The right one was misinterpreted as $e^{i^{\pi}+1} = 0$.

strokes segmentation and symbol recognition. To accomplish that we have assigned correct labels to all actual symbols. It simulates a case of having perfectly identified and recognized symbols. The results of structural analysis are presented in the second row of Table 1. Some of the correctly and incorrectly parsed formulas are shown in Figure 8 and Figure 9.

Table 2 presents results from the most recent CROHME 2013 competition. Totally 8 systems including ours were evaluated on part III of CROHME 2012 database and on CROHME 2013 database. Ratios of completely recognized formulas were published for both databases. For the latter also ratios of formulas recognized with 1 and 3 errors were published [8]. There were two winners of CROHME 2013 competition: non-commercial system developed at University of Valencia and commercial software by Vision Objects company. Both of them outperform our system by an order of a magnitude.

The main reason for a poor overall recognition rate of our system is the symbol classifier. It is only able to recognize approximately $2/3$ of all symbols compared to 85–98 % accomplished by other systems [7]. Unfortunately, one unrecognized symbol makes it impossible to recognize the whole formula correctly.

On the other hand, the structural analysis gives very promising results. We believe that an improvement of the symbol recognizer will make the overall results more satisfying. By that time, the user of our application can interactively select the misrecognized symbols and correct them manually while using the web application [14].

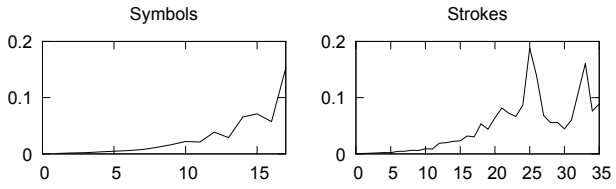

Figure 10: Performance of the structural analyzer expressed as an average time (in seconds) spent on analyzing formulas comprising the specified amount of symbols and strokes. The times on the vertical axes are measured in seconds.

Figure 10 shows an average parsing time spent on analyzing formulas comprising the specified amount of symbols. It was evaluated on a notebook with Intel Core i5 M430 2.27 GHz processor and 8 GB RAM. It can be seen that the results are provided in real time, almost immediately. This proves that although the parsing algorithm has theoretically an exponential complexity, it can be used in practice thanks to the described restrictions on the derived nodes.

6 Conclusion

We have introduced an algorithm for the structural analysis of mathematical formulas. It is driven by a rigid description of formulas employing an unambiguous spatial grammar. However, it also utilizes a statistical model of the expected mutual relative positions of individual symbols. The introduced probability distributions give natural evaluations of derivation nodes. We have tested the proposed method on two independent databases and we obtained promising results.

In future, we would like to focus on the first two phases of online handwritten math recognition. The amount of the symbol candidates generated in the segmentation phase should be reduced, omitting the non-perspective candidates which obviously do not form a symbol. In the symbol recognition phase, amount of the recognized alternative characters per one symbol candidate should be also reduced. Besides, we would like to improve the character recognition itself, perhaps by replacing the template matching algorithm by a better performing one.

Acknowledgement

The first author was supported by the EC project FP7-288553 CloPeMa. The first and second author were supported by the Grant Agency of the Czech Technical University under the project SGS13/205/OHK3/3T/13. The third author was supported by the Grant Agency of the Czech Republic under the project P202/10/1333.

References

[1] Kam-Fai Chan and Dit-Yan Yeung. Mathematical expression recognition: A survey. *International Journal on Document Analysis and Recognition*, 3(1):3–15, 2000.

[2] T. Kanahori, K. Tabata, W. Cong, F. Tamari, and M. Suzuki. On-line recognition of mathematical expressions using automatic rewriting method. In *Proc. 3rd International Conference on Advances in Multimodal Interfaces (ICMI 2000)*, pages 394–401, Beijing, China, 2000.

[3] Stéphane Lavirotte and Loïc Pottier. Mathematical formula recognition using graph grammar. In *Proc. SPIE on Document Recognition*, pages 44–52, San Jose, California, USA, 1998.

[4] Chuanjun Li, Timothy S. Miller, Robert C. Zeleznik, and Joseph J. LaViola. Online recognition of handwritten mathematical expressions with support for matrices. In *Proc. 19th International Conference on Pattern Recognition (ICPR 2008)*, pages 1–4, Tampa, Florida, USA, 2008.

[5] Percy Liang, Mukund Narasimhan, Michael Shilman, and Paul Viola. Efficient geometric algorithms for parsing in two dimensions. In *Proc. 8th International Conference on Document Analysis and Recognition (ICDAR 2005)*, pages 1172–1177, Seoul, Korea, 2005.

[6] Erik G. Miller and Paul A. Viola. Ambiguity and constraint in mathematical expression recognition. In *Proc. 15th National Conference on Artificial Intelligence (AAAI 1998)*, pages 784–791, Madison, Wisconsin, USA, 1998.

[7] Harold Mouchère, Christian Viard-Gaudin, Dae Hwan Kim, Jin Hyung Kim, and Utpal Garain. ICFHR 2012 — competition on recognition of online mathematical expressions (CROHME2012). In *Proc. 13th International Conference on Frontiers in Handwriting Recognition (ICFHR 2012)*, pages 1497–1500, Bari, Italy, 2012.

[8] Harold Mouchère, Christian Viard-Gaudin, Richard Zanibbi, Utpal Garain, Dae Hwan Kim, and Jin Hyung Kim. ICDAR 2013 CROHME: Third international competition on recognition of online handwritten mathematical expressions. In *Proc. 12th International Conference on Document Analysis and Recognition (ICDAR 2013)*, pages 1460–1464, Washington, D.C., USA, 2013.

[9] Réjean Plamondon and Sargur N. Srihari. On-line and off-line handwriting recognition: A comprehensive survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):63–84, 2000.

[10] Daniel Průša and Václav Hlaváč. Mathematical formulae recognition using 2D grammars. In *Proc. 9th International Conference on Document Analysis and Recognition (ICDAR 2007)*, pages 849–853, Curitiba, Brazil, 2007.

[11] Daniel Průša and Václav Hlaváč. Structural construction for on-line mathematical formulae recognition. In *Proc. 13th Iberoamerican Congress on Pattern Recognition (CIARP 2008)*, pages 317–324, Havana, Cuba, 2008.

[12] M. I. Schlesinger and Václav Hlaváč. *Ten Lectures on Statistical and Structural Pattern Recognition*. Springer, Berlin, Germany, 2012.

[13] Jan Stria, Martin Bresler, Daniel Průša, and Václav Hlaváč. MfrDB: Database of annotated on-line mathematical formulae. In *Proc. 13th International Conference on Frontiers in Handwriting Recognition (ICFHR 2012)*, pages 540–545, Bari, Italy, 2012.

[14] Jan Stria and Daniel Průša. Web application for recognition of mathematical formulas. In *Proc. 11th Conference on Theory and Practice of Information Technologies (ITAT 2011)*, pages 47–54, Vrátna dolina, Slovak Republic, 2011.

[15] Seiichi Uchida and Hiroaki Sakoe. A survey of elastic matching techniques for handwritten character recognition. *IEICE Transactions on Information and Systems*, E88-D(8):1781–1790, 2005.

[16] Richard Zanibbi and Dorothea Blostein. Recognition and retrieval of mathematical expressions. *International Journal on Document Analysis and Recognition*, 15(4):331–357, 2012.