On-Line Handwritten Formula Recognition using Hidden Markov Models and Context Dependent Graph Grammars

Andreas Kosmala, Gerhard Rigoll, Stéphane Lavirotte, Loïc Pottier

On-Line Handwritten Formula Recognition using Hidden Markov Models and Context Dependent Graph Grammars

Andreas Kosmala & Gerhard Rigoll
Dept. of Computer Science
Faculty of Electrical Engineering
Mercator University Duisburg
D-47057 Duisburg, Germany
kosmala@fb9-ti.uni-duisburg.de
rigoll@fb9-ti.uni-duisburg.de

Stéphane Lavirotte & Loïc Pottier
Cafe and Lemme Teams
INRIA Sophia-Antipolis
2004, route des Lucioles BP 93
06560 Sophia Antipolis, France
Stephane.Lavirotte@sophia.inria.fr
Loic.Pottier@sophia.inria.fr

Abstract

This paper presents an approach for the recognition of on-line handwritten mathematical expressions. The Hidden Markov Model (HMM) based system makes use of simultaneous segmentation and recognition capabilities, avoiding a crucial segmentation during pre-processing. With the segmentation and recognition results, obtained from the HMM-recognizer, it is possible to analyze and interpret the spatial two-dimensional arrangement of the symbols. We use a graph grammar approach for the structure recognition, also used in off-line recognition process, resulting in a general tree-structure of the underlying input-expression. The resulting constructed tree can be translated to any desired syntax (for example: Lisp, LaTeX, OpenMath ...).

1. Introduction

Currently, most people who need to enter mathematical expressions for a computational treatment need to type formulae in an ASCII form or use an editor. But regarding today's word processors or mathematical software tools, the input interface still offers poor convenience for editing expressions. Obviously, handwritting could serve as a superior man machine interface for the input of formulae from lots of different domains, like physics, mathematics, chemistry.... Beside the mentioned desk-top applications, such a recognition engine could also be used as an interface for the upcoming keyboard-less "personal digital assistants" (PDA), in order to integrate complex calculator facilities. According to [1], the solution for the mathematical expressions recognition can be sub-divided into two main problems: segmentation and recognition, analysis and interpretation of the symbol's spatial-arrangement.

2. System overview

The system proposed here is capable of recognizing mathematical expressions with a set of 100 different characters or symbols in a writer dependent mode. Beside small and capital letters, and digits, the system contains several mathematical symbols ($+ - \cdot \; : \; / \; \overline{\quad} \; \hat{} \; \sqrt{\;} \sum \prod \int \; , \; ' \; = < > \leq \geq \rightarrow \leftrightarrow \approx ! \infty$), as well as some of the most frequently used Greek letters ($\alpha, \beta, \gamma, \lambda, \mu, \Delta, \pi, \omega, \epsilon, \tau, \phi$) and some parentheses ((,), [,], {, }). To model the spaces between the symbols, an additional 'space' model is introduced. For initialization, each of these symbols is written several times well separated on a sheet. For an embedded training, 100 common mathematical or physical formulae are collected as training set and an additional set of 30 formulae is used as test set, represented by 150000 feature vectors and 45000 feature vectors, respectively. Figure 1 gives an overview of the entire system. The processing levels as they are shown, will be described in the following sections.

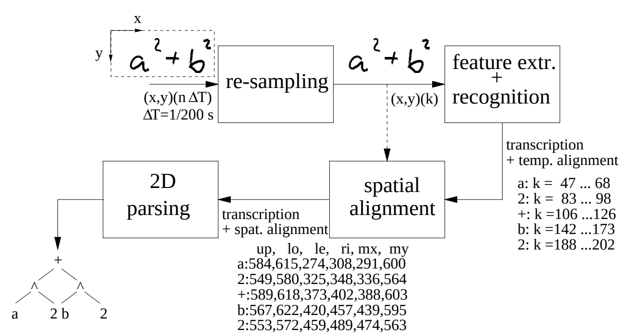

Figure 1. System overview

3. On-line character recognition

With the presented approach, it is not necessary to enter each single symbol well separated, neither spatially nor temporally. This continuous formula recognition can be considered as a kind of sentence recognition without previous word segmentation. The segmentation of the distinct words is realized within the HMM framework, applying a special 'space' HMM for the detection of word boundaries. Instead of attempting to identify each symbol separately, the efficient HMM-based decoding techniques allow a complete identification of the entire formula string. Another advantage of the HMM approach for this application is the possibility to incorporate high level syntactic constraints (e.g. grammars) directly into the low level recognition process.

3.1. Pre-processing and low-level feature extraction

The raw data is captured with a constant sample rate of 200 Hz, resulting in a temporal vector sequence of the Cartesian coordinates of the pen position. The pre-processing step is a re-sampling of the captured pen trajectory with vectors of constant length [10]. Beside the data reduction effects of 1:2 up to 1:3, a further advantage of the re-sampling is, that the implicitly given writing speed, resulting from different distances between two samples and the constant sample rate, is eliminated. Writing speed is especially in the context of formula recognition supposed as a highly inconsistent feature, because obviously identical handwriting images can be produced with completely different writing speeds. For instance, this could happen if the writer holds on to think about his currently written formula. It should be stressed, that the re-sampling preserves the spatial information, i.e. the Cartesian pen coordinates.

On the recognition level, several features are extracted from the re-sampled vector sequence. The first type of features are the online features [10], which is the orientation α of the re-sampling vectors coded in sine and cosine as well as the sine and cosine of the differential angle $\Delta\alpha$ of two successive re-sampling vectors. The third online feature is the pen pressure, which is extracted as a binary feature and indicates if the pen is set down or lifted. The pen pressure is helpful to model word or symbol boundaries within an expression. The second type of feature is an off-line feature [3, 7], which is a sub-sampled bitmap, sliding along the pen-trajectory after re-sampling. In a subsequent step, each of these feature streams, except the binary pressure, is quantized with a vector quantizer (VQ). As vector quantizer, usually a k-means VQ is used or a MMI neural net (NN) VQ [9] with different codebook sizes for each feature, respectively. Subsequently, the resulting discrete multi-stream is presented to the HMMs.

3.2. HMM training and recognition

For the modeling of the symbols, discrete left to right HMMs without skips and with different numbers of states are used. With the features described above, discrete, i.e. hybrid NN-HMMs have shown to be superior compared to continuous HMMs [9]. One reason for this is the discrete nature of the pen-pressure (pen up or pen down). Furthermore, the α and $\Delta\alpha$-features are due to the constant re-sampling vector length distributed on the unit circle, which enables an efficient vector quantization, while Gaussian pdfs as they are commonly used in continuous HMM systems can not be sufficiently mapped to this kind of distribution. HMMs with 12 states are used for capital letters and larger mathematical operators, like sum or product. Small letters and smaller operators are modeled with HMMs with 8 states, while very short symbols (.,) are modeled with 3 state HMMs.

For recognition purposes, a synchronous Viterbi decoding is used [8]. By means of the Viterbi algorithm, it is possible to determine the most likely state sequence $\mathbf{q}^\star$ of a set of HMMs λ for a given sequence of frames $\mathbf{O}$.

$$P(\mathbf{O}, \mathbf{q}^\star|\lambda) = \max_{\mathbf{q}} P(\mathbf{O}, \mathbf{q}|\lambda) \tag{1}$$

This can be very effectively exploited for formula recognition by analyzing the resulting alignment of each feature frame to its best matching HMM state, with the result, that the indexes of start- and end-frames of the recognized symbols within an expression can be taken directly from the decoding or recognition process. The achieved recognition with the simultaneous segmentation of the symbols is an important information for further extraction of geometrical features for the structure analysis.

The initialization of the HMMs is realized with the Viterbi training using the separate collected, isolated samples of the symbols. HMM parameter optimization is carried out by an iterative application of the Viterbi algorithm in order to find an optimal state sequence, and a re-estimation path of the pdfs.

For the embedded training of the complete formulae, the Forward Backward algorithm is used. Again, in an iterative way, optimized HMM parameters $\hat{\lambda}$ can be found by a maximization of the *Kullback-Leibler distance* $Q(\lambda, \hat{\lambda})$:

$$\max_{\hat{\lambda}} Q(\lambda, \hat{\lambda}) = \max_{\hat{\lambda}} \sum_{\mathbf{q}} P(\mathbf{q}|\mathbf{O}, \lambda) \log P(\mathbf{O}, \mathbf{q}|\hat{\lambda}) \tag{2}$$

The re-estimation formulae for the HMM parameter set can be derived directly from $Q(\lambda, \hat{\lambda})$ [8].

4. Structure analysis

As described before, the result from the Viterbi decoder is not only the transcription (the sequence of recognized words and symbols) but also the start- and end-frames of each symbol. The following sub-sections describe, how this information can be used for further processing.

4.1. Spatial alignment

The temporal alignment of frames together with the re-sampled vector-data allows the extraction of geometrical features of the symbols or a spatial alignment. These geometrical features are the center of a recognized symbol and the size and position of its bounding box. As shown in Fig. 2, the bounding box of a word is extracted by stepping through the re-sampled vector sequence starting at the start-frame number of the current word until the stop-frame number is reached, and searching for minima and maxima in x- and y-direction in the determined part of the sequence of Cartesian vectors. The spatial alignment for each symbol is characterized by the parameters of the bounding box up, lo, le, ri and denotes the upper, lower, left and right boundary of the symbol. The centers of the bounding boxes mx and my of each symbol are shown in the last two columns in figure 1 and figure 2.

Before starting analysis of spatial alignment, we introduce other computed information which will be attributes of recognized symbols. The introduction of this data will help to recognize formula's structure.

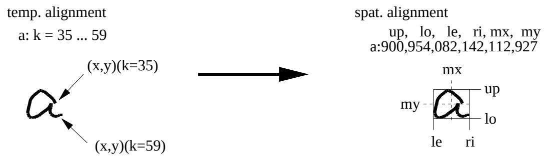

Figure 2. Spatial- from temporal alignment

4.2. Deduction of criterions

Next step is to deduce an approximative baseline depending on the recognized symbol and to compute a relative size for each recognized character in order to make some groups of symbols. This last process is not just depending on the bounding box size but is also relative to the recognized symbol. For example, if we compare “e” and “g” just considering the size of the bounding box, it will lead to the conclusion that “e” is smaller than “g”. So we ponder each symbol by a relative size depending on the manner to write the character. This can be done while collecting symbols during the training set. All this information will be used to determine the symbol potentially in a subscript or a superscript position.

4.3. Lexical analysis

Before starting the syntax recognition we give a lexical type to each recognized symbol. This method helps us to locate operators and avoid some parsing mistakes. It was first mentioned in [6] as an approach to help subscript and superscript recognition. Thus, recognition is based on spatial coordinates as well as local context depending on the lexical type. This system prevents misrecognition of relationship between two symbols. For example, it’s a good way to avoid the detection of “.” as a subscript of x in the example “x.y”, because a dot is not supposed to be a subscript for a letter. For a well formed printed document recognition, this problem of spatial arrangement is not as important as in a hand printed recognition because the writer does not follow the writing baseline exactly. This technic is helpful for subscript and superscript recognition but also to determinate other spatial relationship and especialy to avoid some of them depending on lexical rules.

5. Two dimensional parsing

The goal is now to generate a tree with recognized symbols as it is shown in figure 1.

5.1. Graph construction

With all these elements (symbols, lexical type, bounding box, approximative baseline and size) we have a lot of graphical information that we must translate in a more structured format to be usable and parsable. So, we introduce graph which provides a good formalism to describe structural manipulations of multi-dimensional data.

The graph building process is very important because too many links lead to ambiguities and with too few links one looses important data. This mechanism of data graph building using the spatial arrangement of symbols is described in [5]. It was tested with paper printed documents. We have modified and enhanced the system to improve links between symbols in graph and we have reduced constraints for links creation to suit handwriting style, which is more “random” than style of printed documents.

With the first small tests, we had good results for the building process. But with bigger formulae, the process was too slow, due to a $O(n^2)$ complexity. So we introduced an optimization for graph construction : we now make a subdivision of the space containing symbols, in order to avoid making some neighborhood tests with symbols which are obviously too far away from each other.

For a formula containing 116 symbols, without any optimization for the construction of the first data graph, computation time reaches 35 seconds on a Pentium 200. With a subdivision of space, it drops to less than 4 seconds. The observed speed-up (around 9 for this sample) is not always as high, but for smaller samples computation time is less than 1 second, with or without optimization. Figure 3 shows the evolution of computation time for the mentionned sample, regarding subdivision of the plane along the x and y axis.

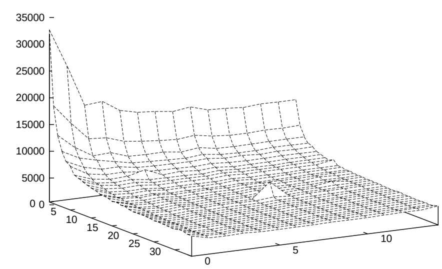

Figure 3. Time optimization for graph building

5.2. Graph parsing

In [4] we have defined a general class of graph grammars with contexts, where, after a theoretical study of their properties, we showed how to precisely and automatically solve ambiguities in these grammars.

A graph grammar acts on a graph as a rewriting system in a bottom-up manner, rewriting matched sub-graph into a single node containing the syntax-tree of the recognized expression. Improvements have to be done on computation time for graph rewriting. This motivates the next section.

5.3. Parsing optimization

The parsing algorithm is simple: we just iteratively apply the first rule which can be, and stop when no more apply.

A rule is given by a term $G - \{C_1, \ldots\} \rightarrow N$, where G is the sub-graph to be matched, $\{C_1, \ldots\}$ is the set of excluded contexts, and N is the produced node. Such a rule can be applied to a graph G' if:

- a substitution σ exists such that σG is a G' sub-graph,
- for each context C_i, there is no substitution τ such that τC_i is a sub-graph of G'', and $\tau = \sigma$ on the intersection of their supports.

When these conditions are satisfied, applying rule consists in replacing subgraph σG of G' by a single node σN.

During the parsing process, only local modifications are made to graph. So we very frequently test matching of subgraphs, particularly contexts, with few success: if a context occurs in several rules (which is the case in practice), and if it occurs in graph, then these rules will not apply, and we will test these occurrences at each step of parsing.

To avoid this unnecessary complexity, we exploit the locality of rewriting by maintaining, during the parsing of a graph G', two global lists:

- a list $\mathcal{G}$ of all pairs (G, σ) such that G is the graph pattern of a rule, and σG is a sub-graph of G',
- a list $\mathcal{C}$ of all pairs (C, τ) such that C is a context of a rule, and τC is a sub-graph of G'.

Searching a rule to apply can then be made by inspecting the lists $\mathcal{G}$ and $\mathcal{C}$ to find a pair (G, σ) such that no pair (C, τ) exists such that $\tau = \sigma$ on the intersection of their supports.

After applying a rule, we have to update the lists $\mathcal{G}$ and $\mathcal{C}$. But since only a small part of G' is modified, the two lists are also locally modified. With an adapted data structure (with pointers from the graph G' to lists $\mathcal{G}$ and $\mathcal{C}$, and pointers from $\mathcal{G}$ and $\mathcal{C}$ to G'), this can be achieved efficiently.

In fact, this process allows to replace a $O(n^2)$ by a $O(n)$ complexity for parsing process (where n is the number of edges of the graph to be parsed), which is optimal.

6. Conclusion

Presented approach leads to two major improvements:

1. Constraints concerning the writing (from left to right and top to bottom) as introduced in [2] can be relaxed.
2. The output is a general description of the handwritten expression in a tree format which can be translated to any other format (Lisp, LaTeX, OpenMath...).

Graph grammar was used to recognize mathematical expressions in printed documents as presented in [1] and [5]. With some more work, we have adapted this method to handwriting recognition of mathematical expressions. The same software is used in both case, just needing to specify the input type (it could be automated). Considering the method used in [2], some constraints were made on how the formula should be written (for left to right and from top to bottom). With this method, we do not have any assumption on how the formula should be written, so there is no constraint for writers and it is more adaptive to different writer style. Figure 4 gives two samples of recognized formulae.

The advantage of a graph grammar system is that it is adaptive to any kind of mathematical notation. To introduce a new notation, one just needs to write the right parsing rule. We are currently working on recognition of handwritten vectors or matrices notations.

In summary, graph rewriting is a good approach for mathematical expression recognition as well as for printed documents and for handwritten recognition. To our knowledge, it is the first experience in using graph grammars in handwritten mathematical formula recognition.

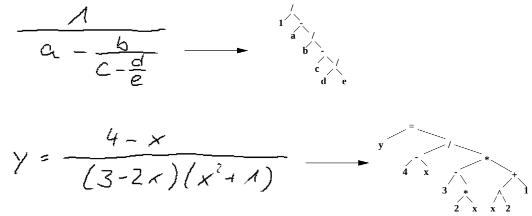

Figure 4. Recognition results

References

[1] D. Blostein and A. Grbavec. Recognition of Mathematical Notation. In H. Bunke and P. S. P. Wang, editors, *Handbook of Character Recognition and Document Image Analysis*, chapter 21, pages 557–582. World Scientific Publishing, 1997.

[2] A. Kosmala and G. Rigoll. Recognition of On-Line Handwritten Formulas. In *6th Int. Workshop on Frontiers in Handwriting Recognition (IWFHR)*, Taejon, Korea, 1998.

[3] A. Kosmala, J. Rottland, and G. Rigoll. An Investigation of the Use of Trigraphs for Large Vocabulary Cursive Handwriting Recognition. In *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3373–3376, Munich, 1997.

[4] S. Lavirotte and L. Pottier. Optical formula recognition. In *Proc. of 4th Int. Conf. on Document Analysis and Recognition*, volume 1, pages 357–361, Ulm, Aug. 1997. IEEE Computer Society Press.

[5] S. Lavirotte and L. Pottier. Mathematical formula recognition using graph grammar. In *Document Recognition V*, volume 3305, pages 44–52, San Jose, USA, Jan. 1998. SPIE - The Int. Society for Optical Engineering.

[6] H. Lee and J.Wang. Design of a mathematical expression recogntion system. In *Proc. of 3rd Int. Conf. on Document Analysis and Recognition*, pages 1084–1087, Montreal, Canada, Aug. 1995. IEEE Computer Society Press.

[7] S. Manke, M. Finke, and A. Waibel. Combining Bitmaps with Dynamic Writing Information for On-Line Handwriting Recognition. In *Proc. Int. Conf. on Pattern Recognition (ICPR)*, pages 596–598, Jerusalem, 1994.

[8] L. R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proc. of the IEEE*, 77(2):257–285, 1989.

[9] G. Rigoll and A. Kosmala. An Investigation of Context-Dependent and Hybrid Modeling Techniques for Very Large Vocabulary On-Line Cursive Handwriting Recognition. In *6th Int. Workshop on Frontiers in Handwriting Recognition (IWFHR)*, Taejon, Korea, 1998.

[10] G. Rigoll, A. Kosmala, J. Rottland, and C. Neukirchen. A Comparison between Continuous and Discrete Density Hidden Markov Models for Cursive Handwriting Recognition. In *Proc. Int. Conf. on Pattern Recognition (ICPR)*, volume 2, pages 205–209, Vienna, 1996.