Combining Prediction and Recognition to Improve On-Line Mathematical Character Recognition

Elena Smirnova, Stephen M. Watt
Ontario Research Centre for Computer Algebra,
The University of Western Ontario,
London, ON, Canada
{elena,watt}@orcca.on.ca

November 30, 2006

Abstract

This paper describes methods to increase the accuracy of mathematical handwriting analysis by using context information. Our approach is based on the assumption that likely expression continuations can be derived from a database of mathematical expressions and then can be used to rank the candidates of isolated symbol recognition.

We present how predicted continuations for an expressions are derived, how they are combined with the recognition candidates, and the effectiveness of the results. We first review the techniques we have used to build and represent a mathematical context database. We then describe different strategies for combining context information with results obtained from the recognition of individual characters. Finally we present a summary of a case study, using a fixed dataset of common mathematical expressions to test the accuracy of on-line analysis.

1 Introduction

1.1 Motivation

To build a mathematical recognizer one would naturally wish to adopt the already well-developed techniques for natural language recognition. However, three major aspects of mathematical notation distinguish handwriting analysis of mathematics from that of natural languages. Most notably, general mathematical expressions occupy a two-dimensional layout, which implies a combination of drawing and writing techniques within a single formula. Secondly, much larger alphabets are used, wherein some characters are only slight variations of others. This makes the usual methods for text-based character recognition insufficient. Furthermore, there is no fixed mathematical vocabulary that may be used for disambiguation.

Dictionary-based methods are essential for recognition in natural languages: most text recognizers choose candidates for individual letters by matching possible combinations of candidates to dictionaries entries. This explains, for example, why the Microsoft text recognizer identifies the word in Figure 1 as "cloud", even though the first and last strokes are identical and can be equally as well recognized as "*cl*" or "*d*".

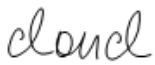

Figure 1: Dictionary-based methods are used to resolve ambiguities in character recognition for natural languages

Figure 2: Ink sample 1: is this "0" is "*o*", "*O*", "omicron", symbol of degrees or simple circle ∘ ?

Figure 3: Ink sample 2: is this an i or $\dot{z}$?

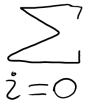

Figure 4: Example for mathematical context for symbols i and 0

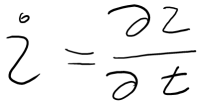

Figure 5: In this context a symbol from Figure3 becomes $\dot{z}$

1.2 Objectives

Just as character recognition in natural language text is improved by considering whole words, we wish to improve handwriting analysis for mathematics using larger context. Although there is no fixed vocabulary of words, human readers will use domain knowledge of common expressions. Our goal is to do the same thing automatically for mathematics.

Let us look at the following examples of context-dependent character identification: It obvious to a human reader that the symbol shown in Figure 2 is "0", when it appears within an expression such as in Figure 4 ($\sum_{i=\square}$), . The context information assures us that this character cannot be a letter "o", "O", "omicron", the degree symbol ° or circle ∘. On the other hand, a recognition system that is not context-aware, may equally well choose any of the these above candidates based on the best geometrical match.

A naïve automated recognizer might identify the character shown in Figure 3 as "i". This would be perfectly acceptable when this symbol occurs in the expression $\sum_{\square=0}$, as shown in Figure 4. However, if the same character is encountered in the expression $\square = \frac{\partial z}{\partial t}$ (Figure 4), the recognizer should choose $\dot{z}$ as the best candidate.

In previous work [8] we have presented a cross-application pen interface for recognizing handwritten mathematical expressions. In [10, 11] Watt and Xie have studied methods to improve the performance of character recognition for large mathematical alphabets based on character *features*. In the present work, we explore techniques to further increase recognition accuracy by using the *context* information of mathematical expressions.

This paper is organized as follows: Section 2 we describes our approaches to data mining to obtain the mathematical analogue of a "vocabulary". Section 3 shows how the prediction information can be derived from the context of the mathematical expression. Section 4 reviews methods to combine prediction and recognition results and presents a survey on how prediction affects the accuracy of character recognition. Section 5 presents our conclusions and outlines some directions for future work in this area.

2 Building Mathematical Dictionaries

It is not feasible to predict all possible mathematical formulae that people may write. However, we can determine a fixed subset of the most common expressions currently used in practice. For this we need to find a reliable source of mathematical content that is neither writer- nor area-specific. Then we can use a collection of the most popular formulae as a "dictionary" for mathematics.

2.1 Data Mining

In earlier work [9] Watt and So have studied the most popular mathematical expressions used in practice. As part of this work, they have analyzed and categorized more than 20,000 mathematical documents from

the mathematical arXiv service [3] for the period 2000-2004. We have used these results to create an initial dataset for building mathematical dictionaries.

2.1.1 Representation of mathematical content

We require mathematical data in a form that is easy to parse and analyze. By default, e-Print archives accept submissions in either of the PS or PDF document formats. Retrieving mathematical content from these formats is a rather complex task; therefore, we considered only those accompanied by TEX sources. We were able to obtain more than 40GB of TEX files from the arXiv service.

Although TEX documents contain explicit encoding of mathematical notation, TEX markup typically does not represent the semantics, or even the grouping, of the formulae. We partially overcome this deficiency by translating mathematical content from the TEX format to Presentation MathML [5]. Using this standard we are able to represent the structure of formulae as an XML trees, which also implicitly encode semantics. As part of earlier work [9], all mathematical expressions from TEX documents were converted to MathML, using a LATEX to MathML translator [4], developed at the ORCCA laboratory [1]. As result, we obtained a 7GB collection of 250,000 expressions, stored in 38,000 MathML files.

2.1.2 Serializing formulae

The theory of probabilistic prediction on strings is more straight forward than that for trees. Moreover text is written in a time order, which gives a series of symbols. Therefore, dictionary entries are represented as text strings. Consequently, in order to compile a dictionary of mathematical expressions, we had to translate the two-dimensional structure of every formula in our collection to a linear sequence.

An intuitive way to do this is to extract the mathematical characters from each expression in a certain way. However, it is not clear in which order two-dimensional formulae should be traversed to be useful for handwriting analysis. For instance, to enter the formula 1, some people would write the underscript $x \to 0$ right after entering the base expression *lim*. Others, might prefer to write the function $\frac{1}{x^2}$ first, then go back and add a limit variable. A similar ambiguous situation arises with operators that take both superscripts and subscripts. As we have noticed from handwritten samples collected with our Mathink system, the order in which people enter scripts often varies. For example, $\int_0^1$ may be written as $\int \nearrow^1$ or $\int^1 \searrow_0$. Another problem case is presented by fractions. For this we have decided to adopt the order in which the numerator appears first, followed by the fraction bar and then the denominator. For the multi-script constructs we have established an order corresponding to the pattern ${}_3^4\square_1^2$. Even though, the content of the subscript sometimes determines what will be entered in a superscript, in most cases the influence will not be significant. We are currently experimenting with a large test suite to determine whether these conventions will be suitable for most cases.

$$\lim_{x \to 0} \frac{1}{x^2} \tag{1}$$

In addition to the 504 mathematical literals, we encoded partial structure information in the string. Our primary concern was to represent spacial relations between individual characters in the expression. For this purpose, we introduced four meta-tags: _,, [,], <frac/> and <root/>. By serializing expressions in this manner, dictionary methods are able to distinguish between the sequences encoding $y\,2$ and y^2. Assuming that the first expression is more popular than the second, a mathematical dictionary preserving spacial information would be able to recommend 2 as a candidate for the character circled in Figure 6.a, while suggesting z for the symbol circled in Figure 6.b

We used this method of expression serialization to convert all formulae from the MathML files to plain strings. Each string contained a combination of meta-tags and mathematical symbols in Unicode format. For platform portability Unicode characters are represented by hexadecimal numeric entities. This allows to stored the strings as sequence of ASCII characters. The serialization process for the expression in Figure 1 gives the following sequence:

l, i, m, _{, x, →, 0,}, 1, <frac/>, x, ^{, 2}.

(a) (b)

Figure 6: Using spacial information to resolve ambiguity in character recognition

Sequence Length	Total # of subexpr.	Different Subexpressions		Max count	
		#	% of total	value	% of total
3	403,009,946	2,342,810	0.56%	3,879,431	0.98%
4		15,475,787	3.84%	807,713	0.2%
5		40,068,696	9.94%	196,482	0.0005%

Table 1: Quantity and frequency of subexpressions of each length

2.2 Collecting Sequences

After the serialization stage we obtained 250,000 sequences of mathematical literals and spacial tags. From these we have constructed all subsequences of length 3, 4 and 5. This allowed us to find common parts of formulae that rarely appear on their own. For example, the expression ∂x^2 is almost never used as a stand-alone formula, but often encountered as a denominator.

For each of the above lengths we obtained 403,000,000 subexpressions over an alphabet of 510 symbols (504 mathematical characters plus 6 spacial tags). Each of three collections was stored in a separate database, corresponding to the maximum length of its entries.

2.2.1 Sequence count

Obviously, not all of the 403,000,000 subsequences are different. In fact, many of them are identical, and on average less then 2% of the subsequences do not have duplicates in the collection. To avoid storing redundant information, all repeated sequences are represented in the database as a single entry, accompanied by its *count* – the number of times the subexpression appears in the original collection. Table 2.2.1 gives the number of different subexpressions encountered for each length. It also shows the count value of the most frequent sequence in each database.

2.2.2 Popular subexpressions

As expected, among subexpressions with high ranks there are commonly used formula fragments, such as $\sum_{i=1}$, $\sin\theta$, (x, y), $f(x)$, etc. As well as actual subexpressions, we also found popular *patterns* for formula structures, such as $\square_{\square=1}^{n}$. This entry, for example, shows that if an operator such as $\sum$ or $\prod$ has the lower bound equal to 1, then the upper bound is likely to be n.

Some of the subsequences we found in the top 10% of database entries could have been foreseen. Others were less obvious. For example, the most popular subexpression of length 5 among the 38,000 sample mathematical documents is "^{- 1} (", which corresponds to the formula fragment $\square^{-1}($. This is why we believe it is important to obtain the entries for the mathematical dictionaries from empirical data, rather than from assertions based on intuition or "common knowledge".

Extended sets of frequent subsequences encountered in mathematical contexts are published on the web site for the Mathink Project at ORCCA. The top entries for each of the three databases can be found under Research Results at [2]

2.3 Building Compact Dictionaries

Once we had populated the databases with subexpressions, we faced a new problem: Even without storing duplicates, the collections are too large. In the worst case, the size reached 660MB. This made the collections difficult to store and slow to access. Moreover, these databases were originally deigned to serve as mathematical dictionaries for assisting character recognizers. Given, that recognizer systems are typically designed to work on portable devices such as the Tablet PCs and Pocket PCs, dictionaries of this size are not usable. We therefore had to find a criterion to reduce subexpression databases to a more reasonable size without sacrificing their useful content.

2.3.1 Expression distribution

Our first observation is that the popularity of an expression is represented by its count, *i.e.* expressions with low counts are not used frequently enough to warrant being included in a dictionary. The second observation is that as count decreases, the number of seldom-used subsequences with that count increases, and there are few subexpressions that are very popular. The data we collected from the three databases showed similar dependency of number of different sequences of each counts on relative frequency. In each case the relation was monotonically decreasing with concave upward, and appeared to be negative exponential, as shown on Figure 7.

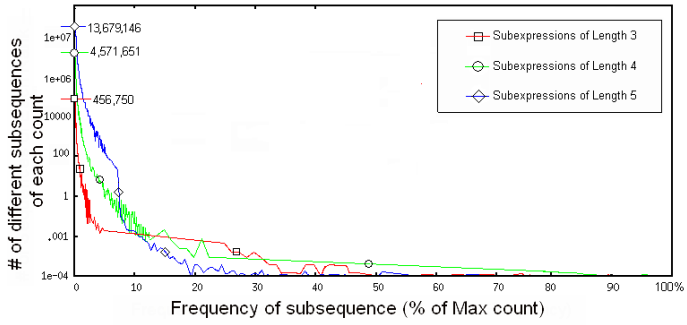

Figure 7: A diagram of distribution of mathematical subexpressions with respect to their frequency

2.3.2 Refining databases

The dependency in expression distribution suggests that by excluding the least popular expressions, we can significantly reduce the database size without reducing the use of the dictionary. We therefore explored what would be a reasonable point at which to cut-off the database entries. For subexpressions of each length we created three databases, corresponding to different levels of completeness: "detailed", "medium" and "light". The properties of each type are summarized in Table 2.3.2.

In "detailed" databases, we kept all the subexpressions with a rank of at least 100 (occurrence rate grater than 0.000025%). The size of each of these databases exceeds 10 MB, therefore this type is mostly used for research and experiments and is not intended for practical applications.

The "medium" type is created for practical use. It is designed to be integrated with recognition systems on fully-functional computers, including desktops, laptops and Tablet PCs. This type of database stores all expressions that were encountered more than 400 times (occurrence rate is grater than 0.0001%). The size of the non-compressed databases in this class varies from 1 to 3 Megabytes, depending on the length of subexpressions they store.

The "light" versions of practical databases contain only the very popular expressions with minimum count of 1000 (occurrence rates greater than 0.00025%). The databases of this type provide less variety in

Type	Purpose	Min Expr. Rank	Min. occurrence rate	# of different entries	Size
"Light"	Recognizers on PDAs	1000	0.00025%	30,000-40,000	0.3-0.5MB
"Practical"	Recognizers on TabletPC	400	0.0001%	80,000-200,000	1-3.4MB
"Detailed"	Experimental	100	0.000025%	220,000-400,000	10-22MB

Table 2: Types of compact dictionaries for subexpressions of length 3-5

the dictionary entries, compared to the medium type, and consequently, they need less space. The size of "light" databases does not exceed 500KB, which allows for their use on handheld devices such as PDAs or SmartPhones.

2.4 Mathematical Context Databases

Dictionary-based methods for recognition of natural languages are based on word matching. This strategy will not work in mathematics: Due to the wide variety of mathematical expressions, dictionary support in pen mathematics should be based on matching arbitrary parts of the formulae, rather than whole expressions.

As described in Section 1, recognition of mathematical characters is often determined by their local context. We suggest the use of collections of subexpressions, as described in Section 2.2, for creating *Mathematical Context Databases*. Thus, given a local context for a character in a handwritten formula, we will be able to determine whether a recognition candidate for this character would likely appear in an expression.

We shall call a length of the sequence encoding the preceding context as the *depth of the context*. As a part of mathematical expression, context is presented by two-dimensional structure; therefore, its encoding will contain meta-tags, that are also counted in total length of the encoding sequence. The *depth of Context Database* is the maximum depth of its entries.

To date, we have built nine context databases, of various depth of context and level of precision (see Section 2.3.2). Four of them, along with the tools for access and management, are included in the mathematical recognition software Mathink, presented in [7]. Figure 8 demonstrates a browser interface for Context Databases.

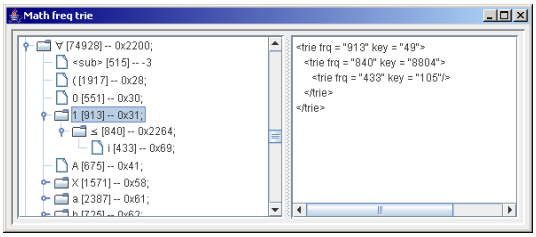

Figure 8: A browser for Mathematical Context Database

2.4.1 Organization of the databases

From the stage when the databases are constructed to the moment they are cleaned-up, their data collections contained hundreds of millions of entries. Therefore, we had to provide effective methods for data storage, access and modification. In particular, we required tools for quick strings searching and to add new entries without affecting the existing structure. To meet these objectives, we chose to organize each database as a

digitally indexed tree, often referred as a *trie* (from reTRIEve) [6]. Key features of its design allow effective encoding and accessing of sequences with a common prefix. Tries are commonly used for storing strings over an alphabet to organize large dictionaries in spell-checking programs or to represent large vocabularies for grammar analysis.

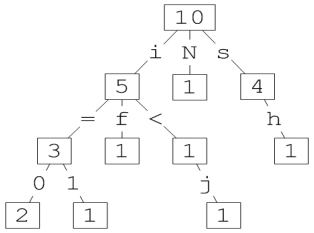

Figure 9: A trie storing expressions $i = 0$, $i = 1$, if, $i < j$, N, sh and s along with their ranks.

Besides encoding sequences of keys, tries may contain supplementary data, associated with the keys. In our implementation the trie encodes strings over the alphabet of 510 literals: The branches are determined by individual mathematical characters or the spacial meta-tags. The supplementary information we store in the trie represents the count of sequences encoded in the trie. Each node contains a number, corresponding to the count of the expression encoded by the path from the root of the trie to the node itself. Figure 9 shows the example of a trie, constructed from the following expressions: , "$i = 1$", "if", "$i < j$", "N", "sh", two instances of "$i = 0$" and three instances of "s".

2.4.2 Context and possible continuations

Each entry in a Context Database has two parts: *local context* and its *continuation*. Depending on the depth of the local context, the database can suggest a different number of continuations. For example, according to the fragment of a Context Database on Figure 10, a formula, ending with "$..i$" may have 5 possible continuations, while an expression ending with "$..i \in$" may have only 3: "I", "{" and "$\mathbb{N}$".

Symbols, found among possible continuations of a particular context can advise recognition systems which of the candidates are more appropriate in this given context. Thus, for example, the Context Database would suggest that a handprinted character on Figure 11 should be recognized as a capital letter I, and not as, for example, a number 1.

This notion can be more precise by considering the counts as defining a measure. Moreover, weights of words can be used to estimate a "likelihood" for the corresponding continuation. Thus, according to the fragment in Figure 10, the letter I would be four times as likely to continue the expression $i \in$ than the open brace symbol, and thirteen times as likely than the symbol $\mathbb{N}$. In the next two sections we show in detail how to use this count information in cooperation with character recognition.

3 Character Prediction in Mathematical Expressions

As described in the previous section, Context Databases suggest possible continuations for an input sequence. For this, we need to extract the local context from a partially-entered formula and match it against Context Database entries. In this section, we describe a technique to obtain the local context from handwritten mathematical expressions. We also demonstrate by example how to use context information to calculate predictions for possible continuations.

3.1 Initial Settings

The context databases and prediction methods are designed to be used for character prediction in mathematical handwriting recognition software. Usually, linear mathematical handwriting flows in one direction.

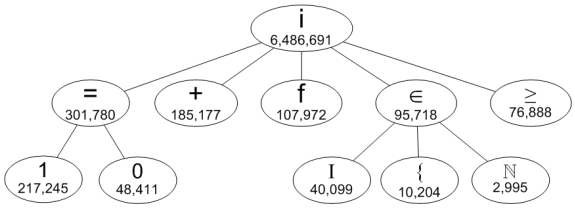

Figure 10: An entries of Context Database, starting with literal i

Figure 11: An ambiguity to be resolved by Context Database

The current version of our recognition system, Mathink, handles Western-style handwriting: from left to right. Furthermore, the design of our system assumes that every character is recognized immediately after it has been entered. This strategy ensures that by the time a new symbol is written, all previous characters have been recognized correctly, so we can use them as data in computing prefixes for further analysis.

These two observations imply that for our existing settings we can consider only context that *precedes* the recognized character. For this reason we have organized the Context Databases as prefix-defined, *i.e.* they provide easy access to *continuations* of a given context, but do not offer efficient methods to retrieve information about characters that may precede the context. In this paper we do not address the question of using the *surrounding context*. However, the approach of using the local surrounding context in global expression recognition appears to be promising.

3.2 Retrieving Context from Mathematical Expressions

First of all, we require a technique to identify the local context of any character in a handwritten formula, regardless of whether this character has been recognized or just entered.

For clarity, we demonstrate the method for context retrieval by example: Given the partially recognized formula of Figure 12, suppose the user has just added a new ink character below the horizontal bar. Now, *before* recognizing the new ink, the system has to identify in which context this character appears. To answer this question, first, it must analyze and serialize the structure of the expression. From the resulting sequence of linearized expression structure it then needs to extract the part that corresponds to the neighborhood areas on the left, above or sometimes below the handwritten character.

To determine the structure of the expression, including the newly entered ink, we cannot use a regular structure analyzer, since it requires the character to be recognized first. On the other hand, we cannot leave the new entry out while computing the context. For instance, in our example the final result depends on the fact that a new character is written *under* the rest of the expression. To resolve this situation, we use a preliminary structure recognizer: a "layout analyzer" that estimates the role of a new entry in the formula structure, by using only the size and the position of its bounding box. Thus, the expression from Figure 12 will be translated into a layout tree, presented on Figure 13.

The layout tree can be easily converted into a linear sequence, using the serializing methods presented in Section 2.1.2. For example, the tree in Figure 13 would be linearized as <root/> 3 <frac/>. Then, the context is defined by the last $n-1$ literals of this sequence, where n is the depth of the Context Database. So, if we use a Context Database of depth more than 4, we can send the whole sequence to the next stage for computing potential continuations.

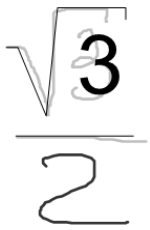

Figure 12: Partially recognized expression: what is a most likely continuation of $\frac{\sqrt{3}}{\cdot}$?

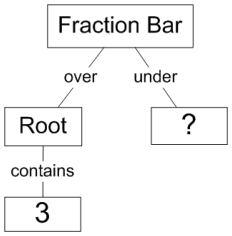

Figure 13: Layout tree of an incomplete formula $\frac{\sqrt{3}}{\cdot}$

3.3 Calculating Character Prediction

To compute possible continuations of a given context, we send a query to the Context Database. If this succeeds, we get the result as a weighted vector of alternatives, which we can normalize and export as a set of ranked candidates for character predictions.

If the current context of length l is not found in the Context Database, we try again with a subsequence of the last $l-1$ literals and repeat the procedure until aa entry is found or the the subsequence is empty. The last case indicates that the original mathematical expression has a relatively uncommon pattern, so we are not able to predict its possible continuations.

For the context of the example from Section 3.2, the "Practical" (medium-sized) Context Database of depth 5 returned no results. On the second try with the reduced context 3 <frac/> we obtained a set of mathematical characters rated according to their original ranks, as shown in Table 3.3.1.

3.3.1 Normalization

Possible continuations of a given context are ranked using the count of a corresponding entry in the Context Database. This entry, in fact, is a concatenation of the context with the candidate character. For brevity, we refer to the count of the entry as the *candidate rating*. Count value thus can be used as a sorting criterion: higher count indicates a better candidate. However, the count of an expression is an absolute characteristic, indicating the its total number of occurrences in the mathematical dictionary. It is reasonable to consider character prediction as the probability of the next symbol to appear in a formula within a given context. For this purpose the candidate rating should be normalized.

There are several common strategies for normalizing cumulative values. The first method is to calculate an absolute ratio of the candidate rating to the count of the context it continues. Then, the normalized rating of a character, $\mathcal{P}_1(char)$ is given by equation (2). The second approach is to divide the count of a character by the sum of counts of all candidates (3). The difference with the first method is that the count of the context is not always equal to the sum of counts of all its continuations. The third method normalizes by the maximum count among all candidates (4). In our system we implemented the third method, because in this case the normalized rating of the best candidate will always be equal to one, and therefore different

Char	Rank
1	1606
2	19096
3	961
4	7015
5	722
6	416
8	1040
n	448

a. Cumulative Ranks

Char	Ranks Normalized with		
	Context rank	Sum of ranks	Max rank
1	0.046	0.051	0.084
2	0.549	0.610	1
3	0.028	0.031	0.050
4	0.202	0.224	0.367
5	0.021	0.023	0.038
6	0.012	0.013	0.022
8	0.030	0.033	0.054
n	0.013	0.014	0.023

b. Normalized Ranks

Table 3: Possible continuations for the context "3 <frac/>" and their ranks

Char	Recognition Confidence	Rank
Z	0.912	1
z	0.841	2
2	1.61	3
≀	0.102	4
≥	0.034	5

a. Ranked character recognition results

Char	Prediction Confidence	Combined Confidence	New Rank
Z	0	0.912	2 (↓)
z	0	0.841	3 (↓)
2	1	1.61	1 (↑)
≀	0	0.102	4
≥	0	0.034	5

b. Re-ranked recognition candidates

Table 4: Recognition results for the handprinted character on Figure 12

recognizers will have comparable ranking. The columns of Table 3.3.1.b present normalized ratings for the candidates listed in Table 3.3.1.a.

$$\mathcal{P}_1(char) = \frac{\text{Count}(context|char)}{\text{Count}(context)} \tag{2}$$

$$\mathcal{P}_2(char) = \frac{\text{Count}(context|char)}{\sum_{all\ candidates} \text{Count}(context|candidate_i)} \tag{3}$$

$$\mathcal{P}_3(char) = \frac{\text{Count}(context|char)}{\max_{all\ candidates} \{\text{Count}(context|candidate_i)\}} \tag{4}$$

From the perspective of expression analysis, possible continuations of a given context can be considered as *prediction candidates* for a newly entered character. Then the normalized rank of each candidate can be used as a *prediction confidence* in further analysis. In the next section we show how prediction information may affect the results of isolated character recognition.

4 Combining Prediction and Recognition

Prediction information can be further used in handwriting analysis to re-rank the recognition candidates. This can be done in a number of ways, and we suggest the following strategy: From a character recognizer we get the candidates that are ranked according to their recognition certainty, as for example shown in Table 3.3.1.a. Then for every recognition candidate, we combine its recognition certainty with the confidence of a matching prediction candidate. If the character is not found among possible continuations, we treat its prediction confidence as zero. The new rank of a recognition candidate is based on the value of the

combined confidences. As shown in Table 3.3.1.b, the correct candidate for the denominator in Figure 12 is "promoted" owing to its prediction confidence.

In this example we used a simple voting method to combine the confidence values. This case is over-simplistic, in that the prediction and recognition results were intersected only over one character, which consequently was re-ranked as the best candidate. However, in practice we commonly face situations where prediction and recognition results contradict each other. In these cases we have to decide how to combine the confidence values of character prediction with those of recognition.

4.1 Combination Functions

If we define the final rank of a candidate as a function of two arguments: *recognition certainty* $\mathcal{R}$ and *prediction confidence* $\mathcal{P}$, then we need to find a reliable method to combine these values. Regardless of the type, the combination function $\mathfrak{C}$ shall have two parameters. These are weight coefficients for recognition and prediction confidences. For concreteness, in this work we have considered three types of combination functions $\mathfrak{C}(\alpha, \beta)$:

$$\mathfrak{C}_1(\alpha, \beta) = \alpha \cdot \mathcal{R} + \beta \cdot \mathcal{P} \quad (5)$$

$$\mathfrak{C}_2(\alpha, \beta) = \mathcal{R}^\alpha \cdot \mathcal{P}^\beta \quad (6)$$

$$\mathfrak{C}_3(\alpha, \beta) = \alpha \cdot \exp(\mathcal{R}) + \beta \cdot \exp(\mathcal{P}) \quad (7)$$

Within this setting our goal is to estimate the best combination of α and β maximizing $\mathfrak{C}_i$ over a set of handwriting samples. We must avoid choosing the coefficient based on the results of too few experiments with single users' handwriting. Unfortunately this tactic of estimating the default parameters in handwriting analysis is commonly practiced by system developers. The approach we took is instead based on the use of empirical data obtained from larger test suites. Although in our experiments we used prototype test suites, the approach we have developed can be extended to larger collections to obtain more general results.

4.2 A Testbed

We chose to search for a combination of the coefficients α and β that optimizes the recognition rate for a fixed set of handwritten expressions. To do this, we have built a test suite that had used seventeen representative formulae from different areas of mathematics. We then collected handwritten samples of these formulae from 10 different writers.

To analyze the experimental data, we have developed a software environment for testing based on the Mathink recognition system [7]. For this purpose Mathink has been extended with modes and tools specific to the experiment. These include an ink input mode for collecting handwritten samples associated with a loaded test suite, a module for ink annotation and an engine for analysis of recognition performance on various grids of coefficients. The interface of Mathink in testing mode is presented in Figure **??**.

Once all the handwritten samples were collected, we ran the Mathink recognizer on a set of 170 expressions, using the grid $[1..10] \times [0..10]$ for α and β, repeating calculations for each of the three combination formulae.

4.3 Experimental Results

The experiments have shown that there is always a combination $\langle \alpha, \beta \rangle$ that increases recognition accuracy for each writer individually, for every formula and, even better, for an average of all individual writers and formulas. The experiments have also confirmed that heavily relying on prediction, when recognition confidence is low will significantly decrease the overall recognition rate, in the worst case up to 10%.

The diagrams in Figures 14-18 represent typical results of the test cases for $\mathfrak{C}_1$ - $\mathfrak{C}_3$. They show the increase of accuracy compared to the results of character recognition, obtained without using prediction information. The overall distribution of the coefficients α and β yields the following patterns of Figures 15 - 19.

Our search for good values for α and β has been based on a naïve evaluation of points on a grid. More sophisticates optimization strategies should prove both more efficient and yield better results.

5 Conclusion

We have explored methods to use the context to improve on-line mathematical handwriting analysis. The key idea of our approach is to employ empirical data to increase accuracy of character recognition in large mathematical alphabets.

While dictionary-based methods are widely practiced in handwriting recognition for natural languages, there is no fixed mathematical vocabulary to support them in mathematical analysis. Nevertheless, in practice some mathematical expressions are used much more often than others. This observation implies that we can define a measure on the set of expressions and use this as a dictionary for mathematics.

At the same time we have realized that the criterion "I know this occurs often" for building such dictionaries can do more harm than good. We therefore chose to build user-independent sets of mathematical expressions used in practice. In this work we have discussed how mathematical dictionaries and context databases can be built from sets of user-independent mathematical expressions used in practice. We also showed how we derive the local context from the partial structure of mathematical formulae and demonstrated how this context information can be matched against Context Databases.

On the practical side, we have incorporated context prediction in our mathematical recognizer environment, Mathink. Furthermore, we have investigated methods of combining prediction information with recognition results. We have built a framework based on the Mathink system to validate our approach. We used this testing environment to drive an experimental study designated to determine the best combination of context prediction and character recognition. The experimental results confirmed the existence of a combination method, increasing the accuracy of isolated character recognizers.

Our on-going research focuses on determining the best strategies for such combinations, using the context databases and character recognizers specific to particular areas of mathematics. We are also working on enlarging our data collection with more handwritten samples to obtain the more general results.

References

[1] *Ontario Research Centre for Computer Algebra*, `http://www.orcca.on.ca`.

[2] *The repository of frequent subsequences encountered in mathematical contexts*, `http://www.orcca.on.ca/PenMath/materials/researchResults/MathContext/`.

[3] *ArXiv e-Print Archive*, `http://arxiv.org/`, 2000-2004.

[4] *LaTeX to MathML on-line converter*, `http://www.orcca.on.ca/MathML/texmml/textomml.html`, (c) 2002-2005.

[5] R. Ausbrooks, S. Buswell, D. Carlisle, S. Dalmas, S. Devitt, A. Diaz, M. Froumentin, R. Hunter, P. Ion, M. Kohlhase, R. Miner, N. Poppelier, B. Smith, N. Soiffer, R. Sutor, and S. Watt, *Mathematical Markup Language (MathML) version 2.0 (second edition)*, World Wide Web Consortium, 2003.

[6] Paul E. Black, *"trie", from dictionary of algorithms and data structures*, `http://www.nist.gov/dads/HTML/trie.html`, NIST, 2003.

[7] Elena Smirnova and Stephen Watt, *A pen-based mathematical environment* Mathink, Tech. report, University of Western Ontario, `http:\\www.orcca.on.ca\TechReports\TR-06-05`, 2006.

[8] Elena Smirnova and Stephen M. Watt, *A context for pen-based computing*, Maple Conference 2005, Maplesoft, 2005, pp. 409–422.

[9] Clare M. So and Stephen M. Watt, *Determining empirical properties of mathematical expression use*, Fourth International Conference on Mathematical Knowledge Management, (MKM 2005), Springer Verlag, 2006, pp. 361–375.

[10] Stephen M. Watt and Xiaofang Xie, *Prototype pruning by feature extraction for handwritten mathematical symbol recognition*, Maple Conference 2005, Maplesoft, 2005, pp. 423–437.

[11] _____, *Recognition for large sets of handwritten mathematical symbols*, IEEE International Conference on Document Analysis and Recognition (ICDAR 2005), vol. 2, 2005, pp. 740–744.

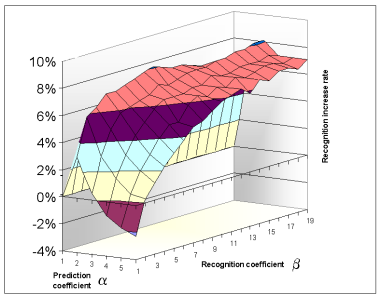

Figure 14: Results for user 1. the best improvement for $\mathfrak{C}_1$ is 8.1% at (9,1)

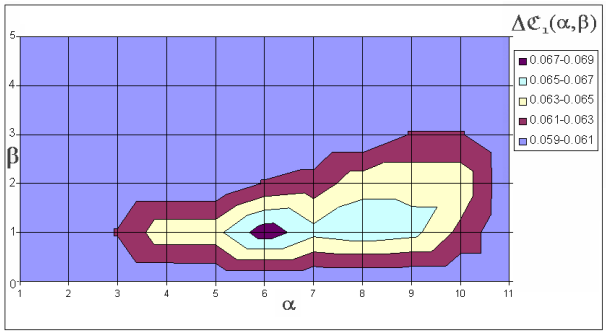

Figure 15: Improvement in $\mathfrak{C}_1$ as a function of α and β

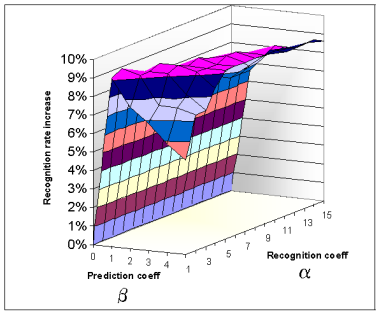

Figure 16: Results for user 2. the best improvement for $\mathfrak{C}_2$is 9.5% at (2,1), (4,3) and (7,4)

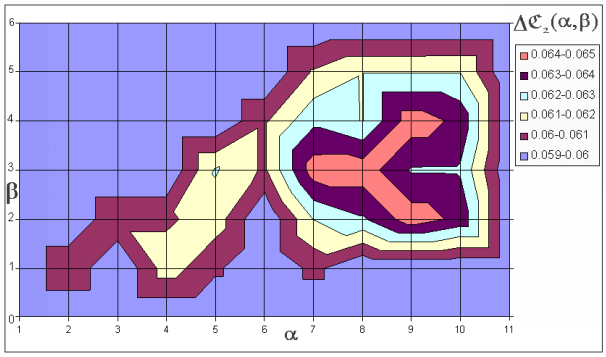

Figure 17: Improvement in $\mathfrak{C}_2$ as a function of α and β

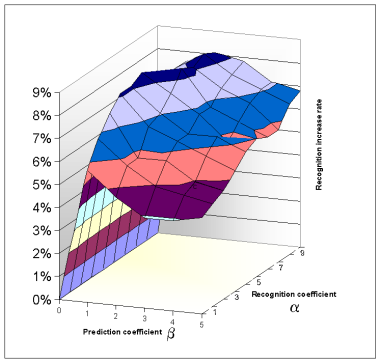

Figure 18: Results for user 3. the best improvement for $\mathfrak{C}_3$ is 7.5% at (6,1), (7,2) and (9,2)

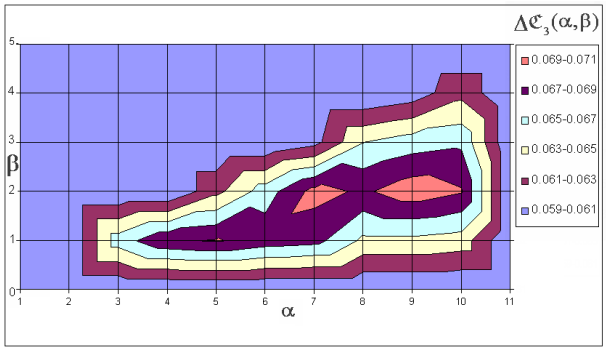

Figure 19: Improvement in $\mathfrak{C}_3$ as a function of α and β