Baseline Extraction-Driven Parsing of Handwritten Mathematical Expressions

Lei Hu, Kevin Hart, Richard Pospesel* and Richard Zanibbi
*Department of Computer Science, *School of Interactive Games and Media*
Rochester Institute of Technology, Rochester, USA
{*lei.hu, kth1775*}*@rit.edu, pospeselr@gmail.com, rlaz@cs.rit.edu*

Abstract

We generalize recursive baseline extraction algorithms for symbol layout analysis in math expressions so that handwritten strokes may be provided as input. Specifically, baseline extraction is used for lexical analysis in a modified LL(1) parser, returning a set of candidate symbols when the leftmost or next symbol along the current baseline (from left-to-right) is requested by the parser. Candidate symbols are used to produce a forest of parse trees, and the highest ranked parse returned. Hidden Markov Models (HMMs) are used for symbol classification, and horizontal adjacency between symbols is determined using two probabilistic quadratic classifiers, one for ascenders (e.g. 'A') and another for centered and descender symbols (e.g. 'y' and 'x'). The system placed second in the CROHME 2011 handwritten math recognition competition.

1. Introduction

Converting math expressions written on a tablet computer to a *symbol layout tree* (e.g. in LaTeX) is a structural pattern recognition problem. Given a set of handwritten strokes, we want to determine which strokes form symbols, the type (class) of each symbol, which symbols are adjacent on baselines, and the hierarchy of spatial relationships between baselines [5, 15]. For the expression $\frac{2x}{3}$ there are three baselines: $-$ (the *main baseline*), $2x$ (above $-$), and 3 (below $-$).

Math symbol classification is difficult due to the number of symbols in use: for example, tools such as detexify[1] are used to make it easier to find symbol codes amongst the hundreds available in LaTeX. Handwritten symbols vary significantly in appearance and form across writers: for example, some write 'x' using two outward curves (')(') while others use two intersecting lines ('$\times$'). Segmentation of strokes is also difficult, particularly when a cursive writing style that connects symbols in a single stroke is used [3]. Symbol layouts such as subscripting vary significantly across writers, and are often ambiguous [13]. Detecting fractions whose arguments extend past the fraction line is challenging, for example. These tasks are interdependent [15], and context is often needed for disambiguation [12].

To exploit contextual/linguistic constraints, commonly stochastic or fuzzy language models are used when recognizing handwritten expressions (see [2, 4, 7, 9, 14], and [15]). A grammar defines legal symbol layouts, and a two-dimensional parser searches through alternative segmentation, classification and layout hypotheses as productions are applied. The most likely (legal) interpretation is returned. Like all syntactic methods, this approach can be brittle; expressions not detected as in the language result in no output. Error-correcting parsing [6] can mitigate this problem.

Baseline extraction parses symbol layout by locating symbols on the main baseline from left to right, and then repeating this process in regions around baseline symbols recursively (e.g. for symbols located in an exponent [16]). The method is deterministic, and requires symbol bounding boxes and their typographic/layout classes (e.g. ascender, descender, centered, root, non-scripted) as input. The (implicit) expression language accepts nearly all symbol arrangements, but does not consider math expression syntax. In a parser-driven system, interpreting '2+' for an input would lead to backtracking, and other interpretations being considered.

In this paper we generalize baseline extraction for use with handwritten strokes, embedding it within the lexical analyzer of a modified recursive descent LL(1) parser. Providing alternatives for leftmost and horizontally adjacent symbols along baselines allows multiple interpretations (i.e. symbol layout trees) to be

[1] http://detexify.kirelabs.org/classify.html

considered. In the next section we introduce a simple stroke data structure for finding baseline symbols, the *Left Blocking Tree (LBT)*. Hidden Markov Models are used for symbol classification, and quadratic classifiers to identify symbols that are adjacent to a given symbol on the current baseline. The system placed second in the CROHME 2011 handwritten math recognition competition [13]. We describe some extensions, and then present and discuss new results for the system on the CROHME 2011 dataset.

2. Left Blocking Trees (LBTs)

To identify the leftmost symbol on the main baseline of an expression, we first need to identify the leftmost strokes. We do this using a simple data structure, the Left-Blocking Tree (LBT), illustrated in Figure 1. In an LBT, an edge between two nodes (strokes) represents the child node (stroke) being overlapped vertically at left by the parent node (stroke). More than one stroke may be unblocked at left (see Figure 1).

To construct an LBT, strokes are sorted by leftmost x-coordinate from left-to-right, and the first stroke (s_1) is placed in a child node of the root, representing the left end of the expression. We then traverse the list of n strokes in sorted order. For stroke s_k, we search right-to-left for a stroke s_b ($s_{k-1} \geq s_b \geq s_1$) that vertically overlaps (blocks) stroke s_k. If a blocking stroke is found, an edge is added from the node for stroke s_b to the node for stroke s_k. If no stroke blocks s_k, a node for s_k is added as a child of the root node. Each time a new baseline symbol is identified, strokes for the symbol are removed from the LBT, and the LBT is re-built.

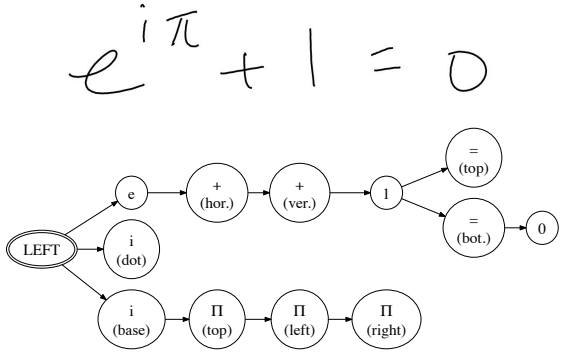

Figure 1. Left Blocking Tree (LBT) for an Expression. Three strokes do not have a vertically overlapping stroke at left: the e, and the two strokes in the i

3. Detecting Baseline Symbols in Strokes

We now address detection of candidate baseline symbols from within a stroke set. Section 4 describes how these methods are used for lexical analysis.

Leftmost Symbol: For each stroke that is a child of the root in the LBT, we find the two closest neighboring strokes, with stroke distances determined by the two closest sampled points on the strokes; intersecting strokes are automatically joined. This produces a set of candidates for the leftmost baseline symbol, with the intention of capturing leftmost baseline symbols that are not leftmost in the expression, such as when summations have limits that extend past the operator's left end. Each leftmost symbol candidate is classified using an HMM (see Section 5), and strokes for the candidate are removed from the LBT for the associated parse.

Adjacent Baseline Symbols: After the leftmost baseline symbol is located, additional baseline symbol are detected left-to-right. The method used for finding the next baseline symbol depends upon whether the current baseline symbol is:

1. Non-Scripted (e.g. '+', with no sub/super-scripts),
2. an Ascender (e.g. 'A'),
3. or a Centered (minim) symbol (e.g. 'x').

Descender symbols (e.g. 'y') have their descender portion cut off before analyzing layout using a simple projection-based method, and are then treated as Centered symbols. For Non-Scripted symbols, we simply repeat the procedure for finding leftmost baseline symbols, using the current LBT.

For Ascender and Centered symbols, we search for the leftmost unused stroke where the probability of horizontal adjacency is greater than a superscript or subscript relationship, given that the current symbol is an Ascender or Centered symbol. We use two quadratic classifiers defining the conditional probability of superscript, adjacent at right, and subscript relationships between the current and a candidate baseline symbol, e.g. $P_{Centered}(superscript \mid (-0.2, 0.1))$. The feature vector represents the top and bottom y-coordinate offsets for the next baseline symbol candidate's bounding box relative to the top of the current baseline symbol. The distances are normalized by the height of the current baseline symbol. Values are negative for coordinates above the top of the current symbol's bounding box. A stroke is detected as adjacent to the current baseline symbol if the probability for adjacency at right is higher than superscript or subscript. Note that the Gaussian spatial relationship densities are trained using *symbols* from ground truth, but we detect adjacency using individual strokes.

Just as for leftmost baseline symbols, once an adjacent stroke is detected we consider its two closest neighboring strokes to identify candidates for the next baseline symbol. When a new baseline symbol is chosen, its strokes are removed from the current LBT, and the LBT is rebuilt.

Partitioning Strokes not in Baseline Symbols: Unused strokes at left of a newly detected baseline symbol are associated with regions of the current baseline symbol. For now, we simply partition strokes based on the vertical center of the new baseline symbol, and the right-most end of the current baseline symbol (there are a small number of special cases to handle for the first and last symbol on a baseline). The center of the bounding box for each unused stroke is used to determine which region of the current symbol (above, below, superscript, subscript, contains) that the stroke belongs to. For each new region, an LBT is created for strokes in the region.

4. Modified LL(1) Parser

Our parser produces a set of *symbol layout trees* consistent with a context-free grammar (LaTeX math expressions are an example of symbol layout trees [15]). Efficient parsing techniques (e.g. CYK) could be employed, but we used LL(1) for its simplicity, constructing a forest of layout trees depth-first and returning the highest ranked layout tree. We represent legal layout trees using a positional grammar [10], where spatial relations act as terminals along with symbols. Here is a simple arithmetic language as a *non-left recursive* positional grammar:

$$
\begin{array}{l}
E \rightarrow *FRAC \mid *FRAC\ *OP\ E \\
E \rightarrow *DIGIT \mid *DIGIT\ *OP\ E \\
*OP \rightarrow + \mid - \\
*FRAC \rightarrow - \\
*DIGIT \rightarrow 0 \mid 1 \mid 2 \mid \ldots \mid 9 \\
[[\mathbf{Layout}]] \\
*FRAC \rightarrow \text{above } E \text{ below } E
\end{array}
$$

Examples of legal expressions include: $2 + 2$, and $\frac{1}{2} - \frac{1}{4}$. Legal relationships between baselines are given in the **[[Layout]]** section of the grammar. For example, nonterminal *FRAC represents a line with required subexpressions *above* and *below* the line.

Lexical Analysis: Consider starting a parse by applying $E \rightarrow *DIGIT\ *OP\ E$. The leftmost baseline symbol is requested from the parser whenever considering strokes in a new region; here we seek the leftmost main baseline symbol, considering only associated symbol classes belonging to $*DIGIT$. The parser invokes the adjacent baseline symbol detector when locating symbols following the current symbol in a production, such as for $*OP$ and E. Each time a baseline symbol is requested, if more than one candidate is returned, the candidates will be considered in turn, producing a forest of parse trees. To reduce the combinatorial explosion, for each nonterminal that generates a set of terminals (e.g. *OP and *DIGIT in our example), we use just the highest probability symbol class.

As baseline symbols are found, symbols at left of a symbol are partitioned into non-baseline regions (see previous Section) - these are represented by child nodes of a symbol (terminal) in the parse tree, with all strokes associated with a region (e.g. $above$) and the associated non-terminal from the grammar stored in the node. Parsing is *depth-first*: if there are nested regions, we parse strokes for the child regions of the current symbol before completing the current baseline.

Ranking: To score a parse (layout tree), for each stroke we add the classification probability of its associated symbol in the interpretation. Given an expression with n strokes, where $Parse(m)$ represents the m-th valid parse, s_k represents the k-the stroke and P_k is the probability for the symbol that s_k belongs to in $Parse(m)$, the score for $Parse(m)$ is calculated as: $Score(Parse(m)) = \sum_{k=1}^{n} P_k$. The highest scoring parse is then returned. This scoring function is naïve, considering only symbol classification - it would be beneficial to also consider spatial relationships between symbols [2, 9].

5. HMM-Based Symbol Classification

Classification of a candidate symbol (i.e. stroke set) is performed using Hidden Markov Models (HMMs) to represent handwritten symbols and strokes in time-order [8]. For each symbol class, we design a continuous left to right HMM (i.e. a linear topology) with six hidden states and five features. A mixture model of five Gaussians is used to represent the stroke feature distributions observed within each state at a single sample point on a stroke. A variant of segmental K-means is used to initialize the Gaussian Mixture Model parameters [8]. Symbol recognition rates reported in this section are resubstitution estimates, obtained by training and testing on the CROHME Part-II training data [13].

Symbol Priors: The symbol class distribution for the CROHME 2011 data is skewed. Using the product of the symbol priors and the posterior probabilities from the HMM, the isolated symbol classification rate increased to 92.7% from 91.7% (HMM only).

Stroke Sampling: The number of points uniformly sampled from each stroke determines the resolution of

Table 1. Results for CROHME 2011 Dataset. Successful and failed parses, and system failures (Time-out and Out-of-Memory) are shown at right. Metrics: % strokes in a symbol with correct class (ST_{rec}), % correctly segmented symbols in ground truth (SYM_{seg}), recognition rate for correctly segmented symbols (SYM_{rec}), and expression recognition rate (EXP_{rec})

DATA (NUM. FILES)	ST_{rec}	SYM_{seg}	SYM_{rec}	EXP_{rec}	Out	Fail	Time	Mem
Train Part-I (296)	56.07%	54.88%	94.62%	19.26%	223	69	4	0
Test Part-I (181)	41.69%	47.35%	87.00%	13.26%	120	61	0	0
Test Part-I w. Part-II Classifier (181)	42.12%	48.67%	86.54%	14.36%	120	61	0	0
Train Part-II (921)	9.95%	11.64%	91.88%	5.32%	266	112	445	98
Test Part-II (348)	17.74%	22.66%	80.16%	6.03%	193	36	92	27
Test Part-II w. Part-I Grammar (348)	25.57%	31.01%	84.62%	8.05%	183	154	11	0

features. We found that the recognition rate (92.7%) obtained for 30 points was 0.2%-2% higher than when sampling using additional points (35, 40, or 45).

Window Size: Three features need neighboring points for their calculation. If we use a window size of 7, then for each point the previous 3 sample points and following 3 points in time series will be used. For a fixed window size, points near the stroke start and end do not have sufficient neighbors to compute the features; we use value 0 in this case. We considered variable window sizes, where smaller window sizes are used as needed to obtain non-zero feature values near stroke ends. Using a fixed 5-point window produced a recognition rate (92.7%) that was 0.8%-6.5% higher than when using variable-sized windows for odd window sizes from 3 to 15 sample points.

New Feature: We use four local features from [8]: cosine of the slope, normalized y-coordinate, normalized distance to stroke edge and sine of the curvature. Three additional features considered were negative sine of local curvature, absolute sine of local curvature, and cosine of the diagonal. Local curvature for the current point $(x(t), y(t))$ is defined by the angle between the lines connecting the previous and following two points $(x(t-2), y(t-2))$ and $(x(t+2), y(t+2))$. Cosine of the diagonal is the angle between the main diagonal of the bounding box and the line connecting the current point and the left top corner of the bounding box. Although negative sine of the slope is related to cosine of the slope, it represents the vertical direction of pen movement: when the pen moves upwards, downwards or horizontally, its value will be positive, negative or zero. Adding the negative sine of slope as a fifth feature produces the largest improvement, obtaining a 93.9% symbol recognition rate.

6. Results on CROHME 2011 Dataset

The dataset used to evaluate our system is from CROHME 2011 [13]. The training and test sets have two levels of complexity, Part-I and Part-II. Part I has 36 symbol classes, and Part II has 56 symbol classes and a more complex grammar. An earlier version of our system placed second in CROHME 2011, with expression rates of 4.4% (Part-I) and 2.6% (Part-II), training the system using just the provided data sets. The winning system for CROHME 2011 employed a Stochastic Context-Free Grammar (SCFG) and Cocke-Younger-Kasami (CYK) parsing [1], with expression rates of 29.3% (Part-I) and 19.8% (Part-II). A system from the organizers [2] was also run, using a neural net, grammar and dynamic programming to minimize symbol and layout penalties. The expression rates were 40.88% (Part-I) and 22.41% (Part-II). Some systems (including the organizers') were trained using data in addition to the CROHME training sets.

Rates are low partly because of the effort needed to construct tools[2] and recognition systems from scratch (as our group did). Also, stroke data is provided in different co-ordinate systems and resolutions. Writers are from multiple countries (France, India, and Korea), with a large variation in both symbol forms and writing style. Finally, writer sets for the Part-II training and testing data are disjoint, making Part-II a difficult task.

Table 1 provides results for our current system on the CROHME 2011 data, trained using the CROHME training sets. We show the number of expressions parsed successfully (Out) as well as the number of parse failures, when no legal expression was produced (Fail). Our system uses a time-out threshold of 5 minutes, after which the parse stops without producing output (Time). Sometimes our system also runs out of memory (Mem).

Our updated system is able to parse more expressions than before, and obtains better expression recogni-

[2] http://saskatoon.cs.rit.edu/inkml_viewer (data visualization)

tion rates of 13.26% (Part-I) and 6.03% (Part-II). Interestingly, using the Part-II symbol classifier for the Part-I data increases the expression rate to 14.36%, because the Part-II training data provides more training samples (see Table 1). Another interesting result is that using the Part-I grammar to parse the Part-II test data increases the expression rate to 8.05%, 2.02% higher than when using the Part-II grammar, even though the number of expressions producing outputs *decreases* by 10. This is because the Part-I grammar has a simpler language model with fewer alternatives, improving recognition for expressions consistent with the grammar while leading to many parse failures (154). In contrast, our LL(1) parser is running out of time and memory in many cases when using the Part-II grammar.

Our parse ranking function sometimes prevents the correct parse from being ranked highest. Our current implementation is also inefficient, repeatedly rebuilding LBTs. Additional constraints (e.g. using a minimum spanning tree [11]) may improve segmentation accuracy and speed. Finding adjacent symbols using the leftmost stroke with a probability of adjacency higher than sub/super-script causes parse failures for some expressions where symbols 'dip,' such as if one writes '10' slanting down to the right, so that the system detects '1_0', which is illegal. Finally, grammars may be revised to accept more inputs, with parses illegal in the original grammar being penalized in the ranking.

With further investigation and changes as described above, we believe that baseline extraction-driven parsing will provide a simple but competitive methodology for handwritten math recognition.

7. Conclusion

We have presented a generalization of baseline extraction from operating on symbols to handwritten strokes, as part of a lexical analyzer for LL(1) parsing. Our results on the CROHME 2011 data set have been improved over our earlier version of the system, and we have identified future directions for research in baseline extraction-driven parsing. The performance of our current system is modest but encouraging in light of the simple segmentation and parse ranking methods used.

Acknowledgements: This material is based upon work supported by the National Science Foundation under Grant No. IIS-1016815.

References

[1] F. Alvaro, J.-A. Sanchez, and J. Benedi. Recognition of printed mathematical expressions using two-dimensional stochastic context-free grammars. In *Proc. ICDAR*, pages 1225 –1229, Beijing, Sept. 2011.

[2] A. Awal, H. Mouchere, and C. Viard-Gaudin. Improving online handwritten mathematical expressions recognition with contextual modeling. In *Proc. ICFHR*, pages 427 –432, Nov. 2010.

[3] R. G. Casey and E. Lecolinet. A survey of methods and strategies in character segmentation. *TPAMI*, 18(7):690–706, 1996.

[4] M. Celik and B. Yanikoglu. Probabilistic mathematical formula recognition using a 2D context-free graph grammar. In *Proc. ICDAR*, pages 161 –166, Sept. 2011.

[5] K.-F. Chan and D.-Y. Yeung. Mathematical expression recognition: A survey. *IJDAR*, 3(1):3–15, Aug. 2000.

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

[7] P. A. Chou. Recognition of equations using a two-dimensional stochastic context-free grammar. In W. A. Pearlman, editor, *Vis. Comm. & Image Proc. IV*, volume 1199 of *Proc. SPIE*, pages 852–863, 1989.

[8] L. Hu and R. Zanibbi. HMM-based recognition of online handwritten mathematical symbols using segmental k-means initialization and a modified pen-up/down feature. In *Proc. ICDAR*, pages 457 –462, Sept. 2011.

[9] S. MacLean and G. Labahn. A new approach for recognizing handwritten mathematics using relational grammars and fuzzy sets. *IJDAR*, pages 1–25, March (avail. online) 2012.

[10] K. Marriott, B. Meyer, and K. D. Wittenburg. A survey of visual language specification and recognition. In *Visual Language Theory*, pages 5–85. Springer, NY, 1998.

[11] N. Matsakis. Recognition of handwritten mathematical expressions. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, May 1999.

[12] E. G. Miller and P. A. Viola. Ambiguity and constraint in mathematical expression recognition. In *Proc. AAAI*, pages 784–791, Madison, WI, July 1998.

[13] H. Mouchere, C. Viard-Gaudin, D. H. Kim, J. H. Kim, and U. Garain. CROHME 2011: Competition on recognition of online handwritten mathematical expressions. In *Proc. ICDAR*, pages 1497 –1500, Sept. 2011.

[14] T. Rhee and J. Kim. Efficient search strategy in structural analysis for handwritten mathematical expression recognition. *Pattern Recognition*, 42(12):3192–3201, 2009.

[15] R. Zanibbi and D. Blostein. Recognition and retrieval of mathematical expressions. *IJDAR*, pages 1–27, Oct. (available online) 2011.

[16] R. Zanibbi, D. Blostein, and J. R. Cordy. Recognizing mathematical expressions using tree transformation. *TPAMI*, 24(11):1455–1467, Nov. 2002.