Ambiguity and Constraint in Mathematical Expression Recognition

Erik G. Miller and **Paul A. Viola**
Massachusetts Institute of Technology
Artificial Intelligence Laboratory
545 Technology Square, Office 707
Cambridge, MA 02139
emiller@ai.mit.edu viola@ai.mit.edu

Abstract

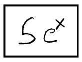
The problem of recognizing mathematical expressions differs significantly from the recognition of standard prose. While in prose significant constraints can be put on the interpretation of a character by the characters immediately preceding and following it, few such simple constraints are present in a mathematical expression. In order to make the problem tractable, effective methods of recognizing mathematical expressions will need to put intelligent constraints on the possible interpretations. The authors present preliminary results on a system for the recognition of both handwritten and typeset mathematical expressions. While previous systems perform character recognition out of context, the current system maintains ambiguity of the characters until context can be used to disambiguate the interpretation. In addition, the system limits the number of potentially valid interpretations by decomposing the expressions into a sequence of compatible convex regions. The system uses A-star to search for the best possible interpretation of an expression. We provide a new lower bound estimate on the cost to goal that improves performance significantly.

Introduction

Handwriting recognition has greatly improved in recent years, in both the handprinting and cursive domains, yielding commercial systems for a wide variety of applications (for example (Yaeger, Lyon, & Webb 1995)). It may appear that the problem of mathematical expression recognition is essentially equivalent to the recognition of standard prose, but there are critical differences which distinguish the two problems and preclude us from applying the standard solutions of handwriting recognition to mathematical expressions.

Mathematical Expression Recognition

A mathematical expression is a binary image in which collections of black pixels represent symbols which are at different scales and positions:

$\boxed{5e^x}$ –OR– 

Our goal is to take a mathematical expression and return a semantic interpretation:

`5 * (e ^ x)`

This representation can be used for document retrieval or an interactive algebraic manipulation system(Fateman & Tokuyasu 1996; Okamoto & Miyazawa 1992; Lee & Wang 1997; Lee & Lee 1994; Lavirotte & Pottier 1997; Martin 1967).

We adopt a generative model approach for the recognition of mathematical expressions. Several authors (Winkler, Fahrner, & Lang 1995; Hull 1996; Chou 1989) have noted that *stochastic context-free grammars* (SCFG's) represent a good generative model for most types of mathematical expressions[1]. SCFG's enable modeling of many types of relationships such as the pairing of parentheses, braces, and brackets, and the association of an integral sign ($\int$) with its differential such as dx. In order to model the spatial layout of symbols in a mathematical expression, the spatial layout rules for various productions (like exponentiation) must be also specified, as in (Hull 1996). Recognition inverts the generative process and produce the appropriate semantics[2].

Context-free grammars have proven to be very useful in the parsing of programming languages (Hopcroft & Ullman 1979). Recently stochastic context free grammars have also been applied successfully to the parsing of natural languages (Charniak 1993). In a programming language the symbols are perfectly unambiguous and linearly ordered. But in an image of a mathematical expression each symbol is ambiguous and there is no natural ordering. As we will see, both ambiguity and ordering are critical issues which, if not handled effectively, will lead to algorithms which are intractable.

In this paper we present a new approach for limiting the number of possible parses. We also present a new method for dealing with character ambiguity so that characters can be interpreted in context rather than

[1] A stochastic context free grammar associates a probability with each production and as a result every valid expression can be assigned a non-zero probability.

[2] Others have used stochastic grammars as models of generic problems in visual recognition (Mjolsness 1990).

in isolation, and so that no probability threshold must be applied in the character recognition process. The search for the most likely interpretation is performed using A-star search (following the work of (Hull 1996)). We propose a new underestimate to the goal that significantly reduces computation time.

Polynomial Time Parsing

The parsing of programming languages is a solved problem. The Cocke-Younger-Kasami (CYK) algorithm (Hopcroft & Ullman 1979), a dynamic programming algorithm, is cubic in the number of characters. The first step of CYK is to build a table (or "chart") with one entry for every sub-sequence of symbols. Each sub-sequence can be indexed by its starting symbol, a number between 1 and N, and its ending symbol, also between 1 and N. There are approximately $\frac{N^2}{2}$ such entries. The algorithm then attempts to find the most likely interpretation for each sub-sequence, an inherently recursive process in which the interpretation of a long sequence is dependent on the interpretation of shorter constituent sub-sequences. The key to computational efficiency is the observation that a particular sub-sequence will be the constituent of many larger sub-sequences, yet the computation need only be done once.

The computational efficiency afforded by the CYK algorithm depends on the existence of a decomposition of the problem into a polynomial number of unique sub-problems. This polynomial nature is a direct result of the linear ordering of text. If such a linear sequence exists we can with impunity limit the set of sub-problems to the set of all consecutive sub-sequences of characters. Mathematical expressions, which are collections of symbols arranged in two dimensions, do not have a straightforward decomposition into a polynomial number of subsets.

Take for example the expression shown in Figure 1. Written out in Matlab syntax, which relies on a linear ordering of symbols, it would be "e ˆ x * (A / B)". In the linear form we can conclude that 'x' is part of the exponential because it is separated from the fraction by a parenthesis. In the graphical form 'x' is equally close to the 'A' and the 'e'. There is almost no local information from which we can conclude attachment. In this case we must rely on contextual information, in the form of a graphical SCFG, to make the attachment. Like CYK parsing, in order to find the best possible parse of a mathematical expression we must enumerate all possible subsets of symbols. Since there is no simple linear constraint we must find some other constraints on the number of allowable subsets. Other authors have dealt with this issue in several ways.

Rectilinear mathematical expressions

Chou (Chou 1989) presents a system in which expressions are represented as rectangular arrays of terminal symbols. He limits the number of potential parses dramatically by requiring that the baselines of characters be within one pixel of their expected location. While this works well for the simulated data presented, it cannot be expected to extend to handwritten input, since the variation in handwritten input is frequently more than a pixel.

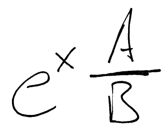


Figure 1: A simple expression which is difficult to decompose into a small number of subsets.

A recognizer with moderate positional flexibility

Winkler et al. (Winkler, Fahrner, & Lang 1995) describe a more flexible system in which two subexpressions can be considered as part of a larger expression as long as the second expression is within certain predefined regions relative to the first expression. However, this system imposes severe limits on the distance which a symbol can be from another symbol while still being part of a common subexpression. We wish to avoid this type of limit, as we believe it will rule out the true answer in some situations. This system also decides on characters out of context, which has problems that are discussed below.

Modeling positions of characters as Gaussian variables

Hull (Hull 1996) comes closer to our desired goal, by allowing more general positioning of terminals and subexpressions relative to each other. The probability that two subexpressions are in a particular relationship relative to each other (e.g. one is the subscript of the other) is defined by a two dimensional Gaussian distribution around the expected position of the second expression. Hence, the further the second expression is from the center of the Gaussian defining that relationship, the lower the probability of that particular operation.

With this more flexible model of character positions, there are no rigid geometric constraints which can be used to automatically prune away the exponential number of symbol subsets. Instead Hull's algorithm attempts to enumerate all possible subsets, and prunes away those that are very unlikely. Because of the potentially exponential number of subsets, Hull uses A-star search to order the enumeration of subsets. Our implementation also uses an implementation of A-star search, explained below.

A-Star Search A-star search is similar to best-first search, but in addition requires that one produce an underestimate of the "distance" to the goal (a full parse) from a given sub-parse. To apply A-star in a proba-

$$\frac{b_2+b_3}{A+B+C} \qquad \frac{b_2+b_3}{A+B+C}$$

Figure 2: An expression where convex hulls are useful for pruning interpretations. Note that the probability that the enclosed characters are in fact the expression A^2 is reasonable. But this subset is eliminated because their convex hull intersects the fraction symbol.

bilistic setting, one conventionally uses the negative log likelihood as the notion of distance. The probability of an interpretation of an image is the product of the unconditional terminal probabilities multiplied by the probabilities which stem from the geometry rules. The unconditional terminal probabilities are an underestimate of the cost to achieve the final goal – an interpretation of the entire image. This underestimate is added to the negative log likelihood of a sub-parse in order to evaluate whether the sub-parse is a good candidate for expansion.

We compute the A-star estimate by accumulating maximum likelihood estimates of the terminals. Hence, the A-star penalty is computed as:

$$\sum_{c \in \mathcal{C}} \min_{t \in \mathcal{T}} \left(-\ln Pr\left(c = t\right)\right)$$

where $\mathcal{C}$ is the set of uninterpreted connected components (terminals) and $\mathcal{T}$ represents all possible characters in the grammar.

Convex Hulls: A new pruning criterion

As a second method for pruning the search space of parses, we attempt to prune away possible subsets based on a simple geometric test: a subset of characters is *not allowed* if the convex hull of the subset contains a character that is not part of the subset[3]. We define the convex hull of the character to be the smallest convex polygon that contains all of the pixels which define the character (for handwritten text the convex hull is the smallest convex polygon which contains all of the strokes defining the character). The convex hull of a set of characters is the convex hull of the union of the convex hulls of each character in the set (see Figure 2).

There are several justifications for this criterion:

- It is consistent with the layout rules for typeset expressions and is not violated for any of our test data. In fact, we are unaware of a typeset expression for which the constraint is violated.
- For linear text this criteria is identical to the constraint that subsets must contain consecutive symbols. Any non-consecutive subset will have a convex hull that contains a character which is not in the set.
- The algorithms necessary for computing with convex hulls are very efficient. A convex hull of a point set can be computed in $O(n \log n)$, where n is the number of points (Cormen, Leiserson, & Rivest 1991). Computing the convex hull union of two convex hulls is $O(m+l)$, where l and m are the number of vertices in the convex hulls. The intersection of two convex hulls can be found in $O(m+l)$ also.

Unfortunately it is possible to construct an artificial arrangement of two dimensional symbols for which this criterion yields no pruning[4]. But, for many two dimensional mathematical expressions there are roughly N^3 allowable subsets.

Furthermore, some valid expressions violate the convex hull rule described above. This could be at least partially remedied by eroding characters a small amount before the convex hull test is performed. The preliminary convex hull results are encouraging (See Figure 8).

Maintaining ambiguity of character identification

In addition to dealing with all of the possible arrangements of symbols and deducing the operations based on relative positions, we must tackle the problem of disambiguating individual characters. As already suggested, ambiguity makes recognition of the whole expression difficult. We categorize the problems caused by ambiguity into three groups.

The first we call the "q-9" problem, and it is illustrated in Figure 3. There are pairs of characters which are, without context, impossible to distinguish. Accurate recognition requires that multiple character hypotheses be maintained until the ambiguity can be resolved from context. There are many examples of confusable characters, even when the characters are written quite carefully. For sloppily handwritten text the overlap among character classes becomes quite large.

The second type of problem caused by ambiguity we call the "threshold problem". This is illustrated in Figure 4. It shows an example of a situation in which the constraints provided by the grammar could allow us to successfully interpret the '+', even though out of context, it would appear extremely unlikely to be a '+'. Any system which discards hypotheses based on a probability threshold runs the risk of missing a correct recognition in such cases.

The third type of problem arising from ambiguity is "interpretation explosion". When multiple hypotheses are maintained for a single character the number of possible parses grows rapidly. For example, the geometric measures of grammar classes such as "exponentiation expression" depend upon the baseline of the base of the exponent. When we maintain ambiguity in the individual characters we must also maintain ambiguity in these

[3]This rigid rule could be converted to a soft constraint in our probabilistic framework.

[4]Take a set of very small symbols arranged uniformly on the circumference of a very large circle. The convex hull of every possible subset of symbols is allowable.

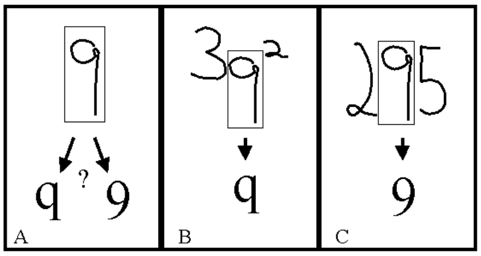


Figure 3: **A.** *The isolated analysis of a character has inherent ambiguities which cannot be resolved without context.* **B.** *Clearly the context here suggests that the character is the letter 'q'.* **C.** *The context in this figure suggests the character is the digit '9'.*

dependent higher level classes. This causes the size of the grammar to grow by a factor which is roughly equal to the number of character classes for which we maintain ambiguity. This can make the grammar impractically large. Below, we discuss a compromise between committing to a character *out of context* (to be avoided) and maintaining all possible character hypotheses during the parsing process (too complex).

Returning for a moment to previous work, we note that Hull performs a pre-processing step that recognizes each of the characters in the expression. Because this recognition takes place before parsing, it does not make use of available contextual information. He is thus vulnerable to the "q-9" problem discussed above.

The following simple analysis demonstrates the necessity of maintaining multiple character hypotheses in developing a robust system. Suppose we have a system for which isolated character recognition is correct with probability R. If an expression consists of t characters, we have probability R^t of correctly recognizing all of the characters without contextual information. With $R = 0.95$ and $t = 20$, we obtain a surprisingly low probability, $0.95^{20} \approx 0.36$, of getting every character right. This is an *upper bound* on the probability we will correctly parse the entire expression (it is possible that the parse might fail for other reasons). Obviously, this upper bound will only shrink if we recognize larger expressions or allow a more varied set of characters.

The system described in (Chou 1989) maintains ambiguity, but in a limited sense. For each character it retains only those interpretations which are above a certain likelihood. His character models are Gaussian; he proposes a threshold at three standard deviations from the mean. While this may seem like a conservative threshold, a similar analysis shows that in a 20 character expression, on average one correct character will be eliminated for more than 5 percent of expressions, since $1 - 99.74^{20} \approx 0.0507$. Also, this scheme may lead to keeping around large number of character hypotheses in handwriting recognition, where the characters have large variations.

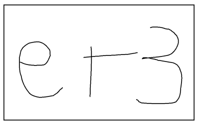


Figure 4: An example of an expression with a very noisy character. The '+' is very difficult to interpret correctly without the context, but with constraints provided by the grammar could potentially be correctly identified.

A Compromise: Maintaining a Hypothesis for Each Character Class Many of the terminals in our grammar are in the same syntactic class. As a result a grammatical analysis can never resolve ambiguity between them. So for example if a character is slightly more likely to be a 'b' than an 'h' there is no syntactic information that can resolve this ambiguity[5]. We can save a lot of effort by simply deciding at the outset that the terminal is a 'b'. On the other hand, as we saw in Figure 4 the ambiguity in the central symbol can be resolved through syntactical analysis. In this case it is important that multiple hypotheses be maintained.

As a middle ground, between maintaining hypotheses for all possible terminal symbols and committing to the single most likely character, we define classes of characters which play the same syntactic role in the grammar. Since the characters in these classes are grammatically equivalent, no constraint from the SCFG can ever be used to disambiguate members of the class. Only the single most likely interpretation for each syntactic class need be maintained.

The concept of syntactic class must be expanded to account for the fact that characters which are traditionally considered syntactically equivalent (like 'p' and 'P') may behave differently with respect to the geometric aspects of the grammar. An example is shown in Figure 5, where the most likely *out of context* interpretation of the leftmost character is *lowercase* 'p'. There is no advantage in maintaining the hypothesis that the character is a 'q'; it has low probability and all of the important spatial properties of the glyph as a 'p' (its baseline, for example) are the same as if it were a 'q'. This is not true for the interpretation as a 'P' however, since the baseline under this interpretation would be higher. And in this case, it turns out that the most likely *context-dependent* interpretation is as a 'P', not as a 'p'.

This leads to *four* terminal classes of *letters* (a single class traditionally) for which we maintain hypotheses throughout the interpretation process. These classes are ascending letters, descending letters, small letters (neither ascending nor descending), and large letters (both ascending and descending). "P", "p", "e", and "Q" represent examples from each respective class. Other authors have used similar classes but not to

[5]Other forms of contextual information might be useful for this sort of problem. We are currently exploring this possibility.

Figure 5: The first character (a "p" or a "P"?) can only be resolved by considering its relationship to the subsequent character.

maintain different classification hypotheses (Winkler, Fahrner, & Lang 1995).

In addition to these separate letter classes which are based strictly on their geometric properties, other classes were defined based on their non-spatial role in the grammar. For example, other syntactic terminal classes include: "zero", non-zero digits, left parentheses, right parentheses, and fractions. Each of these plays a distinctively different role in the grammar of a mathematical expression. In all, a separate hypothesis for each of the 14 different terminal classes is maintained.

A New Approach

Hence, our approach contains three innovations. The first is the geometric convex hull constraint, which limits the growth of the number of possible parses of the expression. The second is a new (but still conservative) A-star completion estimate. The third is the bookkeeping necessary to maintain an hypothesis for each character class, which allows us to interpret individual characters using greater contextual information. We now describe some implementation details of a preliminary system.

Implementation

Certain assumptions were made in order to get a preliminary system working. In the typeset version of the system, it is assumed that all characters are distinct and do not overlap. We make no attempt to deal with scanner distortion. In fact, all of the typeset examples in this paper were generated on-line and are essentially distortion-free.

Overview

The typeset system takes a binary image as input. A simple connected components algorithm is run on the image, generating a list of terminals. A character recognizer based on the Hausdorff distance (Huttenlocher, Klanderman, & Rucklidge 1993) is used to generate probabilities that each connected component or pair of connected components is a particular character. The set of terminals for the initial system is, according to class:

- ascender letters: $b, d, h, i, k, l, t, \delta, A$-$Z$ (except Q),
- descender letters: g, p, q, y, γ,
- small letters: $a, c, e, m, n, o, r, s, u, v, w, x, z, \alpha$,
- ascender/descenders: f, j, Q, β,
- binary operators: $+, -, =$,
- zero: 0,
- non-zero digits: 1-9,
- other symbols (each its own class): $(,), [,], \{, \}$, fraction symbol.

While the grammar is much too large to include here (80 classes, 200 productions), it is worth mentioning the following:

- It is represented in Chomsky Normal Form. This facilitates the coding of the grammar.
- It contains productions supporting exponentiation, subscripting, fractions, "loose" concatenation (multiplication), and "tight" concatenation (the appending of digits in a number).
- The *a priori* probability of each operation is the same. Probabilities of operations are assigned only according to the geometry of the layout and the probability of their sub-parses.

After the program has calculated the probability of each character being in each class, the program starts the dynamic programming process which is closely related to the CYK algorithm mentioned before. The first step in this process is to build the previously mentioned "table" in which there is one entry for every subsequence of characters. This can be done in order from the shorter sequences to the longer sequences, by first computing the probability of each sequence of length 1, then from these computing the probability of each sequence of length 2, and so on.

In our implementation, we limit the size of the parse table using a very conservative restriction on the distance between sub-sequences as an initial filter and the convex hull criterion described previously as a second filter. When we finish building the table, the entry which has the full number of characters and the highest probability represents the best legal parse of the given mathematical expression.

Building the Table

Figure 6 shows how the parse table is generated for a sample mathematical expression show in Figure 7. The table contains one column for each subset of terminals, and one row for each possible interpretation for that subset. The input to the parsing algorithm is a set of connected components. Associated with each component is a probability vector, containing the conditional probability for each possible terminal symbol. As a first step only the most likely terminal from each syntactic class is retained. For example, the most likely interpretation for the leftmost component is as an 'a'. Nevertheless several other hypotheses are maintained: as an ascender it is most likely to be an 'O'; as a descender it is most likely to be a 'q'; as an ascender/descender

	Trmnl 1	Trmnl 2	Trmnl 3	Trmnls 1,2	Trmnls 2,3	Trmnls 1-3
Global properties of the connected component set	Min x, max x, min y, max y, centroid	Min x, max x, min y, max y, centroid	Min x, max x, min y, max y, centroid	Min x, max x, min y, max y, centroid	Min x, max x, min y, max y, centroid	Min x, max x, min y, max y, centroid
As a small letter	Prob: 0.1 Baseline: 47 Pt-size: 12	Prob: 0.05 Baseline: 55 Pt-size: 13	Prob: 0.05 Baseline: 52 Pt-size: 10	Prob: 0.0 Baseline: 0.0 Pt-size: 0.0	Prob: 0.0 Baseline: 0.0 Pt-size: 0.0	Prob: 0.0 Baseline: 0.0 Pt-size: 0.0
As a large letter	Prob: 0.01 Baseline: 45 Pt-size: 8	Prob: 0.02 Baseline: 53 Pt-size: 9	Prob: 0.01 Baseline: 50 Pt-size: 9	Prob: 0.0 Baseline: 0.0 Pt-size: 0.0	Prob: 0.0 Baseline: 0.0 Pt-size: 0.0	Prob: 0.0 Baseline: 0.0 Pt-size: 0.0
As a digit	Prob: 0.03 Baseline: 47 Pt-size: 9	Prob: 0.02 Baseline: 55 Pt-size: 11	Prob: 0.01 Baseline: 52 Pt-size: 11	Prob: 0.0 Baseline: 0.0 Pt-size: 0.0	Prob: 0.0 Baseline: 0.0 Pt-size: 0.0	Prob: 0.0 Baseline: 0.0 Pt-size: 0.0
As an exponentiation	Prob: 0.0 Baseline: 0.0 Pt-size: 0.0	Prob: 0.0 Baseline: 0.0 Pt-size: 0.0	Prob: 0.0 Baseline: 0.0 Pt-size: 0.0	Prob: 0.001 Baseline: 47 Pt-size: 12	Prob: 0.00007 Baseline: 55 Pt-size: 13	Prob: 0.0003 Baseline: 47 Pt-size: 12
As subscript	Prob: 0.0 Baseline: 0.0 Pt-size: 0.0	Prob: 0.0 Baseline: 0.0 Pt-size: 0.0	Prob: 0.0 Baseline: 0.0 Pt-size: 0.0	Prob: 0.00005 Baseline: 47 Pt-size: 12	Prob: 0.002 Baseline: 55 Pt-size: 13	Prob: 0.00000043 Baseline: 47 Pt-size: 12

Figure 6: Part of a parse table used to do dynamic programming.

it is most likely to be a 'Q'[6], and so on. In addition to computing these probabilities, we compute certain interpretation-dependent quantities of the characters such as the baseline and the point size. They are interpretation-dependent since they are different for different rows of the table. Finally, we also compute global properties of the current connected component group. These properties include the minimum and maximum coordinates, the centroid, the convex hull, and so forth. These global properties only need to be computed once per connected component group, since they are invariant to the interpretation of the group.

In addition to the terminal classes there are rows in the table for compound expressions made up of multiple terminals (e.g. exponentiation, fractions, binary operations, etc.) There is zero probability that these expressions can generate an output with a single component.

The initial state of the system is shown in the first three columns of the table. There is one entry for each distinct component in the image. Parsing is a process by which new columns are added to the table, each describing a valid subset of components from the original image. The table is expanded toward the right, as large collections of components are enumerated and evaluated.

There are three possible two-character combinations

[6]This is a very unlikely hypothesis, but it is the best in that class.

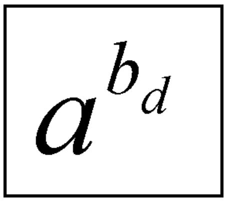


Figure 7: The sample mathematical expression used in the description of table building. Notice that no entry was made in the table for the two-character combination of characters "a" and "d". This is because the a-d pair do not fit the convex hull criterion.

of the three characters. However, if the two characters fail the convex hull test described above, as with the "a" and the "d" in our example, then an entry in the table will not be made. Since the two component groups cannot be interpreted as a single terminal, these rows are flagged as impossible. As new columns are added to the table, the interpretation-dependent and global properties of the group are computed.

The final column in the table shows probabilities of interpretations of the entire expression. Column 6 shows that the correct interpretation (as an exponentiation at the highest level) has the highest probability, so in this case the procedure worked properly.

Experiments

To verify the basic functionality of the system, we performed a simple series of tests. For each letter x in the system, the following expressions were used as inputs to the system: x_x, x^x, x^{x_x}, $\frac{x}{x}$. Where allowed by the grammar, the digits were also used in these tests. All of these test images were evaluated correctly by the system.

We attempted to evaluate the effectiveness both of the the convex hull pruning constraint and A-star search using our underestimate of the distance to goal. Without using either type of heuristic, parse times were very long – on the order of minutes. Using both criteria parse times are in the seconds. In order to quantify this we created a series of expressions of varying complexity from 3 terminals to 20 terminals. On the left of Figure 8 is the performance of the A-star system with and without the convex hull constraint. On the right is the performance using the convex hull constraint with and without the A-star heuristic[7] . There is significant improvement in performance using either of these heuristics, and even greater improvement when both are used. We collected data from the system running without either heuristic, but this curve quickly exists the top of the graph.

It is interesting to note how the system performed on "illegal" images, for example the expression 0^{0_0}. This input is considered illegal since our grammar provided no mechanism for subscripting a number. As a result the system recognized the input as the expression "600". This can be understood as follows: since the system could not generate 0_0, it apparently concluded that the last two zeroes were effectively the same size. However, the output 0^{00} would also be illegal since a two digit number cannot begin with a 0. The system's fix was to consider all three digits as being approximately the same size, and then using a much less likely, but legal interpretation of the first digit, it settled on "600" as the best interpretation. This example illustrates a good deal about the system's flexibility in generating reasonable answers in tough situations.

Anecdotal examples of expressions which the system successfully parsed are given in Figure 9. The actual time spent parsing these expressions on a 266MHz Pentium Pro was approximately 4, 0.2, 16, and 25 seconds. This does not include the time to generate character hypotheses for the connected components. These examples are not difficult, but they validate the basic method and show that it is feasible to do the extra bookkeeping which we have incorporated in this system. The only failures in the current typeset system were due to mis-recognition of characters.

[7]The points on the two graphs are the same set of expressions, the graph on the left contains fewer points because the computation time for A-star without convex hull pruning grows far too rapidly for us to measure. We simply gave up after many minutes of run time.

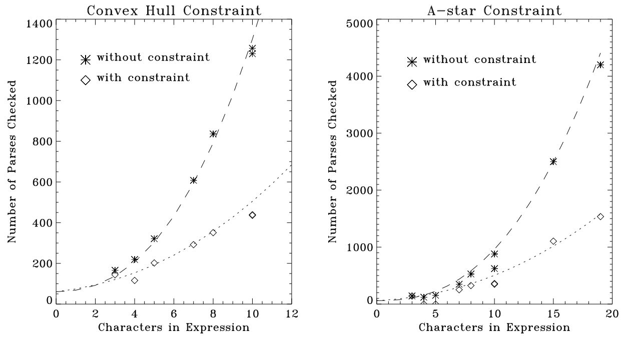


Figure 8: The left graph shows the pruning of the search space by the convex hull criterion. The right graph shows the performance gained by adding the A-star constraint. The algorithm reduces to best-first search without the A-star constraint.

$$((t+m)+wt)$$

$$b_f p_a$$

$$k_2^9 + \frac{\frac{3}{2}+o^5}{e_6+z_7}$$

$$\frac{123}{456^7 + \frac{\frac{2}{4}}{a+b+c_d}}$$

Figure 9: Some of the expressions successfully recognized by the system.

Future Work

A major focus in developing this system was to prepare for the migration to handwritten mathematical expressions. Our preliminary work with handwritten expressions is illustrated in Figure 10. We show three examples, the first of which was parsed correctly, the second of which contains a geometry-based error, and the third of which contains a character-identification error. We hope that by improving the character recognizer and learning the parameters of our geometry models that we can significantly improve the performance of the system in the future. While the accuracy of the current system needs great improvement, we feel we have laid the groundwork for a practical system's implementation.

Acknowledgments

We would like to thank Tom Rikert and Nicholas Matsakis for contributions and discussions related to this work.

References

Charniak, E. 1993. *Statistical Language Learning.* Cambridge, MA: MIT Press.

Chou, P. A. 1989. Recognition of equations using a two-dimensional stochastic context-free grammar. In

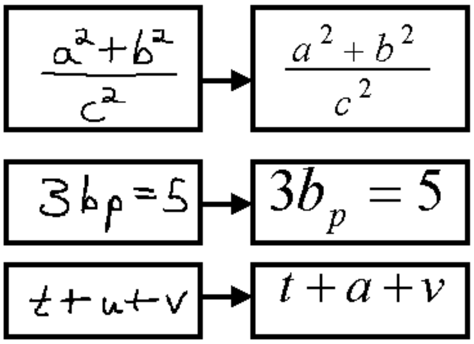


Figure 10: Some examples of handwriting and their parses.

SPIE Vol. 1199 Visual Communications and Image Processing, SPIE, 852–863. Murray Hill, NJ: SPIE.

Cormen, T. H.; Leiserson, C. E.; and Rivest, R. L. 1991. *Introduction to Algorithms*. Cambridge, Mass.: MIT Press.

Fateman, R. J., and Tokuyasu, T. 1996. Progress in recognizing typeset mathematics. In *Document Recognition III, SPIE Volume 2660*, 37–50. Murray Hill, NJ: SPIE.

Hopcroft, J. E., and Ullman, J. D. 1979. *Introduction to Automata Theory, Languages, and Computation*. Reading, Mass.: Addison-Wesley.

Hull, J. F. 1996. Recognition of mathematics using a two-dimensional trainable context-free grammar. Master's thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science.

Huttenlocher, D. P.; Klanderman, G. A.; and Rucklidge, W. J. 1993. Comparing images using the hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 15(9):850–863.

Lavirotte, S., and Pottier, L. 1997. Optical formula recognition. In *Fourth International Conference on Document Analysis and Recognition*, ICDAR, 357–361. Ulm, Germany: IEEE.

Lee, H.-J., and Lee, M.-C. 1994. Understanding mathematical expressions using procedure-oriented transformation. *Pattern Recognition* 27(3):447—457.

Lee, H.-J., and Wang, J.-S. 1997. Design of a mathematical expression understanding system. *Pattern Recognition Letters* 18:289–298.

Martin, W. A. 1967. A fast parsing scheme for hand-printed mathematical expressions. MIT AI Project Memo 145, Massachusetts Institute of Technology, Computer Science Department, MIT, Cambridge, MA.

Mjolsness, E. 1990. Bayesian inference on visual grammars by neural nets that optimize. Technical Report 854, Yale University, Department of Computer Science, Computer Science Department, Yale University, New Haven, CT.

Okamoto, M., and Miyazawa, A. 1992. An experimental implementation of a document recognition system for papers containing mathematical expressions. In Baird, H. S.; Bunke, H.; and Yamamoto, K., eds., *Structured Document Image Analysis*. Berlin: Springer-Verlag. 36–63.

Winkler, H. J.; Fahrner, H.; and Lang, M. 1995. A soft-decision approach for structural analysis of handwritten mathematical expressions. In *International Conference on Acoustics, Speech, and Signal Processing*.

Yaeger, L.; Lyon, R.; and Webb, B. 1995. Effective training of a neural network character classifier for word recognition. In Mozer, M.; Jordan, M.; and Petsche, T., eds., *Advances in Neural Information Processing*, volume 9. Denver 1996: MIT Press, Cambridge.