EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling (2007)
M. van de Panne, E. Saund (Editors)

Managing Ambiguity in Mathematical Matrices

David Tausky, George Labahn, Edward Lank and Mirette Marzouk

David R. Cheriton School of Computer Science
University of Waterloo, Waterloo, ON, Canada, N2L 3G1

Abstract

In this paper we describe strategies for recognizing and using hand drawn matrices in a pen math system. This includes a new technique to recognize common short-forms of writing matrices using ellipsis (...). Ellipsis are commonly used in sketched matrices to illustrate the structure of a matrix without fully specifying the matrix. A second contribution of this paper is a new method to estimate the parameters of the hand drawn matrix, such as the number and position of the rows and columns. This is done using a modified clustering algorithm, allowing one to reduce the number of hard-coded constraints.

Categories and Subject Descriptors (according to ACM CCS): I.5.4 [Computing Methodologies: Pattern Recognition]: Applications

1. Introduction

Pen-math systems allow users to follow a familiar pen-and-paper paradigm for working with mathematics. Computer algebra systems such as Maple or Mathematica do not follow the familiar pen-and-paper interface but do provide rich environments for actually doing mathematics : manipulations, solving equations and so on. We believe that a complete pen-math system should attempt to combine the ease of use of pen-math environments with the power of computer algebra systems (in this case either Maple or Mathematica).

Once one is able to input and do mathematics in a pen-math system, then it is natural to also want to use all the typical conveniences that are commonly used when inputting mathematics with pen-and-paper. For example, users often will input an expression of the form

$$1+\cdots+100$$

with the understanding that this represents the sum of the first 100 nonzero digits. Any pen-based math system that hopes to provide the same level of ease of use will need to understand and work with short-cuts in expressions. Making use of convenient, well accepted short-cuts is particularly important when one works with matrix algebra. While a user can be taught that the sum of the first 100 nonzero digits should be input as

$$\sum_{i=1}^{100} i$$

a similar statement is not true if the user wants to work with matrices that are normally written as

$$A=\begin{bmatrix} 1 & x_0 & \cdots & x_0^5 \\ 1 & x_1 & \cdots & x_1^5 \\ \vdots & \vdots & & \vdots \\ 1 & x_5 & \cdots & x_5^5 \end{bmatrix}$$

which is a short form for a Vandermonde system or

$$B=\begin{bmatrix} 1 & 0 & \cdots & -8 \\ 2 & 1 & \cdots & -7 \\ \vdots & \vdots & & \vdots \\ 10 & 9 & \cdots & 1 \end{bmatrix}.$$

In addition to inputting and recognizing such expressions, a system also needs to correctly interpret such matrices and allow for further computation. For example one could have the computational engine compute the factored form of the determinant of A to obtain

$$(x_0-x_1)(x_0-x_2)(x_0-x_3)(x_0-x_4)(x_0-x_5)$$

$$(x_1-x_2)(x_1-x_3)(x_1-x_4)(x_1-x_5)(x_2-x_3)$$

$$(x_2-x_4)(x_2-x_5)(x_3-x_4)(x_3-x_5)(x_4-x_5)$$

verifying a well-known identity, or else finding that the eigenvalues of B are given by

$$0,0,0,0,0,0,5+20\sqrt{2}i,5-20\sqrt{2}i,$$

something which is not easily known in advance.

This paper describes a number of techniques to recognize matrices, with an emphasis on recognizing matrices similar to those a user would draw, with short cuts to represent the structure of the matrix. We propose multiple techniques based on both local features and global optimizations. Finally we address a few of the unresolved problems in matrix recognition such as the limitations of the current MathML standard.

This paper is organized as follows: Section 3 reviews relevant research in the field of matrix and table recognition, and provides a brief summary of the MathBursh environment platform that we use as a test bed to evaluate our recognition techniques. Section 4 provides an overview of our method to recognize matrices. Sections 5 and 6 discusses alternative techniques to those used in section 4, with an emphasis of increasing the robustness by finding a globally optimum solution.

2. Definitions

A *fully specified* matrix is a matrix where each and every element of the matrix has a symbolic or numeric value and each value is stated. Figure 1 shows an example of a fully specified matrix.

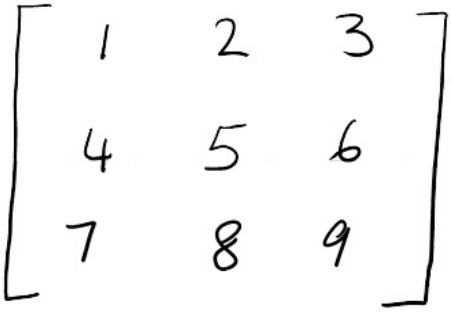


Figure 1: *A fully specified matrix.*

An *under-specified matrix* or *abstract matrix* refers to a matrix where each element is not specified, but can be inferred by the structure of the matrix as indicated by the presence of ellipsis (a sequences of dots) and the elements which do exist [SS06]. In the case of under-specified matrices, there are three subclasses that we consider: those matrices where the dimension of the matrix can be inferred directly, those where the dimension or indexing cannot be directly inferred, and those where the indexing is ambiguous. Figure 2 shows two examples of an under-specified matrix. The dimension of the left matrix of figure 2 can be inferred whereas the right matrix is an under-specified matrix where the dimension cannot be.

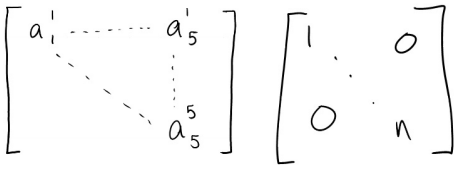


Figure 2: *Two under-specified matrices. The dimensions of the left matrix can be inferred. The dimensions of the right matrix cannot be.*

Finally an *ambiguous matrix* is a matrix where there are multiple interpretations or conflicting interpretations of the dimension or indexing, and no clear heuristic for resolving the ambiguity exists. Figure 3 shows an example of ambiguous matrix. At first glance, the matrix appears consistent; however, the second row will resolve to have the form a_2 a_3 a_4 which is not consistent with the 5x5 dimension of the matrix.

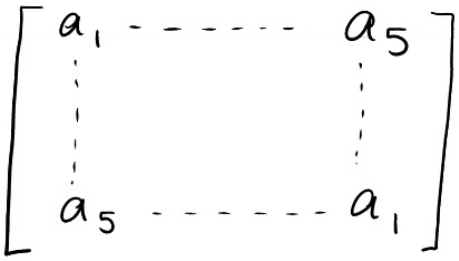


Figure 3: *An ambiguous matrix*

3. Background

Some relevant research has already been done on recognizing matrices and interpreting short forms. However to our knowledge no one has integrated matrix recognition and under-specified matrix expansions into a hand-drawn mathematical system.

Zanibbi, Blostein and Cordy [ZBC04] present a comprehensive survey of table recognition techniques in general. Matrices can be viewed as a specific class of tables. Most relevant work in table recognition literature operates on scanned images of typeset data, therefore the techniques differ from those presented here. Chandran and Kasturi [CK93]

and Zuyev [Zuy97] present systems to recognize typeset tables, where cell segmentation is performed by horizontal and vertical profiling, which has a similar premise to the element segmentation performed in section 4.2. However, their system only works with off-line, typeset tables. Hu et. al [HKLW01] present a graph based system which has some similarities to the graph based representation used in our system, but do not address the issue of recognition or structure extraction. Hu et. al. also describe details of a graph matching algorithm which can be used to verify the recognition of a table against ground truth data. Specific to recognizing matrices Kanahori and Suzuki [KS01, KS03] are the only researchers to have developed a system to recognize type-set scanned matrices as part of their math OCR package, InftyReader.

A general discussion of the problems of matrix recognition is discussed in Blostein and Grbavec [BG96], along with a discussion regarding the difficulties of recognizing mathematical notation in general.

In an alternate direction, Sexton and Sorge [SS06] present an algorithm for interpreting under-specified matrices and expanding them into fully specified matrices, using linear interpolation. Their research is devoted to the interpretation, not recognition, of under-specified matrices. Their research assumes the under-specified matrix is entered in a very specific format using a keyboard. However their research is the basis for our anti-unification method described in section 4.4.

3.1. MathBrush Overview

Our matrix recognition technique is currently integrated into the MathBrush interactive pen-based math system. For a more through discussion of MathBrush please refer to [LMM*06].

MathBrush is an experimental pen-based math system having as the goal of allowing users to perform complex mathematical tasks on a Tablet PC platform. The mathematical tasks are performed via interaction with one or more computer algebra systems, in this case Maple or Mathematica. MathBrush takes as its starting point that doing mathematics with a pen-based device requires much more than simply being able to input handwritten mathematical expressions and having the ability to do a few mathematical operations. Rather a mathematical pen-based math system needs to worry about such matters as what to do with the vast (and complicated) functionality of existing mathematical systems and how to deal with the huge and unwieldy results that can come from such systems (answers that do not really appear when working with ordinary pen and paper), to name just two nontrivial issues. At the same time users need the functionality to do mathematics (often even for elementary mathematics) and have no choice but to work with whatever answers are returned for their computations (large or small).

Of course there has already been considerable research on systems that recognize handwritten mathematical input. In addition, some of these math recognition systems link to existing mathematical systems so such linkage is by no menas new. We mention MathPad (which links to Maple via the symbolic toolbox of MatLab) and MathJournal (which has its own math system) as two examples where pen-based systems work with mathematical engines. However these systems either have very limited mathematical capabilities or else only allow a few simple interactions with the math system. In either case one can do only a very small amount of mathematics, certainly much less than one could using the various text and window based interfaces of existing computer algebra systems. This situation mirrors events that occurred when computer algebra systems were first gaining prominence some 20 years ago. Namely at that time a number of interesting interfaces for mathematical systems were proposed, mostly for MacIntosh computers. Examples include Theorist, Milo and others [KS98]. For the most part the limited capabilities of these systems relegated them as being math assistants rather than complete math systems.

Beyond the basic set of features necessary to support transformation of handwritten expressions and interactions with computer algebra systems, MathBrush has a number of additional properties useful for mathematical analysis. For example, there is support for input and output of large expressions, interactive manipulation of mathematical expressions via the use of context menus, a logging mechanism for capturing and archiving of problem solving rationale and the ability to interactively swap one computer algebra system for another. The latter feature allows users to find alternative answers for problems - e.g. computer algebra systems often produce different answers for the same problems (e.g. integrals or solutions of differential equations). In many cases results are easier in one system than the other.

The MathBrush system itself is build in five main modules depicted in figure (4). The *interface module* passes the ink collected from a user to the *character recognizer*. The recognizer detects different characters and generates a set of bounding boxes along with a set of candidate and confidence pairs and passes it back to the *interface module*. The *interface module* then passes the information generated by the *character recognizer* to the *structural analyzer*. The analyzer processes this information, constructs a well-formed mathematical expression, and generates a MathML representation of the expression. The *interface module* send the MathML together with the operation selected by the user to the *CAS interface tool*. This tool is used to interact with the target computer algebra system and returns back the computed results generated from the CAS as presentation MathML. The presentation MathML and the format defined by the user are sent to a *MathML rendering tool*, which generates a set of boxes and characters for the *interface module* to display. All of the modules are explained in more details in [LMM*06].

We use MathBrush as a test bed for studying and evaluating our techniques.

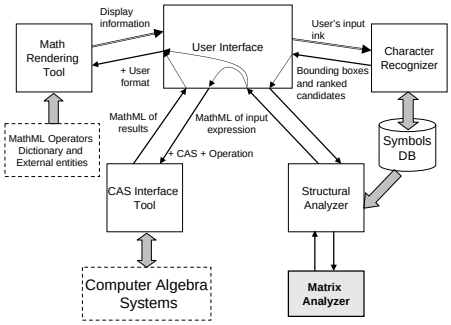


Figure 4: *MathBrush System Components*

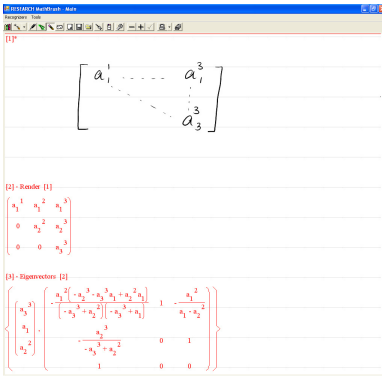


Figure 5: *MathBrush Matrix Recognition Results*

In the context of this paper, we are primarily interested in the matrix analyzer, which is a separate extension of the structural analyzer. Therefore, the input to our matrix analyzer is the character recognizers list of recognized characters and bounding boxes. The matrix analyzer then outputs a recognized, MathML based representation of the matrix, which is presented to the user, using the MathBursh user interface.

4. Matrix Recognition Pipeline

Recognizing sketched matrices is a hard problem. Even if the matrix is fully specified, the individual elements can be complex, vary in size from each other, and contain a different number of characters. The spacing between the characters is not uniform. Recognizing under-specified matrices is a harder problem, as fewer clues about the layout of the matrix exist.

In this section we will describe in detail each stage of our recognition pipeline. Throughout this section we will examine the recognition of the first example presented in the left side of figure 2. We will also explain our rationale for choosing the approach we did, and offer our insight into other strategies or assumptions.

4.1. Pre-Element Processing

The first stage of the pipeline simply removes the square brackets, and separates the ellipsis from the other elements of the matrix. The matrix is first recognized without the ellipsis and then refined using the information contained in the ellipsis. Figure 6 shows the result of this stage. The elements on the left (a) are processed in section 4.2. The dots (b) are processed in section 4.3.

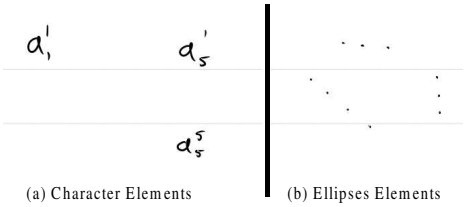


Figure 6: *The matrix after pre-element processing.*

4.2. Element Processing

This stage of the pipeline determines the layout of the drawn elements of the matrix. The operations performed in this stage consist of grouping individual characters into elements and estimating dimensions of the matrix. Figure 7 illustrates the operations. When the ellipses are reintroduced in section 4.4 the size of the matrix will be increased.

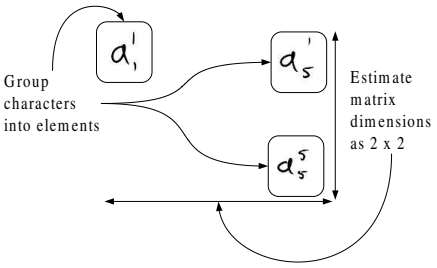


Figure 7: *The example matrix after element grouping*

Unlike typeset matrices, sketched matrices have variable character sizes which cannot be rigidly modeled, and variable spacing between the characters and the elements of the matrix. In this paper we present two methods to group characters. Our first method, based on only local features, is described below. We are currently experimenting with the second method that is based on global features and it is described in section 5.1.

In this method we examine the distance between adjacent characters, and compare it to a soft threshold based on the average width of the characters. Using this threshold, characters are group into elements. This approach is very similar to the approach by Kanahori and Suzuki [KS03] used to recognize typeset matrices. The main difference is that we use a single soft threshold based on the size of the characters whereas Kanahori and Suzuki use multiple hard thresholds, based on the class of the characters being compared.

Once every character has been grouped into an element, each element is then passed back to the structural analyzer, which returns a MathML based expression tree for the element, as each element is essentially a self-contained expression. Each element is then classified as being *indexable* if it is a simple element of the form a, a_c^b or $a_{c,d}^b$.

At the completion of this phase of the pipeline, each element is a fully parsed expression, and the structure of the matrix has been resolved. If the matrix is a fully specified matrix, the recognition phase is essentially complete.

4.3. Ellipsis Processing

Whereas the element processing stage groups characters into elements, the ellipsis processing stage links individually recognized dots into ellipsis. Ellipses are defined as a set of co-linear dots aligned in a horizontal, vertical, diagonal or anti-diagonal direction.

Unlike typeset matrices, the dots that form an ellipsis in a sketched matrix may be irregularly spaced, and drawn with a significant deviation from an ideal line between the intended end points. Consider figure 8 showing the grouped dots. In this paper we present two methods for detecting ellipses, one based on local features, and an alternative technique is presented in section 6.2 that finds a global optimum. In our current implementation only local features, ellipses are detected by searching for groups of dots that may be co-linear within a hard-coded tolerance, and are close in proximity. The tolerance was obtained by statistical evaluation of our data set. Once we have detected ellipses, we search for elements that are co-linear with the ellipsis and link the elements together as described in the next section.

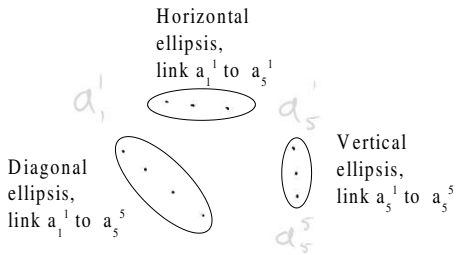
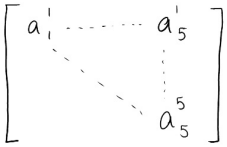


Figure 8: *Example of a matrix with three detected ellipses.*

4.4. Matrix Anti-Unification and Consecration

For hand-drawn matrices, it is frequently the case that user will under-specify the matrix. There are a number of reasons for this; to save time, the matrix may be too large to be completely drawn, or the user transferred the paper paradigm to the pen math system. In this step we resolve the under-specified matrices into fully specified matrices. The algorithms used here are variants of the algorithms by Sexton and Sorge [SS06, SS05]. In comparison, we use a graph based representation of the matrix rather than text based grammars. We only perform linear interpolation between integer values.

The steps in this stage consist of identifying the structure of ellipsis regions (anti-unification), interpolating these regions, and checking for consistency. The three types of regions which we detect are linear, triangular and rectangular regions. Once regions are detected we first interpolate across the boundary ellipses, then interpolate within the region. This is shown in figure 9. If a conflict occurs during the interpolation, an error is reported and the recognition process is terminated.



(a) The initial matrix.

$$\begin{bmatrix} a_1^1 & a_2^1 & a_3^1 & a_4^1 & a_5^1 \\ & a_2^2 & \cdots & \cdots & a_5^2 \\ & & a_3^3 & \cdots & a_5^3 \\ & & & a_4^4 & a_5^4 \\ & & & & a_5^5 \end{bmatrix}$$

(b) The matrix after performing linear interpolation over the boundaries.

$$\begin{bmatrix} a_1^1 & a_2^1 & a_3^1 & a_4^1 & a_5^1 \\ & a_2^2 & a_3^2 & a_4^2 & a_5^2 \\ & & a_3^3 & a_4^3 & a_5^3 \\ & & & a_4^4 & a_5^4 \\ & & & & a_5^5 \end{bmatrix}$$

(c) The fully instantiated matrix after resolving the newly added horizontal ellipses.

Figure 9: *Interpolation of a triangular region*

4.5. Rendering output

Presentation MathML is then generated by transforming the graph structure from the pervious stage into a tree structure and traversing the tree. The math rendering tool in MathBrush accepts presentation MathML and generates a set of

boxes for the user interface module to display. Currently the Presentation MathML of the fully-specified Matrix is generated by the structural analyzer. The rendering tool renders the matrix elements taking care of alignments in the rows and columns, so that the height of a row is the same as of the highest element in it and all the elements are centered vertically in there, and same for the column width. If the matrix width exceeds the width of the display area, the matrix will be broken by keeping the big square brackets to indicate matrix breaking and separating the elements of each row by commas and include them in square brackets. Rows are also separated by commas.

5. Current Work on Methods to Improve Robustness

5.1. Applying clustering techniques to matrix structure recognition

In section 4.2 we described a simple metric for assigning characters to elements and determining the position of the elements using local features. We are currently investigating a more robust and flexible approach to element segmentation using the clustering techniques based on the EM algorithm [FP03].

A characteristic of a matrix is that each element of the matrix is part of a column and a row, and that the mid-point of each element in a particular row or column should be in a line parallel to one of the two primary axes. Therefore the problem can be reduced to two one-dimensional clustering problems, where in each dimension we project the centroid of each element onto a horizontal and vertical axis, and find clusters in the projections. The advantage of this dimensionality reduction is that we reduce the complexity of the problem and solution by exploiting the structure of the problem. Figure 10 illustrates this dimensionality reduction. This projection approach is already used by Hu [HKLW01] and Chandran [CK93] to segment scanned typeset tables, however once the profiling is done they use normalized thresholds to segment the table. We have implemented a more robust and adaptive method:

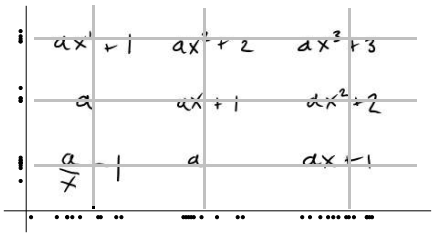


Figure 10: *Projection of character centroids on the axes are show as dots. The thick lines indicate the center of the matrix elements*

We use mixture models to solve the clustering problem, where k Gaussian distributions are initially uniformly distributed over the projected axis. Using an EM-algorithm based formulation, the model parameters we wish to optimize are μ_i and π_i, the mean of each Gaussian model and the mixing proportions. The variance σ_i is fixed to a constant ratio based on the absolute size of the matrix. The optimization can be formulated as finding the μ that optimizes the probability of the characters being generated from that distribution. In the E (or expectation) step, we calculate the probability that the character centroids were generated by the current Gaussian distribution. In the M (or Maximization step) we use a weighted least squares metric to optimize the Gaussian means, μ_i. This process of assigning ownership of characters to each distribution, calculating the probability, and then shifting the distribution based on the points associated with that distribution continues until the algorithm converges.

The method presented in section 4.2 uses only local features, the distance between one character and an adjacent character, to determine the number and position of the elements of a matrix. Therefore, the expected dimension of the matrix is not useful. In contrast, this clustering method uses global features to solve the same problem. Therefore this method works very well, if the expected dimensions of the matrix are known which in turn dictates how many Gaussian models to use. If the expected dimension of the matrix is unknown, we are currently exploring other methods of estimating the parameters.

One method is to overestimate the number of Gaussian models. Typically, if the number of models exceeds the actual number of rows or columns, one of two detectable events will occur. Either one (or more) mixture model has a very low mixing proportion ($\pi_i < \varepsilon$), or two mixture models, with very similar means, will have approximately equal mixing proportions ($|\mu_i - \mu_k| < \varepsilon$ and $\pi_i \simeq \pi_k$). We can then reduce the number of models until we no longer detect these conditions.

This mixture model technique has many advantages over traditional heuristic methods when dealing with pen data. Users, when writing matrices often significantly vary the size and spacing of the characters as they attempt to visually fit the characters within the elements perceived bounds as shown in figure 11. By finding a global optimum we are overcoming the brittleness of heuristic methods that use only local information.

6. Future Work

6.1. Incorporating additional information

There are several classes of matrices which could be resolved if extra information is provided beyond what is commonly drawn. For example if the dimension of the matrix is known, either given explicitly, or by analysis of the

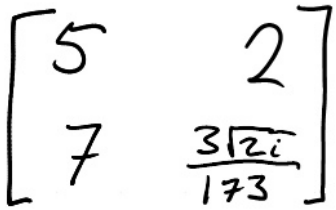


Figure 11: *A matrix with varying character sizes*

surrounding information, then the following under-specified matrices could be resolved.

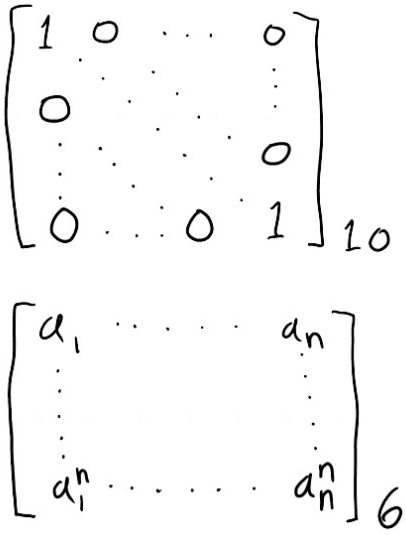


Figure 12: *Two matrices where the dimension cannot be inferred directly, but is indicated as a subscript following the matrix.*

Previous techniques for resolving ambiguity assume that the dimension of the matrix is known prior to analysis. We are currently exploring using pen-based input to allow the user to explicitly indicate the dimensionality as shown in figure 12.

Another example where external information would supplement the recognition process is if each element is more complex than a simple base-subscript-superscript configuration, it may be difficult for the recognition algorithm to infer which variables should be incorporated into generating an index function. Allowing the user to explicitly indicate which variables should be indexed should increase the versatility of our method.

6.2. Ellipsis estimation using clustering techniques

In section 5.1 we discussed a strategy we are currently implementing to improve the robustness of term clustering. A similar but non-Gaussian clustering technique could be applied to the detection and estimation of ellipses. A reasonable approach would be to modify k-mean line fitting algorithm [FP03]:

Hypothesize k lines, for some small k, possibly 1.
Assign points to each line, fitting the line using least squares.
Constrain the line be either vertical, horizontal, diagonal or anti-diagonal.
while log likelihood no longer decreases **do**
  Allocate each point to the dot line.
  Refit the line (translate or rotate in increments of 45°.)
  Calculate perpendicular distance of dots to the nearest line (log likelihood).
end while
If log likelihood is sufficiently large increase k

If the input was further constrained such that each ellipsis contains only 3 (or some other fixed number) of dots, then k could be directly inferred.

6.3. Fill Patterns

Another common technique for indicating the structure of matrices is that of drawing a single concrete term to represent filling a region with that term. We intend to add an additional stage to the pipeline after the anti-unification stage to process fill term. However it is unclear how to intuitively distinguish a fill term from a non-fill term. Sexton and Sorge [SS06] provide algorithms to accommodate fill terms, but in their work, fill terms are explicitly labeled in the input. Currently, a fill pattern of 0 for unspecified regions of the matrix is assumed.

6.4. User Interface Issues

One of the problems we are still investigating is how to represent very large matrices efficiently and naturally, giving the user the option to view all or part of them for better manipulation. Presentation MathML does not support specifying short forms for mathematical expressions. Given the fact that we are using presentation MathML to communicate between modules and with the CAS, we have to generate MathML for the fully specified matrix which could be inefficient for large matrices. Furthermore, allowing the user to view the matrix only in its fully specified form might not be the most natural way one thinks of the matrix. We are looking at ways to efficiently represent and display the short-cuts in matrices and provide the capability to zoom in at certain areas without the need to view the whole complete matrix. This may involve extending the MathML syntax. One other issue we are investigating is matrix editing to operate on certain elements, rows or columns for manipulation.

Consider for example the following matrix shown in figure 13. Although it can be drawn very compactly, the fully-instantiated version would contain over 100 million elements, and the size of the resulting MathML string would be over 2GB.

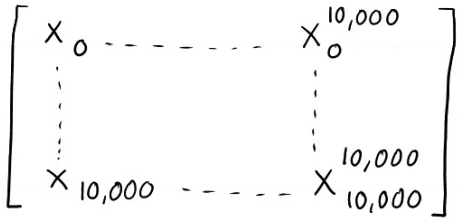


Figure 13: *A sketch of a matrix. Fully instantiated, the matrix would contain 100 020 001 elements.*

Extending the MathML syntax to include underspecified matrices and equations has other benefits. If the user was to enter:

$$x = 1 \cdots n$$

and later enter $n = 10$, using the strict MathML definition we would not be able to represent x, and therefore could not instantiate x even when n is known.

7. Conclusion

In this paper we have described the problem of recognizing hand drawn matrices, and the sub-problem of recognizing matrices with ellipsis drawn to illustrate the structure of the matrix. We then presented an approach to recognizing such hand-drawn matrices. Finally we presented several alternatives that attempt to optimize the recognition over the complete matrix. We believe that recognizing matrices is a critical component of any pen-math system, that also illustrates the benefits of pen-based interaction.

References

[BG96] BLOSTEIN D., GRBAVEC A.: *Recognition of Mathematical Notation*, vol. Handbook of Optical Character Recognition and Document Image Analysis. World Scientific, 1996, pp. 557–582.

[CK93] CHANDRAN S., KASTURI R.: Structural recognition of tabulated data. In *Proceedings of the Second International Conference on Document Analysis and Recognition* (1993), pp. 516–519.

[FP03] FORSYTH D., PONCE J.: *Computer Vision: A Modern Approach*. Prentice Hall, Upper Saddle River, New Jersey, 2003.

[HKLW01] HU J., KASHI R., LOPRESTA D., WILFONG G.: Table recognition and its evaluation. In *Proceedings Document Recognition and Retrieval VIII* (2001), vol. 4307, pp. 44–55.

[KS98] KAJLER N., SOIFFER N.: A survey of user interfaces for computer algebra systems. *Journal of Symbolic Computation 25*, 2 (1998), 127–159.

[KS01] KANAHORI T., SUZUKI M.: A recognition method of matrices by using variable block pattern elements generating rectangular area. In *GREC 2001* (September 2001), Springer, pp. 320–329.

[KS03] KANAHORI T., SUZUKI M.: Detection of matrices and segmentation of matrix elements in scanned images of scientific documents. In *Proceedings of the Seventh International Conference on Document Analysis and Recognition* (2003), pp. 433–437.

[LMM*06] LABAHN G., MACLEAN S., MARZOUK M., RUTHERFORD I., TAUSKY D.: A preliminary report on the mathbrush pen-math system. In *Maple Conference 2006 Proceedings* (July 2006), pp. 162–178.

[SS05] SEXTON A., SORGE V.: Processing textbook-style matrices. In *Mathematical Knowledge Management, 4th International Conference* (2005), pp. 111–125.

[SS06] SEXTON A., SORGE V.: Abstract matrices in symbolic computation. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC)* (2006), Association for Computing Machinery, pp. 318–325.

[ZBC04] ZANIBBI R., BLOSTEIN D., CORDY J. R.: A survey of table recognition. *International Journal on Document Analysis and Recognition 7*, 1 (2004), 1–16.

[Zuy97] ZUYEV K.: Table image segmentation. In *International Conference on Document Analysis and Recognition* (1997), pp. 705–708.