EUROGRAPHICS Symposium on Sketch-Based Interfaces and Modeling (2009), pp. 1–9
C. Grimm and J. J. LaViola Jr. (Editors)

Tools for the efficient generation of hand-drawn corpora based on context-free grammars

Abstract

In this paper, a novel technique is presented for creating unbiased training and testing sets based on performing random walks using a context free grammar. We describe a study conducted to generate a corpus of over 4500 hand-drawn mathematical expressions from 20 subjects. Finally, details are given for an automated system which generates high quality ground-truth data from the hand-drawn expressions. The techniques presented in this paper are illustrated through the creation of a ground-truthed corpus of mathematical expressions, but they are applicable to any sketching domain that can be described by a formal grammar.

Categories and Subject Descriptors (according to ACM CCS): I.5.5 [Computing Methodologies]: Pattern Recognition—Implementation

1. Introduction

One challenge faced in the implementation and evaluation of sketch recognition systems is access to realistic corpora of ground-truthed input. Corpora are valuable in two ways. First, they allow researchers creating recognition systems to train their recognizers, thus improving recognition accuracy. Second, they provide an independent data set against which to test recognizer accuracy. To be useful in training and testing, any corpus must have two attributes. It must be accurately ground-truthed, both at the semantic level and at the level of individual symbols and relations between symbols. It must also be sufficiently large and varied to allow training and testing on realistic input from varied drawers.

One simple approach to creating a sufficiently large corpus is to manually select a set of representative sketches from a domain of interest, have several participants transcribe these sketches, and then manually ground-truth each sketch individually. This process suffers from several drawbacks. These drawbacks include the possibility of biased or narrow coverage and tedium resulting from creating unique sketches or from ground-truthing a large set of sketches. We expand on these drawbacks here.

The process of selecting sketches is prone to bias and narrow coverage. Problems of biased or narrow coverage have been noted in the database and software performance community when designing benchmarks [LC82] – arguably an analogous task to designing corpora for recognition systems. When designing a recognition system, designers may unintentionally select the sketches on which to test the system based on their understanding of the strengths and weaknesses of their algorithm, essentially an instance of the Pygmalion Effect. As well, designers are typically researchers in sketch recognition, not mathematicians, engineers, animators, etc. As a result, designers may select sketches to be recognized based on their experience within the target domain, which is narrower than domain experts. For example, having only experienced introductory calculus, a designer of a math recognition system might not consider multi-variable integrals as an essential component in a math recognition system.

Whether participants transcribe the same set of sketches or unique sketches for any corpus has an effect on training and testing with that corpus. If some form of user-omission is used to test, i.e. train on n-1 users and test on the remaining user, then using the same sketches means that the system has been tested on sketches (from one user), but it has experienced each of these sketches before as training data from the other n-1 users. Leaving out sketches means the system has been trained on that specific user's handwriting, which may also provide a higher than realistic performance measure. On the other hand, providing unique sketches for each user can become tedious, particularly when a large number of participants are desired to ensure good coverage of drawer variability.

Finally, the process of manually ground-truthing a set of sketches can prove tedious and error prone. While what constitutes “sufficiently large” in the context of any corpus may be subject to interpretation, it seems obvious that, to be of any use in training and testing, a corpus would need to contain at least several hundred or thousand sketches.

Our motivation for creating a large public corpus is our math recognition system MathBrush. MathBrush is an experimental system for working with mathematics using pen-based devices. The system [LLM*08a] allows users to write mathematical expressions as they would using a pen and paper, and to edit and manipulate the mathematical expressions. Hand-drawn math recognition systems research spans four decades (eg. [BA69], [BCZ02], [LZ06]), and, in the absence of established corpora, it is challenging to assess the strengths and weaknesses of the wide variety of approaches to this problem explored in the past 40 years. Despite extensive research in hand-drawn math recognition, we are aware of no large, publicly available, ground-truthed math corpus for use in training and evaluation. This significant lack has motivated our creation of such a corpus. While our work focuses on mathematics recognition, tools for generating large, accurately ground-truthed corpora, with broad coverage of notation, are of value to all researchers studying sketch recognition.

In this paper, we describe our work on the generation of a publicly available ground-truthed corpus of mathematical expressions. The drawbacks of manual corpus creation are addressed in several ways:

1. To avoid bias, random walks of a grammar-based math recognizer are used to ensure coverage and variability in our expressions.
2. To generate a large corpus, 20 participants were asked to transcribe expressions generated by this random grammar. Participants received \$10 for one hour of transcription and produced more than 4500 expressions in all.
3. To avoid the tedium of manual ground-truthing, we designed a novel greedy algorithm that generates ground-truth for symbols and relationships between subexpressions with more than 90% precision.

Together, it is our hope that these contributions stimulate further work on the creation of large, transparent, public corpora for various sketch recognition domains.

This paper is structured as follows. Section 2 describes the process of generating our ground-truthed mathematical corpus and presents accuracy measures of our automatic ground-truthing technique. Next, strengths and weaknesses of our approach are discussed, and we highlight some open areas for future research. Finally, we contrast our approach with related work in corpus generation.

2. Creating the Mathematical Corpus

Creating a ground-truthed mathematical corpus required three tasks. First, we created an algorithm that generates random mathematical expressions using a relational context-free grammar. Next, study participants transcribed generated math expressions, producing a large collection of handwritten expressions. Finally, the handwritten expressions were automatically ground-truthed and the effectiveness of this process was evaluated. This section describes each of these tasks.

2.1. Mathematical Formula Generation

To create an unbiased set of equations, we use an automated technique that performs a constrained walk through a context-free grammar describing mathematical equations. This approach has many advantages. First, expressions are created based on random sampling of symbols and relationships, ensuring that rare symbol combinations are reflected in the collected data. Secondly, this approach limits biases that a researcher may have in selecting testing data, or, at the very least, makes the process of selecting equations somewhat transparent. Finally, this technique provides a limitless source of equations for participants to transcribe.

Grammar-based approaches to math recognition are not new. As mathematical expressions have strong positional semantics, it is natural to use a relational grammar to describe the structure of expressions, and to map the relationships identified by a grammar to the semantics of a complete expression. Our approach to math recognition uses a fuzzy relational context-free grammar to model the spatial structure of handwritten math. In this section, we first describe fuzzy relational context-free grammars, and then highlight how a relational grammar can be used in the generation of expressions.

2.1.1. Fuzzy relational context-free grammars

To capture the structure of handwritten math for recognition, we use a fuzzy relational context-free grammar formalism. Our work is described in detail in MacLean (2009) [Mac09]. The details are summarized below.

Recall that a *fuzzy relational context-free grammar* (fuzzy r-CFG) is a tuple $G = (\Sigma, N, S, T, R, \tilde{r}_\Sigma, P)$, where

- Σ, N are sets of terminal, nonterminal symbols respectively,
- $S \in N$ is the start symbol,
- T is the set of observables,
- R is a set of fuzzy relations on (T, T),
- $\tilde{r}_\Sigma$ is a fuzzy relation on (T, Σ),
- P is a set of productions, each of the form $A_0 \to A_1 \tilde{r} A_2 \tilde{r} \cdots \tilde{r} A_n$, where $A_0 \in N; n > 0; A_i \in \Sigma \cup N, 1 \leq i \leq n; \tilde{r} \in R$.

During recognition in our system, we take T to be the set of all possible ink inputs, R to be the set of relations between symbols, and $\tilde{r}_\Sigma$ to be the output of a symbol recognizer. Furthermore, each grammar production p is associated with a tree generator and a string generator. The tree generator produces an expression tree that describes how terminal symbols (leaves) are combined using mathematical operations to represent the syntax of the math expression. The string generator produces a string representation (e.g. LaTeX, MathML) of the expression tree. Fuzzy values are used to measure the most likely spatial relationships between groups of symbols in the hand-drawn input. They are combined with symbol recognition results to obtain confidence scores for expression trees produced when mapping a mathematical expression onto possible derivations in the grammar.

For extensibility, the grammar and associated generators are encoded in an external text file. For example, the following defines the grammar production for addition:

```
ADD :: [AT0] <R> plus <R> [RT]
       {ADD(%1 'EXPR_LHS', %3 'EXPR_RHS')}
       `%1 + %3` ;
```

In this example, on the first line, ADD is the name of the production's LHS nonterminal, AT0 and RT are two other nonterminals, plus is a terminal name, and R is the relation code for the RIGHT relation. The second and third line represent the tree and string generators, respectively. The second line of the production, between the braces, describes a tree with root label "ADD" with two children. The first child is labeled "EXPR_LHS" (the left hand operand of the addition operation) and is linked to the tree output by the tree generator for AT0. The second child is labeled "EXPR_RHS" (the right hand operand) and is linked to the tree output by the generator for RT. The string generator is described on line 3 (between the back ticks). As with tree generation, the %n notation indicates where to insert the output of string generators associated with the left hand operand AT0 and the right hand operand RT.

In our grammar, semantic content is described by the root labels produced by tree generators. The ADD production above therefore has semantic type ADD. Not all productions include tree generators, however. For example, consider the following three productions: (Pipe symbols are used on the RHS to separate distinct productions.)

```
RT :: [ADD] | [SUB] | [AT0] ;
```

The nonterminal symbol RT simply represents a collection of expression types with the same level of precedence, in this case, addition, subtraction, and an isolated addition term.

In this example, each of the three nonterminals that the symbol RT can produce have distinct semantic types. RT itself does not have a fixed semantic type. Rather, it inherits the expression tree (and hence the semantic type) given by the tree generator for the single nonterminal it produces in a particular derivation step. Unlabelled nonterminals like RT can therefore represent different semantic types in different contexts. Unlabelled nonterminals can derive other unlabelled nonterminals, so the semantic types they can assume are not always immediately apparent from their productions.

2.1.2. Random derivations

A random expression generator must balance two properties to be useful. It must generate samples typical of math expressions written on tablet computers (so that machine learning algorithms have a representative training set. It must also generate as many combinations as possible of bounding box shapes and relative positions (so that algorithms can be trained on low-probability spatial relationships). However, only a relatively small number of examples can be collected from any subject. In our experience, approximately 225 equations can be written by a subject in a one hour session. As a result, there exists a tradeoff between accurately representing the frequency of various mathematical relationships and providing a broad enough coverage of various mathematical relationships to fully train and test a recognizer. Our approach opts for broader coverage at the expense of reflecting the true frequency with which particular mathematical relationships occur in real-world expressions. This means, for example, that in our generated expressions, exponents, subscripts, and fractions are more common than one would normally encounter.

Random expressions are generated using the r-CFG for mathematical expressions described above. Fuzzy relationship values are not relevant when using a grammar to generate expressions. The essential idea of generating a random equation is quite basic: given a current symbol, choose a grammar production arbitrarily and recurse on each nonterminal in the RHS. However, care must be taken to avoid three problems:

1. Recursing blindly may lead to extremely large expression generation. Expression length must be constrained since the expressions are to be transcribed by human users.
2. Our recognition grammar is designed to reflect normal operator precedence; however, lower-precedence operations have shorter derivation distance from the start symbol than higher-precendence operations. In other words, choosing productions at random would give a biased set of expressions with lower-precedence operations over-represented.
3. Ascender (e.g. upper-case symbols, some lower-case symbols such as 'b'), descender (e.g. 'p'), and baseline (e.g. 'a', 'o') symbols provide different bounding box profiles for relationships. We must ensure that no type of symbol (ascender, descender, baseline) is over-represented in spatial relationships.

To limit expression length, we introduce a parameter $0 < p_{\text{inc}} \leq 1$. The algorithm begins with $p = 0$. Instead of simply choosing a production, it draws x from a uniform distribution on $[0,1]$. If $x < p$, a derivation leading to a single terminal symbol is selected if possible; otherwise p is incremented by p_{inc}, a random production is selected, and the algorithm recurses. This process does not guarantee a maximum expression size, but, by varying the value of p_{inc}, we can control the expected output size while still allowing a degree of variability in expression length.

To ensure that low-precedence operations are not over-represented, note that each precedence level is represented by a single unlabelled nonterminal such as `RT` from the second example in section 2.1.1. If the algorithm reaches a nonterminal which can derive multiple semantic meanings, it selects a meaning at random, derives the nonterminal having that meaning, and recurses. Otherwise, it selects a random production and proceeds as above. This modification gives a uniform distribution over the various mathematical operations supported by the grammar.

Finally, to obtain broader coverage of relative bounding box positions, the Latin and Greek letter symbols in the grammar were grouped into classes based on their characteristic shape with respect to a baseline (ascender, descender, baseline). The grammar was modified so that each class is produced by a single non-terminal. In this way, we obtained a uniform distribution over symbol shapes rather than symbols.

Below are two examples of expressions generated by the above process.

$$\left[\frac{B}{7}\right]^{N+6} \qquad \beta + \frac{z - L(v)}{\sqrt{s}} \int 24 d X_h$$

Figure 1: *Generated expressions*

2.1.3. Output formats

The algorithm sketched above generates random derivations from a grammar. Using the tree generator, one can produce and serialize parse trees for use in recognizer testing. Using the string generator, one can generate LaTeX strings, which can be converted to images and displayed to subjects for transcription. Strings representing generated expressions for use in a computer algebra system such as Maple or Mathematica can also be generated. The latter is particularly useful for automatically evaluating recognizer accuracy via simple, built-in arithmetic. These structures are generated for each randomly-derived expression.

The algorithm also creates a *derivation string* describing the spatial layout generated by the derivation. For example, suppose the expression $2 + m^a + \frac{b}{3}$ is generated. The corresponding derivation string is `2 _R_ plus _R_ (m _AR_ a) _R_ plus _R_ (b _B_ horzline _B_ 3)`. Here the parentheses indicate nested subexpressions and the underscores delimit relation codes (`_R_`=RIGHT, `_AR_`=ABOVE-RIGHT, `_B_`=BELOW). The derivation strings are used to guide the automatic ground-truthing process which we will described in section 2.3.

2.1.4. Supplementary expressions

In order to obtain training data for more common expressions and to accomodate subjects' expectations for transcribing math equations, 53 common expressions were prepared for transcription. For these expressions, the parse trees, LaTeX strings, and derivation strings were written by hand. An example of a prepared expression appears in Figure 2. Common expressions were interspersed with random derivations during transcription. Including a set of common expressions adds value to the corpus. The corpus comprises unique expressions as well as multiple samples of a single expression.

$$\frac{-b+\sqrt{b^2-4ac}}{2a}$$

Figure 2: *A prepared expression*

2.2. Data Collection

Twenty undergraduate students from the University of Waterloo participated in our study. Nineteen participants had high mathematical literacy (were in the Faculty of Math or Engineering) and one, from the Faculty of Arts, had modest mathematical literacy. Participants were recruited using posters and were given a $10 gift certificate in exchange for transcribing mathematical expressions for a one-hour session.

Data was collected using various models of Tablet PC's running custom collection software using the Microsoft Ink SDK under Windows XP. A screen shot of our data collection software is shown in Figure 3. The equations that participants were asked to transcribe consisted of automatically generated expressions. As well, 53 prespecified expressions, as described in section 2.1, were included in the first 150 expressions drawn by each participant. The software collected position, time and pressure information for each stroke drawn. Examples of typical transcribed equations are shown in Figures 4 and 5.

Each session was organized as follows. Participants were seated at a table in an ergonomic office chair in a private room. A Tablet PC was placed in front of them displaying the transcription interface shown in Figure 3. The functionality of the interface was described to participants and demonstrated by the researcher conducting data collection. Participants were asked to draw expressions presented at the top of the screen and then click 'Next' to generate a new expression. They could also click 'Clear' to clear the display and redo an expression. Participants were asked to write as legibly as they would on an assignment they would hand in for grading. Finally, participants were told that if they did not recognize a symbol, or were unsure how to draw a symbol, they could either ask for help or skip the expression.

As can be expected in such a study, a number of samples

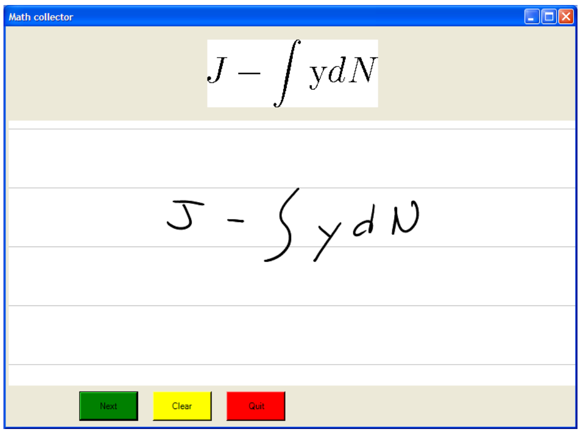

Figure 3: *Our collection software in action.*

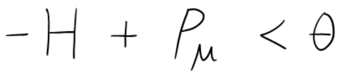

Figure 4: *A hand-drawn, randomly generated expression.*

were discarded. Our basic discard policy was that if an equation was incomplete or illegible to a human expert, it was discarded. An example of a discarded equation is shown in figure 6. We felt it was unreasonable to expect an automated system to correctly infer the expected equation if a human expert was unable to do the same.

In all, 5119 hand-drawn expressions were collected from 20 users. Of these, 109 were blank and 355 were illegible, resulting in 4655 valid hand-drawn expressions. Within these equations there were 25963 symbols drawn and 21264 relationships between subexpressions. The next section describes the process by which ground-truth was established for these expressions.

2.3. Automatic Ground-Truth Generation

When generating a corpus of ground-truthed data to train and/or validate recognition systems, it is convenient to use a program to automatically ground-truth at least part of the collected data. An obvious argument against this approach is that one needs a recognition system to generate the ground-truth data. The ground-truth annotations produced by the first recognizer are then used to train a second, new recognition system. With this approach, any new recognition system is constrained by the recognition system used to generate the initial ground-truth data. If the initial recognition system is less accurate that the new recognition system, there is no way to measure recognizer improvement without manually annotating the testing corpus with ground-truth.

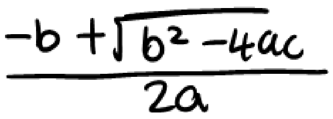

Figure 5: *A hand-drawn, predetermined expression.*

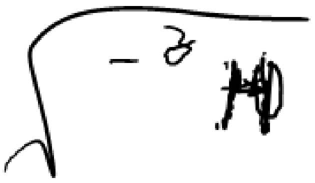

Figure 6: *A discarded, illegible equation.*

One way to overcome the disadvantages associated with manual ground-truthing is to have available additional data associated with a hand-drawn candidate expression. In our experiment, we have this data available: we computationally generated a candidate expression, including data representing its expression tree and its string representation, and then asked our participants to transcribe these expressions. Ground-truthing is, therefore, a tightly-constrained recognition task consisting of matching the symbol labels in the derivation string to groups of strokes in a handwriting sample. As a result of the additional constraint on possible recognizer output, a weak recognition engine should generate useful ground-truth.

2.3.1. Algorithm

We devised a naive greedy algorithm for matching strokes to derivation strings. The algorithm uses a symbol recognizer and spatial relation classifiers trained on a small bootstrap data set which was manually annotated with ground-truth. This bootstrap data is independent of our main corpus and was obtained by having several researchers in our group draw a small set of equations. The ground-truthing algorithm is based on an inaccurate but somewhat useful heuristic: given a derivation string, remove all the parentheses and match the derivation string to sketched equation symbol by symbol. For example, the derivation string in section 2.1.3 is interpreted as `2 _R_ plus _R_ m _AR_ a _R_ plus _R_ b _B_ horzline _B_ 3`.

The algorithm considers a derivation string of the form $S_0 r_1 S_2 r_2 \ldots r_n S_n$ where the S_i are names of terminal symbols and the r_i are any of the spatial relations. Given such a string, suppose the algorithm is currently matching S_m and has assigned the current partial ground-truth derivation a score, z, computed as the product of local relation and symbol recognition confidences on the prefix $S_0 r_1 \ldots r_{m-1} S_{m-1}$.

Local confidences are computed by multiplying relation and symbol recognition confidences. The algorithm proceeds as follows:

```
Order all possible occurrences of $S_m$ in the input ink in decreasing order of confidence
for each occurrence do
  Let $c$ be the local confidence at $r_m S_m$
  Remove recognition results for the relevant input
  Recurse at $S_{m+1}$ with score $zc$
  if a viable match was found then
    return the match
  else
    Replace the symbol recognition results
  end if
end for
return failure
```

This algorithm may take exponential time to report failure in the worst case. During testing the algorithm was stopped if it had not reported a match after two minutes.

2.3.2. Results

To test the accuracy of our algorithm, the entire corpus of 4655 expressions was annotated manually. Automatically-annotated data was then compared to the manually-annotated data using two scenarios.

1. The algorithm described in Section 2.3.1.
2. Pre-training the symbol recognizer for each user on approximately 20% of their input data, selected randomly, and then running the algorithm from Section 2.3.1.

Scenario 2 was introduced because the algorithm frequently failed when it could not match a hand-drawn symbol to a symbol in the specified input string. The second scenario ensures that the symbol recognizer has the greatest possible chance to match user-drawn symbols against symbols in the generated expression tree.

For each scenario, three accuracy measurements are reported, normalized to give percentages:

1. *Full expression.* In this measurement, an annotated sample is considered correct if all symbols and relations are correctly annotated and all subexpression bounding boxes exactly match their counterparts in the manually-annotated ground-truth.
2. *Exact bounding-box.* This measurement concerns individual relations arising from the randomly-generated derivation. A relation between any two subexpressions is considered correct if its constituent symbols are correct and the automatically-annotated bounding boxes of both subexpressions exactly match their counterparts in the manually-annotated ground-truth.
3. *Bounding-box similarity.* This measurement also also concerns individual relations. In it, each relation is assigned a score between 0 and 1. The score is computed by averaging bounding-box similarity measurements from each of the automatically-annotated relation's constituent subexpressions to their counterparts in the manually-annotated ground-truth. We define bounding-box similarity as the ratio of the area of intersection of the two boxes to the area of the larger box. Two boxes thus have similarity 0 if they are disjoint and similarity 1 if they are identical with a range of possible values in between.

These three measurements are defined in increasing order of permissiveness of match. Full expression accuracy requires every symbol, relation, and bounding box in an entire sample to be annotated precisely the same as in the manual ground-truth, while bounding-box similarity accuracy allows symbol bounding boxes to disagree slightly but still count as a close match.

The ground-truthing algorithm produces highly accurate ground-truth for hand-drawn input when compared to manual ground-truth, as shown in Figure 7. For scenario 1, the algorithm achieved 90% full expression accuracy. Exact bounding-box accuracy was 93.6% and overlap accuracy was 97.7%. The accuracy rates for scenario 2 are about the same. The error bars in the graph indicate the algorithm's accuracy on the data of the most- and least- accurately annotated subjects.

Scenarios 1 and 2 differ significantly in their reject rates. The algorithm either produces highly-accurate ground-truth or rejects the entire expression. For scenario 1, the reject rate was 55.1%, while for scenario 2, it dropped to 46.5%. This reject rate will be discussed in more detail in section 3.

The most appropriate measure of annotation accuracy depends on one's application. If annotated data is used to train spatial relation classifiers on bounding box information, it is appropriate to count similar, but not identical, bounding-box annotations as partially correct because they may still provide useful training data. On the other hand, if one uses automatically-generated ground-truth to test the accuracy of an isolated symbol recognition system, then full expression or exact bounding-box accuracy may be a more appropriate measurement.

3. Discussion

Our automatic ground-truthing method has a high accuracy rate but also suffers from a high reject rate. It is important to note that the accuracy of this annotation technique remained high through all experiments. While it would be preferable to maintain accuracy with a lower reject rate, the present algorithm is far more useful than it would be if both accuracy and reject rates were lower. Accurately ground-truthing even half of a large corpus automatically would obviate a significant amount of manual annotation.

It is not clear how to further reduce the rejection rate of the ground-truthing algorithm. In scenario 1, described in 2.3.2,

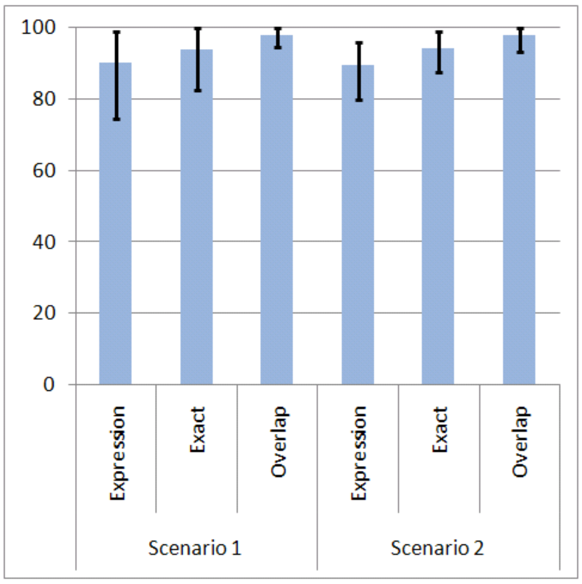

Figure 7: *Automatic annotation accuracy*

a number samples were rejected due to symbol classification errors. The errors arose because the generic models in the symbol classifier differed from particular users' handwriting styles. By pre-training on 20% of samples, the rejection rate was reduced from 55.1% to 46.5%, as is shown in the results of scenario 2. However, experiments pre-training the symbol recognizer on a larger proportion of samples did not demonstrate significant improvement.

One approach to lowering the reject rate is to incorporate manual intervention directly into the ground-truthing application. In cases currently rejected due to low recognition scores, an operator could manually annotate the problem symbols and let the automatic procedure handle the rest. We expect such an approach would handle the majority of the cases rejected by the current system while requiring only a fraction of the time required for full manual annotation. Other approaches to reducing the reject rate may also be possible. Our corpus includes the entire set of expressions, and other researchers are free to experiment with algorithms to perform symbol identification and subexpression relation assignment.

The template expressions transcribed by study participants were generated by uniform random sampling of grammar derivations. We intended for this approach to prevent researcher bias from affecting our sample selection. If one believes that any selection of samples for a corpus is biased, our approach, at the very least, makes the selection of ground-truth expressions transparent. We did not control the selection of expressions to ensure coverage of structures well-recognized by our system. We also had no control over the frequency with which different spatial relationships appeared in our corpus. Our method of generating expressions ensured that we did not unintentionally bias our samples toward (or away from) symbols or relations which are recognized more accurately than others.

Using a context-free grammar to generate expressions allows for future expansion and enhancement of the corpus. Creating additional samples for more subjects to transcribe is completely automated. Expanding the corpus to cover new classes of expressions or symbols, such as those used in set theory for example, is equally trivial. Rather then manually creating hundreds or thousands of new templates, a few additions to the grammar allow for a virtually unlimited number of new templates drawn from the newly added class.

It is interesting to note that in order to verify our accuracy claims, we manually ground-truthed the entire corpus. Unfortunately, with any new technique, some measure of its effectiveness is necessary. In this case, to test how well we could ground-truth data, we needed ground-truthed data to compare our algorithm to! However, we now have the confidence that, should we expand our corpus, we can automate the ground-truthing and still expect high accuracy in the data we generate.

Upon review of our data collection techniques we have some observations that may be relevant to other researchers. Some participants had data entry errors which could have been prevented if they had asked for guidance. The handwriting in users' first few transcriptions was often shaky due to lack of experience using Tablet PCs. Users sometimes left expressions blank or incomplete. Some users were unfamiliar with the shapes of certain Greek letters and copied the typeset representation. In future studies we will likely inspect each participant's data after a few samples and offer suggestions.

We believe the techniques and results presented in this paper have implications beyond mathematical expression generation and are applicable to any sketch recognition domain where the domain can be expressed by a formal grammar. In domains such as chemistry modeling, UML diagramming, military course of action diagrams, or electric circuits, it is possible to generate large collections of examples and ground-truth using random walk and ground-truthing techniques similar to those we have used.

4. Related Work

We are not aware of any work on tools and algorithms to aid in the generation and automatic ground-truthing of large, varied on-line sketch corpora. However, OCR researchers have developed several techniques for automatically generating ground-truthed training and testing data. These techniques generally either generate perfectly ground-truthed synthetic data (eg. [HBAT07], [OP00]), or match real inputs to separate ground-truth created by hand, potentially with mistakes in both matching and ground-truthing (eg.

[BSB08]). Occasionally aspects of both approaches are combined as in [KBNJ06].

Our techniques for generating and ground-truthing data have much in common with both approaches, but also has some important differences. Synthetic data is generated as in the first approach, but this data is intended to be transcribed by human users. Real input is matched to ground-truth, but this ground-truth is automatically generated and free from errors. By decoupling expression generation from ground-truth generation, we are free to experiment with algorithms for each task separately.

As noted in the introduction, research in handwritten mathematics recognition has been ongoing for decades and the field is still an active area of research. It is becoming standard practice to study accuracy and usability by asking participants to transcribe a collection of expressions, as in [LaV07], [SNA99], and [LLM*08b]. However, we are not aware of any large corpora of handwritten mathematical expressions. Most studies have collected a modest set of predetermined expressions, making it difficult to predict the recognizer's capacity to generalize to more varied expression sets.

Among the papers cited above there is substantial variation in the size of the testing suite, the number of participants, the types of expressions, and the number of recognized symbols. Furthermore, there are differences in the criteria used to determine correctness. Is an expression correct if all the symbols are recognized correctly? If the system infers the correct expression despite misclassified symbols? If a recognizer produces multiple interpretations, must the first result be correct? If not, how many incorrect interpretations are tolerable?

5. Conclusions

In this paper, we described a technique for generating unique mathematical expressions by randomly constructing a context-free grammar derivation. Our procedures for collecting hand-drawn expressions were discussed, and a novel technique for automatically generating ground-truth was presented. Although these techniques were demonstrated by building a large corpus of hand-drawn mathematical expressions, the methods can be applied to any sketch recognition domain that can be described by a formal grammar.

The mathematical corpus described in this paper will be made publicly available at the URL:

http://www.cs.uwaterloo.ca/scg/MathBrush/mathdata/

We hope that other researchers will find the data useful not only to aid in the development of their own character or math recognition systems, but also to facilitate comparison between systems. For a comparison to be meaningful, there should exist a common, transparent, and unbiased set of equations, consistent separation of the training data from the testing data and similar criteria for determining accuracy. We have therefore also suggested possible partitions of the data into training and testing sets.

References

[BA69] BLACKWELL F. W., ANDERSON R. H.: An on-line symbolic mathematics system using hand-printed two-dimensional notation. In *Proceedings of the 1969 24th national conference* (New York, NY, USA, 1969), ACM, pp. 551–557.

[BCZ02] BLOSTEIN D., CORDY J. R., ZANIBBI R.: Applying compiler techniques to diagram recognition. In *ICPR '02: Proceedings of the 16 th International Conference on Pattern Recognition (ICPR'02) Volume 3* (Washington, DC, USA, 2002), IEEE Computer Society, pp. 127–130.

[BSB08] BEUSEKOM J. V., SHAFAIT F., BREUEL T. M.: Automated ocr ground truth generation. In *Document Analysis Systems, 2008. DAS '08. The Eighth IAPR International Workshop on* (Sept. 2008), pp. 111–117.

[HBAT07] HEROUX P., BARBU E., ADAM S., TRUPIN E.: Automatic ground-truth generation for document image analysis and understanding. In *Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on* (Sept. 2007), vol. 1, pp. 476–480.

[KBNJ06] KUMAR A., BALASUBRAMANIAN A., NAMBOODIRI A., JAWAHAR C.: Model-Based Annotation of Online Handwritten Datasets. In *Tenth International Workshop on Frontiers in Handwriting Recognition* (Oct. 2006), Guy Lorette, (Ed.), Université de Rennes 1, Suvisoft.

[LaV07] LAVIOLA JR. J. J.: An initial evaluation of a pen-based tool for creating dynamic mathematical illustrations. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses* (New York, NY, USA, 2007), ACM, p. 47.

[LC82] LEVY H. M., CLARK D. W.: On the use of benchmarks for measuring system performance. *SIGARCH Comput. Archit. News 10*, 6 (1982), 5–8.

[LLM*08a] LABAHN G., LANK E., MACLEAN S., MARZOUK M., TAUSKY D.: Mathbrush: A system for doing math on pen-based devices. *The Eighth IAPR Workshop on Document Analysis Systems (DAS)* (Sep 16-19 2008).

[LLM*08b] LABAHN G., LANK E., MARZOUK M., BUNT A., MACLEAN S., TAUSKY D.: Mathbrush: A case study for interactive pen-based mathematics. *Fifth Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM)* (June 11-13 2008).

[LZ06] LAVIOLA J., ZELEZNIK R.: Mathpad2: a system for the creation and exploration of mathematical sketches. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Courses* (New York, NY, USA, 2006), ACM Press, p. 33.

[Mac09] MACLEAN S.: *Parsing handwritten mathematics*. Master's thesis, David R. Cheriton School of Computer Science, University of Waterloo, 2009.

[OP00] OKUN O., PIETIKAINEN M.: Automatic ground-truth generation for skew-tolerance evaluation of document layout analysis methods. In *Pattern Recognition, 2000. Proceedings. 15th International Conference on* (2000), vol. 4, pp. 376–379 vol.4.

[SNA99] SMITHIES S., NOVINS K., ARVO J.: A handwriting-based equation editor. In *Graphics Interface* (1999), pp. 84–91.