Math Retrieval Using Leaf-Root Expression Trees

Andrew Norton University of Virginia
Charlottesville, Virginia
apn4za@virginia.edu

Ben Haines University of Virginia
Charlottesville, Virginia
bmh5wx@virginia.edu

ABSTRACT

This paper describes our implementation of a system for searching for mathematical expressions. The paper summarize existing techniques for math retrieval and describes our particular implementation. We propose a few extensions to implement and provide experimental results that measure the effectiveness of these proposals.

1. INTRODUCTION

1.1 Background

Within the field of Information Retrieval, the task of mathematical expression retrieval is a topic that has been attracting an increasing amount of attention in recent years. Math search is useful in a variety of situations, particularly for students and practitioners of technical fields. Particular scenarios include researchers who want to discover relevant work relating to a particular function or a student who needs help solving a particular problem. Existing solutions are often unsatisfying. For example, consider arxiv.org and math.stackexchange.com. These sites are two of the largest resources of collected mathematical information for both professional researchers and student. However, the usefulness of much of this information is decreased by challenges in locating it. The search feature of the arXiv does not permit searching for commonplace symbols such as "+" or "-". Searches on Stack Exchange often return no relevant results despite additional efforts revealing that multiple relevant results do exist. A better math retrieval system could in both cases increase productivity of many users.

Multiple approaches to parsing, indexing, and searching mathematical expression have been proposed. A specific state of the art algorithm does not exist as research in the field has not had time to converge to optimal solutions for the problems of storing and parsing expressions. Approaches tend to fall into one of two categories, those that use established text based search methods with modifications in order to apply them to the particular problem of Mathematical Information Retrieval (MIR) and those that use tailored approaches that attempt to take advantage of the inherent structure of expressions to improve performance. We provide a brief comparison of techniques in these categories and justify our particular choice of a system to implement. Our research focuses on the effectiveness of the Leaf-Root path system for representing structured expressions. The primary contribution of the work is the suggestion of query expansion in order to allow searches for generalized expressions and an examination of how changes in the parsing grammar can effect performance.

2. RELATED WORK

Among the earliest discussions of a math retrieval system, and one of the few describing an actual large scale implementation is the paper by Youssef and Miller regarding the Digital Library of Mathematical Functions[**?**]. The idea proposed involves a sequence of steps to process mathematical notation into a format recognizable by existing search engines. This involves first using macros to map math symbols to standard alphanumeric text representations. For example "+" and "<" are mapped respectively to "plus" and "lt". Next, nested expressions, such as exponents, are flattened. Finally, expressions are normalized by sorting the leaves of the corresponding parse tree in a standardized manner.

The authors select an evolutionary approach that augments existing text search engines due to practicality constraints but they also outline a few relevant challenges that they suggest could be better addressed with a structural approach. These challenges include

- Recognition of mathematical symbols
- Capturing and indexing structure
- Accounting for mathematical "synonyms"

Since the 2003 publishing of the paper discussed above, incremental improvements have been suggested by a variety of sources. In 2007 Miner and Munavalli introduced a more involved for processing inputs while still relying on a standard text search infrastructure for the fundamental search operation.[**?**] Largely within the last five years a number of papers have emerged that attempt to address the problems in more fundamental ways. In 2012 an approach that bridges the gap between text and structure based methods is introduced in the paper "A structure based approach for mathematical expression retrieval".[**?**] The fundamental idea is to use a modified longest common substring (LCS) algorithm to measure similarity between expressions. The expressions

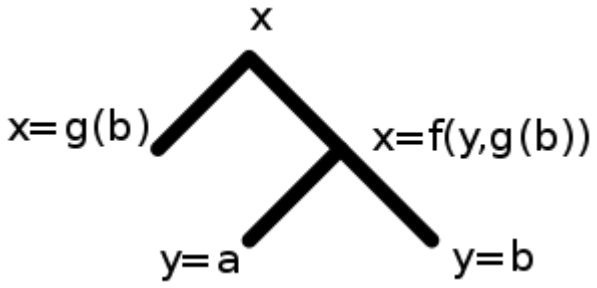

Figure 1: A sample substitution tree

are tokenized and each token is given a label to indicate its nested depth. The LCS algorithm then weights the similarities between two expressions by how closely depth markers are aligned. An alternate structure capturing approach that has been explored in a number of papers is that of substitution trees.[**?**][**?**] In such a tree each internal node represents a generalized expression form and leaves represent specific expressions for which all variables have a substituted value. A simple example tree is shown in the next figure. This structure stores relationships between abstract expression forms. Within each node individual expressions are stored as symbol layout trees that map spatial relationships between individual components of an expression. The authors of this approach reported performance improvements when compared to a standard Lucene search that they attribute to substructure queries enabled by the tree structures.

The approach that most directly influenced our implementation is an alternative method for using trees to store structure. Furthering our intuition that parse trees could be a useful representation of structure the particular implementation by Zhong provided inspiration and solutions to some technical challenges.[**?**]

Despite these recent innovations it is interesting to note that the top performers on MIR tasks, such as the main task of NTCIR11 are still achieved by traditional text based systems with modifications.

3. PROPOSED SOLUTION

Our solution to the problem of Mathematics Information Retrieval naturally divides itself into three categories: document and query representation, indexing method, and the query execution step. We discuss these three components in the sections below.

3.1 Representation

The system was inspired by and particularly targets the Stack Exchange and arXiv use cases which both store math information as LaTeX. Additionally, the corpus of expressions made available to us contains LaTeX expressions. For these reasons we chose to focus on searching for LaTeX expressions rather than alternatives such as MathML. As a result of this we decided to accept LaTeX formatted inputs. The advantage of this choice is that LaTeX is a widely used system for expressing mathematical structures, in particular, users who want to search a LaTeX corpus are probably also capable of composing queries with it. As well as this, many simple expressions can be given without any overt formatting and will still be valid which increases ease of use.

When given a raw string, whether it be as part of the corpus or in a query the system first performs a number of preprocessing operations utilizing a grammar generated with the ANTLR software package. (See Appendix A for grammar and lexing rules.)

First, whitespace and commands which only purely change the display of an expression and not the structure or content are removed. Second, semantically similar operations are grouped roughly into equivalence classes and occurrences are replaced with a single symbol when the lexer runs. For example, "`*`," "`\times`," and "`\cdot`" are all multiplication symbols with different TeX representations. These are treated as identical and replaced with a single token "`MUL`." Other operations that might have clear differences in meaning but are also similar in some way such as "`\int`" and "`\sum`" are replaced with the common command "`BIG_OP`". Numbers are all normalized to the symbol "`NUM`" and single-letter variables (including lower-case Greek letters) to "`VAR`".

After this lexical preprocessing step, the expression is passed through a parser designed to recognize simple LaTeX constructs. At a high level, the idea is that a LaTeX math expression consists of sub-expressions joined by an operator (for instance, "`+`") or prefaced by a function name (for instance, "`\sin`"). However, since we must also account for the way a human would write math (like omitting some operators or parenthesis), we also consider the idea of a "packed" expression which should be treated as a single term. For instance, if one were to write "3âb", we expect them to mean 3^{ab}, not $3^a b$.

3.2 Index Construction

After the expressions have been processed we are left with a collection of tree structures that need to be organized into some way to allow effective structure based search. This is the key component of the leaf-root tree approach. Each expression is given a unique ID number and its tree is decomposed into a collection of leaf root paths. For example, consider the parse tree displayed in figure 23.2 corresponding to the expression "2 * (3 + 4)". After normalization the tree is decomposed into the two distinct paths "2→ TIMES" and "NUM→ ADD→ MUL". These paths provide a loose approximation of the structure of the tree as a whole. There are a few immediate advantages of using parse trees and in particular leaf-node paths. The first is that they assume commutativity of operations. If the expression in figure two was instead "2 * (4 + 3)" its representation as a collection of leaf-root paths would be unchanged. Although there are obvious exceptions this is a good general assumption as it is true for most common operators. The second and more important property is that it inherently supports searching for subexpressions as they form subtrees. Given two expressions E_1, E_2, if E_1 is a subexpression of E_2 then all of its paths will also be sub-paths of $E2$'s paths. Any tree that contains a subexpression of the form NUM * (NUM + NUM) is guaranteed to have these paths as sub-paths. It's important to note that the paths do not provide a complete description of the tree, some information is lost in their creation. That is, it is possible to have trees that represent distinct expressions but which decompose into the same paths. Thus the paths are used as a first filter to reduce search space rather than a final retrieval mechanism. In order to take advantage of the paths, we structure the index as a filesystem. In our implementation we used the computer's actual filesystem for

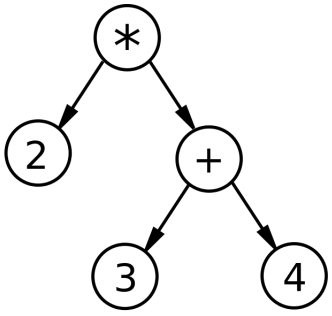

Figure 2: A sample expression tree

convenience but this could be implemented in a more efficient way for this dedicated task. Sub-directories are created for each node on the path, beginning at the leaves and the ID of the expression is stored at the deepest locations. For example, the sample expression would have its ID stored in ./NUM/MUL and ./NUM/ADD/MUL.

The IDs themselves are used as keys to lookup information about the particular expression in an external structure. This might include the original typesetting, the actual parse-tree, the source webpage, or other relevant information depending on the context that the search is operating in.

3.3 Queries and Retrieval

As explained in the representation section, queries are accepted in LaTeX format. They are subjected to the same process of normalization and parsing as the corpus expressions are and decomposed into leaf-root paths. In order to retrieve relevant expressions from the corpus we just traverse the index according to each of paths. Continuing the example from above, assume that the index contains only the expression from figure 2 and that a user has queried for "5 + 2" which breaks down to two identical paths "NUM→ ADD". We first traverse the filesystem to the location ./NUM/ADD and then return the merged contents of all of the subdirectories. This process gives us every expression for which the query is a sub-part.

Depending on the query, and particularly for short queries, the returned set of expressions is a potentially significant portion of the entire corpus. Thus a method is needed for ranking within this filtered set. We use a few different approaches based on the structure of the trees in order to rank, and we also reintroduce the symbolic information removed in the first parsing step in order to compare the literal terms of the query.

To rank the documents, we break the query into all leaf-root paths. For instance, if we were to search for the expression "`(a+b)*(x^3 + \sin(x))`," we would have the following paths:

- VAR→+→*
- VAR→+→*
- VAR→ˆ→+→*
- NUM→ˆ→+→*
- VAR→TRIG→+→*

For each of these paths, we iterate over all prefixes. For each prefix, we add increment the document score of every document within the directory (including subdirectories) corresponding to the prefix. For example, in the first path, we would increment every document within the `./VAR` directory, then within `./VAR/ADD/`, then `./VAR/ADD/MUL`. This gives more weight to document paths that are similar to the query at greater depth within the tree.

To reintroduce term matching, we use a boolean similarity model on the variables (and numbers) used in a query. This similarity is then used to further increase the score of documents that use the same variables as in the query.

4. EXPERIMENTS

The lack of an annotated corpus and the difficulty involved in creating one put limitations on our ability to use standard metrics to evaluate the system's performance. We were also unable to take full advantage of the Stack Exchange corpus due to our system's inability to parse many of the symbols used. As a result, our testing procedure consisted of selecting a "target" expression within a hand constructed corpus and then searching for modified versions of the expression or manually created expressions that we deemed "relevant" and observing if the target expression is returned. Some examples are presented here and general comments are given.

Examples: First we examine the results of a very simple query: $(a + b)$. Some results are omitted for the sake of brevity.

- a + b + c with priority 10.0
- (a + b + c) * d with priority 10.0
- (a + b) * (c + d) * e with priority 10.0
- b + a + c with priority 10.0

We can see that the first "tier" of results consists of expressions that contain exactly the expression we are searching for (or a permutation of it). In the second tier we find the results that have similar structures but use different symbols.

- $3\alpha + x$ with priority 6.0
- $3\alpha + y$ with priority 6.0
- $(x + 2)(x - 2)$ with priority 6.0
- $x + 1$ with priority 6.0
- $x + y + z$ with priority 6.0

A slightly more complex query demonstrates a similar pattern in the results: $\int x^3$

- $\sum x^3$ with priority 18.0
- $\int x^3$ with priority 18.0

These results demonstrate how the grammar groups similar operations, in this case $\int$ and $\sum$. The next tier contains the same structure of an x^3 term within a sum but with the addition of extraneous terms.

- $\sum(x^3 + y + z)$ with priority 10.0
- $\sum(2 * x^3)$ with priority 10.0

After this tier we begin retrieving results that simply contain an exponentiated x.

- $x^2 + 4x - 10$ with priority 8.0
- $x^2 - 4$ with priority 8.0

The results reveal some interesting unintentional properties of our ranking algorithm. One of these is that in some cases it places more priority on matching variables than structures. For example, the query $x + y + z$ returns $\sum(x^3 + y + z)$ with a slightly higher priority than $a + b + c$. Depending on context, this could be seen as a benefit or a detriment. There are some cases however that seem to result in decidedly negative behavior. The query $x^4 - 2$ returns both $x^2 - 4$ and $x^2 + 4x - 10$ higher than $x^4 + 3$ which seems likely to be the most desirable result in most cases. The mistake here is caused by a failure to recognize that in most situations the degree of a polynomial is more important than constant terms. The problem could potentially be solved by introducing some weighting factors that put more value on certain parts of an expression. These weights could be determined by hand or could be computed as a function of the depth in the parse tree.

An unintended consequence of the ranking function is that there is not a straightforward way to compare priority scores across multiple queries. This makes it difficult to integrate the query expansion step discussed earlier. Nonetheless we can see the benefit of such a system by evaluating the results separately.

For example, a search for $\sin(2x)$ returns $x^2 + 4x - 10$ as the top result because the corpus does not contain any expressions that are obviously relevant. However, using one of the alternate forms provided by the Wolfram API we get $2\sin(x)\cos(x)$ as the highest priority result. Visually, besides also including trigonometric functions, there's not an immediate reason to believe this result is relevant without the outside knowledge that allows us to know they are actually equivalent.

The examples so far have been fairly simple for the sake of clarity. The grammar limits the possible expressions mostly to exponential, rational, and trigonometric functions but within this bound the expressions can be arbitrarily complex. For example: $1/2 + (3 * \sin(y) - x^2)/4$

- $(x^2 - 4)/(x^2 - 5 * x + 6)$ with priority 76.5
- $\sin(d)/(4 * x) - x^2$ with priority 52.0
- $x^2 + 4x - 10$ with priority 39.0

5. LIMITATIONS

There are a number of limitations associated with our work here. Some of them are inherent to the field, some are a result of our choice of approach, and some are a result of our particular implementation. Due to the newness of the field there is a scarcity of mature tools for solving this kind of problem. There are also few established or annotated datasets available. Due in particular to the problem of identifying mathematical synonyms, it is difficult to create relevance judgements for arbitrary queries.

A particular weakness of our system is that it focuses on mathematical expressions to the complete exclusion of any textual information. It is not possible for example for a user to type an expression and to indicate what in particular they would like to know about the expression, e.g. it's derivative, or integral, etc. The system would benefit greatly by integration with a standard text search system, perhaps using delimiters to indicate which portions of a query contain mathematics and which text.

Although the author of the Cowpie project provided the grammar used to parse LaTeX, we used an alternate system which didn't allow direct translation. As a result we were unable to capture the full complexity of the markup and our searches were limited to a less interesting subset of the total search space. This led to less interesting results but we expect that the techniques employed would generalize to more complex expressions without too much difficulty.

6. CONCLUSIONS AND FURTHER WORK

Beyond the immediate goal of extending the grammar to allow full LaTeX support there are a few ways that we could add to the system. A way to improve the results returned would be to use a more complex ranking function that takes into account more properties of the structure of the expression. For example, currently we group all variables into one class regardless of name. This disregards the fact that users will often use the same variable in multiple places, expecting it to have the same meaning. For example, in the expression $\int_0^1 x^2 dx$ it is important that the variable x is the same in both occurrences.

A feature that would increase usability would be the ability to mark certain symbols as "free" or "fixed". That is, to indicate to the system that the particular variable name used has some special meaning and should not be allowed to vary. For example, if the user wants to search for a formula involving standard deviation, they may use the variable σ because they know that it is the standard notation. Returning results that appear to be similar but don't contain σ may decrease relevancy in this case.

Beyond these there still remain much larger conceptual problems facing the field of MIR. Although we have attempted to improve the situation by the use of query expansion, the problem of mathematical synonyms introduced by Youssef and Miller still remains a significant challenge. Understanding notational differences such as $\sum_{i=1}^{n} x_i$ compared to $x_1 + \cdots + x_n$ would appear to require a higher level understanding of the operations involved.

APPENDIX

A. LEXER RULES

```
/* Simple math */
MUL : '*' | '\times' | '\\cdot' ;
ADD : '+' ;
SUB : '-' ;
DIV : '/' ;


NUM : [0-9]+ ;
WS : [ \t\r\n]+ -> skip ;

/* Variable characters */
VAR
   : [a-zA-Z]
   | '\\alpha'
   | '\\beta'
```

```
    | '\\gamma'
    | '\\delta'
    | '\\epsilon'
    | '\\zeta'
    | '\\eta'
    | '\\theta'
    | '\\iota'
    | '\\kappa'
    | '\\lambda'
    | '\\mu'
    | '\\nu'
    | '\\xi'
    | '\\pi'
    | '\\rho'
    | '\\sigma'
    | '\\tau'
    | '\\upsilon'
    | '\\phi'
    | '\\chi'
    | '\\psi'
    | '\\omega'
    ;

BIG_OP
    : '\\int'
    | '\\sum'
    ;

TRIG_OP
    : '\\sin'
    | '\\cos'
    | '\\tan'
    | '\\sec'
    | '\\csc'
    | '\\cot'
    ;
```

B. GRAMMAR RULES

```
grammar LatexGram;

s : expr;

/* Expressions: subexpression with operator(s) */
expr
    : atom          # atomic
    | pack          # packed
    | pack '^' pack # pow
    | BIG_OP expr   # bigOp
    | pack pack     # iMul
    | expr DIV expr # div
    | expr MUL expr # mul
    | expr SUB expr # sub
    | expr ADD expr # add
    | TRIG_OP pack  # trig
    ;

pack
    : atom
    | '(' expr ')'
    | '[' expr ']'
    ;

atom
    : NUM             # atmNum
    | VAR             # atmVar
    | '{' expr '}'    # atmExpr
    ;
```