A Novel Similarity-Search method for Mathematical Content in *LaTeX* Markup

Wei Zhong, Hui Fang
Dept. of Electrical and Computer Engineering
University of Delaware
Newark, DE USA
{zhongwei, hfang}@udel.edu

ABSTRACT

A relaxed structural matching search method, along with a symbolic similarity measurement algorithm for mathematical content search is proposed. Our approach uses an intermediate tree representation to capture structural information of mathematical expression, and based on a previous idea which indexes math expression structure through tree leaf-root paths, we further describe an advanced AND search method in a formal way. This search method can be used to test query/document subexpression isomorphism or evaluate the symbolic similarity between math expressions with consideration of their α-equivalence. For the purpose of evaluation, we also implement a search engine based on our idea.

Categories and Subject Descriptors

H.3 [**Information Search and Retrieval**]: Miscellaneous

General Terms

Algorithms

Keywords

mathematical searching, language processing, search engine

1. INTRODUCTION

With *MathJax* becoming popular, more and more LaTeX markups can be crawled directly from many websites. In order to search those mathematical language in LaTeX markups, a search method that can respect the properties of math expression needs to be developed. Although many researches have been conducted to retrieve information in structured content (e.g. *MathML*), information retrieval on LaTeX math content is still not well-studied or exhaustively covered by mainstream IR research, compared to that on general text.

But sometimes we need to rely on conventions and context to distinguish expressions like $f(a+b)$ and $c(a+b)$ because the symbol f in former expression is likely to represent function instead of a variable, in addition, expression such as f^{-1} can either be reciprocal or an inverse function.

Yet the problems addressed above are not considered in this paper, instead, we are focusing on the aspects of the following: First is symbolic similarity, with the consideration of α-equivalence. We know that symbols can be used interchangeably in each math formula to express the same meaning, e.g. $a^2+b^2=c^2$ and $x^2+y^2=z^2$. Nevertheless, we still weight symbolic similarity sometimes, for instance, $E=mc^2$ is considered more meaningful when exact symbols are used rather than just being structurally identical with $y=ax^2$. On the other hand, we should also weight α-equivalent expressions, that is, changes of symbols in expression preserve more syntactic similarity when changes are made by substitution, e.g. for query $x(1+x)$, expression $a(1+a)$ are considered more relevant than $a(1+b)$. Because the "bond variable" a in the former expression is supposed to represent the same value. Second is structural similarity. For example, $ax+(b+c)$ is not equivalent to $(a+b)x+c$ although they have the same set of symbols, this is because their structural difference.

Our system tries a different approach to make use of the structure of mathematical formula to solve the problems addressed above. It uses an efficient way to parse and tokenize mathematic formula, to transform a tree-structured formula to a more comparable structure, and is able to score and rank results.

2. SYSTEM DESCRIPTION

The system presented by us has a WEB front-end to accept user input in LaTeX format and pass it to the back-end. The back-end, mainly consists of a parser and a search program. The function of parser is to do tokenization and tree construction as well as storing output structure into our collection. The search program compares the query and document and evaluates the similarity of them, gives the final ranking through output file for the WEB front-end `CGI program` (Apache Common Gateway Interface, we use it as a way to interact between WEB interface and back-end program) to read. We input mathematical content in LaTeX as document to our parser by either manually creating or by a crawler script specifically targeting at mathematical content website *Mathematics Stack Exchange*[1].

2.1 Tokenization and Tree Construction

[1]http://math.stackexchange.com/

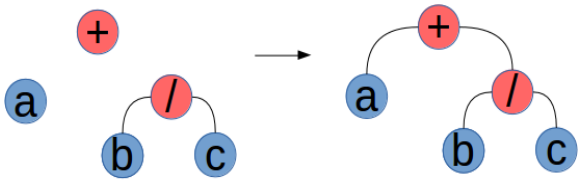

Figure 1: Example of Sub-tree generation for the addition grammar.

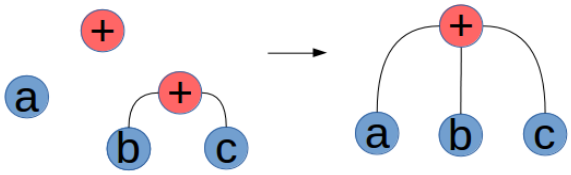

Figure 2: Cases where commutative property applies.

The parser is designed to tokenize a subset of mathematic related LaTeX language, we utilize `Lex/Yacc` tools to tokenize the LaTex Language and construct a "tree" for each equation. The LALR parser generator of `Yacc` can handle language efficiently in guaranteed linear time[1]. The grammar we use will parse mathematical content into different classes of tokens including variables, different basic math operators, equal class, times class, fraction class and square root. And we choose to omit undefined control sequence for the sake of robustness.

When a grammar is reduced, the tokens is converted to a tree node directly or by attaching sub-trees which is reduced previously to the new root. In this way, we will finally get a tree structured representation.

Some operations may have commutative property, in these cases[2], all the sons in two adjacent levels will be attached to the same root.

2.2 Extraction of Branch words

After constructing a tree, A "branch word" is extracted by taking tokens from the leaves to the root of a tree in order. Branch words are used to be compared with those of other trees.

We notice that one math equation can use different symbol sets, so we choose not to distinguish the leaves' actual symbol in the branch word. We also notice the branch word is not enough to distinguish trees, an example would be the two equations $(a+b+c)\times(d+e)$ and $(a+b)\times(c+d+e)$, they have the same branch words yet have different semantic meaning in mathematical language. To avoid this flaw we further introduce the weight for any node n_i in a branch word, defined by the sum of that of its successors, which is

[2]In our system, the cases where we apply commutative property include *addition* and *multiplication* operations.

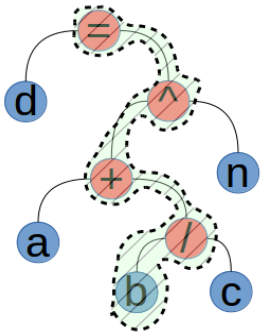

Figure 3: Illustration of One Branch Word for formula $(a+\frac{b}{c})^n = d$.

given by:

$$w(n_i) = \sum_n 1\,, \qquad n \in \{n_i \cup succ(n_i)\}$$

2.3 Indexing and Storage

The storage of mathematical formulas in our system, contains all the branch words from each tree as well as the weight information of each node in branch words. To enable efficient retrieval, branch words are stored in file system where the path is named by token names of a branch word (with weight information) in order[3]. The path where a branch word resides also stores a "posting" file recording all the documents in the collection which contain that branch word.

2.4 Comparing and Scoring

In our system, comparing two pieces of mathematical content is essentially to compare all the branch words from one content with those of the other, calculate the similarity degree between two branch words. In order to rank-order the documents matching a query, a score with respect to the query for each matching document is computed by sum the similarity degree for each related document in all the collection, then the system ranks all the related documents using the sum score.

For the detailed caculation, we put some notations here: Let m be the number of continuous weight matches between two branches from the beginning of the branch word. For one branch word b, let n_b be the number of same branch words from document that extracts b, l_b be the length of branch word b. Then for branch word i in query and branch word j in related document k. The formulas we use to calculate both relevance R between two branch words and the score S_k for document k are:

$$R_{i,j} = \alpha \cdot \min(n_i, n_j) \cdot \frac{m}{l_i} + \frac{1}{|n_i - n_j| + 1} \cdot \frac{m}{\max(l_i, l_j)} \quad (1)$$

$$S_k = \sum_i \sum_j R_{i,j} \quad (2)$$

[3]One example can be `./collection/var/frac/add`

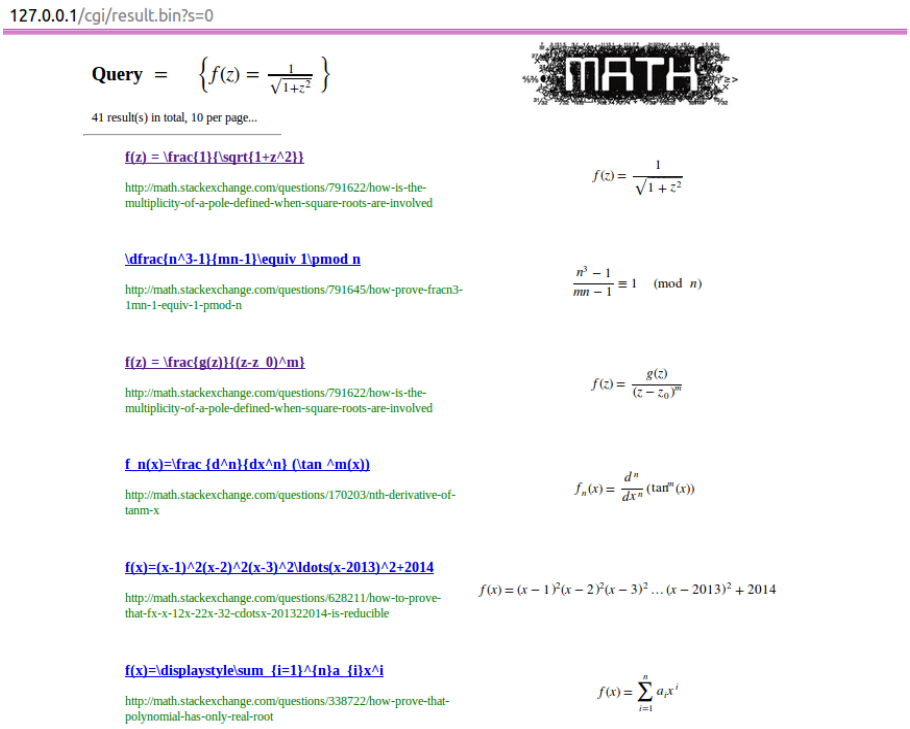

Figure 4: WEB interface for Our Prototype System

Where α is a large constant number to prioritize the first term in (1).

2.5 WEB front-end

The WEB interface contains a homepage for user to input query in an input box, just like a normal search engine does. The difference is, our interface accepts LaTeX as input and has a render preview for math equations. The WEB interface uses CGI program as a middle layer to both get user query (using *libcurl* library to unescape the URL encoding) and generate ranking page in HTML (by writing to the standard output) from the output file written by back-end program.

3. FUTURE WORK

Our current system does not distinguish variables and constant tokens from each other. Although this is good in cases a^2+b^2 and x^2+y^2 is treated as the same, but it is obviously not reasonable to let $n+\frac{1}{n}$ equal with $a+\frac{1}{b}$. Future work may involve unification algorithm[2] to better evaluate the relevance between formula symbols. Second, the resulting collection with input data from our crawler is not as good as that with the input data we manually choose. Part of the issue arises due to the ambiguity of high-level spaces[3]. Erroneous tight-binding spaces and failure in interpretation will result in blank token in our system, which may be deduced by our parser as *multiplication* operation. Moreover, statistical evaluation needs to be done on the effectiveness of our search results. But above all, the most important goal of future work is to explore the unrevealed potential of its usefulness.

4. CONCLUSIONS

In this paper, we introduce and present an experimental mathematical formula search system, also demonstrate the possibility of our approach which tries to utilize the structure of math formula to provide better search result for mathematical content.

Our prototype system is able to return relevant result in cases where the query uses a different set of symbol notations from the document, and a change-of-order for symbols that has mutable property. While there is a lot of future work ahead, the rich possibilities and huge potential for math-aware searching remain inspiring to us.

5. REFERENCES

[1] D. E. Knuth. On the translation of languages from left to right. *Information and Control*, 8(6), 1965.

[2] F. Baader and W. Snyder. Unification theory. *Handbook of Automated Reasoning*, I:447–533, 2001.

[3] Richard J. Fateman and Eylon Caspi. Parsing tex into mathematics. August 1989.