Math Expression Retrieval Using an Inverted Index Over Symbol Pairs

David Stalnaker and Richard Zanibbi[†]
Document and Pattern Recognition Lab
Department of Computer Science, Rochester Institute of Technology, USA
(davidstalnaker@google.com, rlaz@cs.rit.edu)

ABSTRACT

We introduce a new method for indexing and retrieving mathematical expressions, and a new protocol for evaluating math formula retrieval systems. The Tangent search engine uses an inverted index over pairs of symbols in math expressions. Each key in the index is a pair of symbols along with their relative distance and vertical displacement within an expression. Matched expressions are ranked by the harmonic mean of the percentage of symbol pairs matched in the query, and the percentage of symbol pairs matched in the candidate expression. We have found that our method is fast enough for use in real time and finds partial matches well, such as when subexpressions are re-arranged (e.g. expressions moved from the left to the right of an equals sign) or when individual symbols (e.g. variables) differ from a query expression. In an experiment using expressions from English Wikipedia, student and faculty participants (N=20) found expressions returned by Tangent significantly more similar than those from a text-based retrieval system (Lucene) adapted for mathematical expressions. Participants provided similarity ratings using a 5-point Likert scale, evaluating expressions from both algorithms one-at-a-time in a randomized order to avoid bias from the position of hits in search result lists. For the Lucene-based system, precision for the top 1 and 10 hits averaged 60% and 39% across queries respectively, while for Tangent mean precision at 1 and 10 were 99% and 60%. A demonstration and source code are publicly available.

Keywords: Mathematical Information Retrieval (MIR), Performance evaluation, Symbol layout trees

1. INTRODUCTION

A primary motivation for developing math search engines is to facilitate learning. Upon seeing an unfamiliar mathematical expression, a student might use the expression as a query in an effort to learn what it means.[29] Similarly, a researcher could find papers by searching for math contained within them,[30] or use an expression in query reformulation to locate papers with a similar expression. Math search could also be used when mathematical queries are detected in an existing search engine.[7] One problem for math search engines is that math representations are complex, and non-experts are often unfamiliar with them. To address this, tools have been developed that integrate handwriting recognition with search so that users can write, type and even import images to construct query expressions.[23,29]

While text search is well-studied, math search is in its early stages.[32] Even methods for evaluating math expression search results are still developing, as it is unclear what a 'good' result expression requires, and may be task-specific.[9,21] Searching for mathematical expressions is difficult for a number of reasons. There are numerous representations for expressions, including: LaTeX, Content and Presentational MathML, Mathematica, and various forms of rendered output (PDF, images). Mathematical expressions are generally expressed as a tree structure, which necessitates more complex matching algorithms than for text, which is linear. Effective metrics have been developed for ranking text search results (such as TF-IDF[22]), while ranking expression search results remains an open question. There are some analogs between the two problems: for example, the concept of term frequency can be roughly applied in the same way to symbols in an expression,[6] but we feel that a different strategy is needed. A summary of existing systems for query-by-expression are provided in Section 2.

The inverted index[34] remains the basis of most information retrieval systems.[14,22] Numerous adaptions have been developed to make the structure more powerful. By looking up multiple terms and taking the union or

[†]Corresponding author. Telephone: +1 585-475-5023

intersection of the returned set of documents, we can handle multi-term queries. By storing the frequencies of terms in each document, we can order the documents by this frequency. By storing the location(s) of each term in each document, we can search for exact and inexact phrases. In our approach, we apply inverted indices to math expression retrieval (*query-by-expression*), indexing expression structure directly rather than a linear string representation, which is the most common approach.[32] Our intention is to use small structural units, symbol pairs, in the hope that exact matching of these small pieces will allow for more relevant partial matches. Further details of our approach are presented in Section 3.

We believe that implementing a math expression search system using an inverted index on the layout of symbol pairs will: 1) yield more relevant results than text-based retrieval, and 2) be fast enough for real time use. We have conducted two experiments to test these assertions. First, a human study wherein we asked participants to score results from our system and a text search-based comparison system using a Wikipedia corpus,[33] and second, a performance analysis of our system. As part of this effort, we devised a new human protocol for evaluating relevance in search results based on the perceived similarity of results to the query expression. The human experiment yielded substantial, statistically significant increases in the perceived similarity of result expressions for our system, and our system is able to index and retrieve expressions from Wikipedia in real time, with many opportunities for optimization. Experimental results and their discussion are provided in Section 4. An online demonstration and source code for the *Tangent* search engine are available.*

2. RELATED WORK

Mathematical expressions can generally be considered to have a tree structure, unlike text which is linear. As such, the structure of the expression is important to capture in the methods used to select and rank matching expressions.

Text-Based Methods. There are several systems for math search that encode expressions as text and use existing text search systems for indexing and searching.[32] The key effort here is linearizing the math expression for input into the text search system. Zanibbi and Yuan's system[33] indexes individual LaTeX expressions by tokenizing the expression and mapping each symbol to a text representation. In this example from Miller,[15] the LaTeX expression `x^{t-2} = 1` is represented as:

```
x BeginExponent t minus 2 EndExponent Equal 1.
```

Expressions are then inserted into the Lucene search engine and queried with LaTeX expressions processed in the same manner. Mišutka and Galamboš[16] take a slightly different approach, linearizing expressions using a post-fix notation.

Sojka and Líška's MIaS system[25,26] operates on similar principles but differs in several areas. It is a full-text search system that indexes both math expressions and the surrounding text. Instead of single LaTeX expressions, MIaS indexes XHTML documents containing MathML expressions. The overall architecture of the system is similar, with linearized expressions being inserted into a text search system (it too uses Lucene). Munavalli and Miner's MathFind[17] is a similar text-search based system.

Kumar et al.[13] use the Largest Common Subsequence (LCS) to match linearized LaTeX strings. The input LaTeX strings are preprocessed so that each function, variable, and number is mapped to an atomic term, and variables and constants are generalized (represented by their type). A dynamic-programming LCS algorithm on these terms is then used to rank matches. This approach is fairly robust against small changes in structure. However the LCS algorithm is $O(n^2)$ in the expression size and requires a comparison with every expression in the index, which makes it unsuitable for large indexes.

The key problem with this approach is that text search has limited information on the structure of the expression. The advantages are that it works reasonably well in practice, is easy to implement, and benefits directly from decades of research in information retrieval. Our approach uses an inverted index, but indexes on pairs of symbols to better encode the expression structure.

*Demonstration: `http://saskatoon.cs.rit.edu/tangent/random`; Source code: `https://github.com/DPRL`

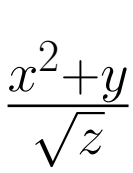
Tree-Based Methods. There have been several approaches to addressing the math expression retrieval problem more directly, making use of the tree structures that underly the appearance (symbol arrangement) and mathematical semantics (structure of operations) in expressions. Kamali and Tompa describe a system based on efficient exact matching of MathML trees.[8] They use an index in which identical subtrees are shared between all expressions, which allows for much better performance. Inexact matching is enabled through the use of wildcards, where each wildcard can be a number, variable, operator, or expression (which can itself contain wildcards). More recently, they describe a system that uses a similar index but uses a tree-edit distance for searching and ranking, which allows for inexact matching without user-specified wildcards.[10] Calculating the tree-edit distance for each expression in the index would be too expensive, but using a combination of early termination for poor matches and caching subexpressions, it achieves competitive performance (an average ~800ms query time on a large corpus compared to ~300ms for MIaS).

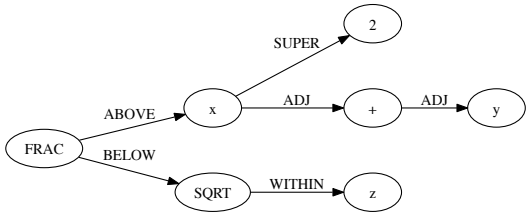
Another approach to the problem uses substitution trees.[3] Substitution trees come from the field of automatic theorem proving. The leaf nodes contain the expressions that have been inserted into the tree. The internal nodes contain expressions with at least one generic term. Each child represents a substitution of one or more of the generic terms of its parent with more specific terms (which can introduce new generic terms). Kohlhase and Sucan's Math WebSearch[12] uses substitution trees to match expressions on their semantic meaning, rather than layout. This approach can match expressions that match the query term exactly up to α-equivalence, which means the structure must match exactly but individual terms may be substituted by terms of the same type. In addition, subexpression matching was enabled by inserting all subexpressions of an expression when inserting it into the index. Input to Math WebSearch is Content MathML (representing the applications of operators to operands, i.e. the operator tree for an expression). Schellenberg et al.'s work[24] instead uses substitution trees to match expressions by layout. Rather than performing an exact expression match (as Math WebSearch does), this system exhaustively searches the tree for matches of a given size. This has the benefit of allowing for partial matches that have structural similarity, but the drawback of greatly increased query times.

An emerging approach uses multiple local structural representations rather than the complete tree structure. Nguyen et al. have created a system for expression retrieval that converts local sub-expressions in Content MathML trees to words for individual arguments and local operator-argument pairs.[18,19] A lattice over the sets of generated words for each expression is then used to define similarity, and a breadth-first search is used to construct a neighbor graph during retrieval. For the first international competition on mathematical information retrieval,[1] Hiroya and Saito constructed a system using an inverted index over paths of node types from the root note of an operator tree represented in MathML to each operator and operand of the expression.[5] Reported expression retrieval performance was brittle, likely due to requiring matches from the root of the expression. Our system has some similarities to this approach, but differs in that we use specific symbols rather than types, and use the relative position of symbol pairs rather than complete paths between symbols for indexing.

Ranking. Symbols in an expression can be broken into four categories: operators, variables, constants, and functions. These symbol categories may have different weights in the ranking function (e.g. operators have more meaning than variables, which can be renamed without meaningfully changing the expression).[31] Relatedly, a useful quality for a ranking function would be to match expressions with identical structure but variables renamed, with a penalty. Another positive quality for a ranking function would be to favor results in which the matched symbols are more connected to each other than not. For example, take the query $x + y$. We feel that the result $(x + y) * z$ would be a better match than $(x + z) * y$, because the query term is an exact subexpression in the first result. From this perspective, the more relevant result would thus be the one that contains symbols that are closest to an exact subexpression.

While many systems rely on underlying text search systems for ranking partial matches, there have been some attempts made at ranking math expressions directly. Schellenberg et al.[24] define a ranking function combining two metrics: a bag-of-words comparison of the individual symbols in the expressions and a bipartite comparison that looks at pairs of symbols (including the structure between them: the relationship between the symbols and position along the baseline). For each of these metrics, the number of matching symbols / pairs is counted, and the average of these two scores is taken. Additionally, symbols that match the correct type but not the exact symbol are counted at 25% of a full match. Our ranking functions were designed based on this approach.





Parent	Child	Dist.	Vert.
FRAC	x	1	1
FRAC	2	2	2
FRAC	+	3	1
FRAC	y	3	1
FRAC	SQRT	1	-1
FRAC	z	2	-1
x	2	1	1
x	y	2	0
x	+	1	0
+	y	1	0
SQRT	z	1	0

(a) Expression (b) Symbol Layout Tree (c) Symbol Pair Tuples

Figure 1: Symbol layout representations. Tuples (c) are defined for every descendant of a symbol in the symbol layout tree (b). In (c), *Dist.* is the path length from the parent symbol to the child symbol in the symbol layout tree, and *Vert.* is the sum of vertical displacements along this path: +1 for each superscript/above edge, -1 for each subscript/below edge, and 0 for horizontally adjacent/within a square root edge.

3. METHODOLOGY

Our proposed method is a query-by-example technique, retrieving expressions similar to a query expressed in LaTeX or (Canonical) Presentation MathML.[2] An inverted index maps a pair of symbols in a particular spatial arrangement to the set of documents containing it. The representation of spatial arrangement is relative, given by the distance from the leftmost/dominant symbol to the other symbol in a Symbol Layout Tree (SLT) (see Figure 1), and change in baseline position from the first to the second symbol. Matching expressions are then ranked using the sets of matching, query and candidate symbol pair sets.

In this section, we describe Symbol Layout Trees (SLTs), our representation for the relative positions of symbols in an SLT, five different ranking functions that may be used to order matching expressions, and summarize our implementation.

3.1 Symbol Layout Representation

Internally, Tangent uses a Symbol Layout Tree (SLT) to represent a query expression. The nodes in this tree are the symbols in the expression and the edges are the spatial relationships between them (see Figure 1). The tree is rooted at the leftmost symbol on the main baseline. Similar to LaTeX, each symbol can have a relationship with symbols above/superscript, below/subscript, adjacent and within (for square roots). In some SLT representations above and superscript relationships are distinguished, but we have combined them to allow for more robust partial matching. We take a similar approach with below and subscript relationships. Fractions are encoded as a FRAC symbol with the numerator ABOVE and the denominator BELOW. A square root can have an expression WITHIN it, and most other symbols will be ADJACENT (at right).

In text information retrieval, the documents are split into words to be inserted into an index - to preserve structural information, we instead insert *pairs* of math symbols into the index along with their relative positions. This symbol pair representation is a tuple $(s1, s2, d, v)$ where s_1 and s_2 are the two symbols in the pair. The distance, d, is the length of the path between s_1 and s_2 when traversing the SLT. It can also be thought of as the number of symbols visited while moving from s_1 to s_2, moving away from the root of the tree. Often this is from left to right, with the exception of fractions and other vertical structures (e.g. summations and integrals). The vertical displacement, v, is the change in baseline position from s_1 to s_2. This increases for ABOVE and SUPER (superscript) relationships, and decreases for BELOW and SUBSC (subscript) relationships. Figure 1 provides an example. There are a worst-case $\binom{n}{2}$ pairs inserted for an expression with n symbols, which occurs for linear expressions (i.e. when there are no branches in the SLT). This symbol pair representation is not well-defined for tables, so currently MathML expressions containing tables are ignored. We also have not supported the relatively rare `mmultiscripts` tag, which allows for scripting at left of a symbol, such as for ${}_NC_2$. Extensions to accommodate these structures were later developed in follow-on work.[20]

Table 1: Ranking Functions. Q, R, and M are symbol pairs in the query, result candidate, and match (intersection) of the query and result, respectively. Each ranking function is a variant of the F-Measure. *d():* symbol pair distance; *ief()* symbol pair inverse expression frequency; *lcp():* symbol pair set for largest common prefix.

Ranking Function	$Definition$	Ranking Function	$Definition$
F-Measure	$\frac{2\|M\|}{\|Q\|+\|R\|}$	Recall-biased	$\frac{(1+1.5^2)\|M\|}{1.5^2\|Q\|+\|R\|}$
Distance	$\frac{2\sum_{m\in M}\frac{1}{d(m)}}{\sum_{q\in Q}\frac{1}{d(q)}+\sum_{r\in R}\frac{1}{d(r)}}$	Inverse Expr. Freq.	$\frac{2\sum_{m\in M}\text{ief}(m)}{\sum_{q\in Q}\text{ief}(q)+\sum_{r\in R}\text{ief}(r)}$
Largest Comm. Prefix	$\frac{2 \mid lcp(M) \mid}{\|Q\|+\|R\|}$		

3.2 Inverted Index for Symbol Pairs

Indexing. The inverted index we have developed is similar to that of a text search engine,[34] using symbol pairs instead of words. The index is a hash table mapping symbol pairs to the list of expressions that contain them. To create the index, we convert each expression from LaTeX or Presentation MathML to Canonical MathML, generate symbol pairs as described above, and look up each symbol pair in the index. We then append the current expression to the list of expressions for the symbol pair. In text search, the inverted index often includes the position in a document where each word occurs. Analogously, we can annotate each expression added to the index for a symbol pair with an identifier giving the absolute position of the second symbol (s_2) in a symbol pair relative to the root of its symbol layout tree. This identifier is a d-element list, where d is the depth of the symbol in the symbol layout tree. The ith element in this list is 0, 1, 2, or 3, representing the ith spatial relationship on the path to s_2 for BELOW/SUBSC, ABOVE/SUPER, ADJACENT, or WITHIN, respectively. Because d tells us the path length between s_1 and s_2, we can later calculate the identifier for s_1 by removing the last d elements of s_2's identifier.

Querying. To query, we construct a second hash table from each matching expression to the list of symbol pairs it has in common with the query. We first look up each symbol pair from the query in the inverted index to obtain the expressions containing the pair (recall that if an expression contains a symbol pair multiple times, it will appear multiple times in the inverted index entry for that pair). Then, for each expression in that list, we add the pair to the hash table entry for the expression. The full list of matched expressions is then ordered by a ranking function, which is given the query symbol pair set, and the constructed table from matched expressions to symbol pairs. When a symbol pair occurs more than once in either the query or a matching expression, the minimum number of occurrences is used in determining the match size. For example, if a symbol pair occurs five times in a result but only twice in the query, we will count it twice.

3.3 Ranking Functions

We have developed several ranking functions, presented in Table 1. F-Measure is the baseline ranking function, balancing between recall and precision of the match. The remaining ranking functions are modifications of the F-Measure, usually through weighting symbol pairs. We describe each ranking function below.

1. F-Measure. The recall-of-match is the percentage of symbol pairs in the query that are in the match, and the precision-of-match is likewise the percentage matched in the result. The F-Measure is the harmonic mean of the recall and precision of the match: $\frac{2|M|}{|Q|+|R|}$, where Q, R, and M are the sets of pairs in the query, result candidate, and match (between the query and result) respectively.

2. Recall-Biased. A modified F-Measure which weights recall more highly than precision. The rationale is that users may prefer partial matches containing more of the query as subexpressions over smaller matches (i.e. that are more precise), but which contain less of the query.

3. Distance. Each symbol pair p is weighted by the inverse distance (path length) between its two symbols ($1/d(p)$), for the match (M), query (Q) and results (R).

4. Inverse Expression Frequency. Adapts the common text similarity metric of TF-IDF (term frequency - inverse document frequency[22]) to the math domain. Instead of using symbol pair set sizes, we instead sum the inverse expression frequency (IEF) of each pair in the matched, query and result sets.

5. Largest Common Prefix. Finds an alignment between the query and result and only includes pairs along this alignment. To do this, we use additional path information associated with each pair in the index. Specifically, for each pair in the match, we get the path to the parent symbol in the query and result separately. For each matching symbol pair in the query and a result, we find where the sequence of spatial relationships diverge, searching from the parent symbol toward the roots of the layout trees. For space we omit further details of this ranking function, which are available elsewhere.[28]

3.4 Implementation

The implementation of our system is designed around the Redis[†] key-value store. Redis is a robust, scalable, in-memory database that maps string keys to strings, lists, sets, sorted sets, and hash tables (all of strings). It was chosen to ease development and for its high performance. The index and all supporting information is stored in Redis. As Redis is a simple key-value store, each type of information encoded using a different key syntax; for example, the key `pair:[p]:exprs` contains the list of expressions containing the pair p.

The Tangent search engine is implemented in Python. There is a core library (approximately 1000 lines of code), which defines the data structures (SLT and symbol pairs), a MathML parser, and a RedisIndex class with functionality for inserting and retrieving expressions. Each ranking method is defined by a static class containing a ranking function and properties that tell the index which information to fetch from Redis. This allows the ranking function to be chosen on the fly, using the same index.

Building on this, there is a command-line tool for both indexing and searching. The primary interface for searching is a web interface built using Flask. This simple web server receives a request containing the query expression and uses the RedisIndex from the core library to perform a search using the algorithm described in Section 3.2. The index then returns 10 result expressions, each with a list of links to articles containing the expression, which are then displayed to the user.

4. EXPERIMENTS

We ran two experiments. The first compared search results between our system and another, and the second was to evaluate the performance characteristics of our system. In this section, we describe the corpora used for development and experimentation, the design of the experiments, and a presentation and discussion of the results.

MREC (Math REtrieval Collection). MREC is a collection of approximately 324,000 academic publications. The arXiv contains preprint papers in science and mathematics. These documents have been converted to XHTML and MathML by the LaTeXML project. MREC was used in the development phase of the project.

Wikipedia. The Wikipedia corpus contains nearly every mathematical expression from the English-language Wikipedia project. It is a desirable corpus because the information it contains is helpful in learning mathematics, which is one of the major use cases of a math search system. Due to the number of expressions, for many queries there are likely a number of similar/relevant expressions in the collection.

Our Wikipedia corpus was assembled from a full XML archive of English Wikipedia created on May 4, 2013. Math expressions delimited by `<math>` tags were extracted from the HTML pages along with their associated articles. Expressions are represented in LaTeX, which we converted to Canonical Presentation MathML[2] using LaTeXML. Initially, 482,364 expressions were extracted from 32,780 articles. Expressions containing matrices, tables or pre-subscripts/superscripts were removed (6126 expressions), leaving 476,238 expressions in the corpus.

[†]http://redis.io, Flask: http://flask.pocoo.org, arXiv: http://arxiv.org, LaTeXML: http://dlmf.nist.gov/LaTeXML

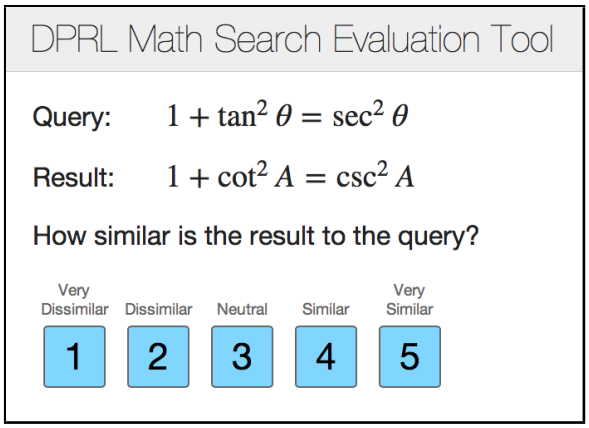


Figure 2: Evaluation tool used by participants.

4.1 Human Evaluation of Search Results

We conducted a human study to evaluate the search results produced by our system. To do this, we compared our system's search results with those produced by an existing text-search-based system constructed using Lucene[33] as a benchmark. We did not have access to recent systems supporting structure-based matching at the time of the experiment.[6, 10, 11, 19, 27]

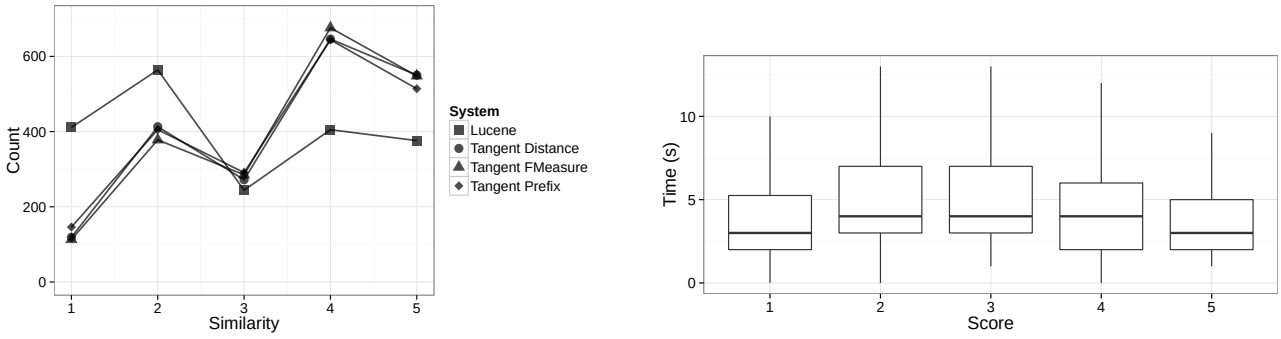
It is difficult to evaluate relevance for query results automatically. We asked the participants to score by similarity rather than relevance because relevance is dependent on the search task, which is beyond the scope of this experiment. The experiment asked participants to score the top ten unique results from each system for ten queries. The participants were shown one query and result and asked "How similar is the result to the query?" They were instructed to answer on a 5-point Likert scale (see Figure 2).

Our Wikipedia corpus was used for the experiment. The queries were chosen by randomly sampling a larger set of queries from the corpus, and picking from this ten that represented diversity in size, type of structure, and field (see Table 2). Due to time constraints in the experiment, we were unable to compare all of the ranking functions we developed. We conducted an informal experiment beforehand and determined that the best-performing ranking functions were F-Measure, Distance, and Prefix. Notably, the IEF ranking function performed worst; this analog of TF-IDF in text search does not appear to be useful. The experiment was thus a comparison between Tangent with these three ranking functions and the Lucene-based system (henceforth referred to as Lucene).

The experiment was run by the first author in a controlled setting (a quiet room with a desk and computer), one participant at a time using a web-based evaluation tool (see Figure 2). Search results were pre-computed to avoid response time affecting participant ratings. To avoid order effects, query and result presentation were randomized for each participant: all results for a single query were presented sequentially in a random order, with the query order randomized. Students and faculty from the Computing and Science colleges at our institution were recruited through email and posters, with the expectation that this group may find math search useful.

Table 2: Queries Used in the Experiment

No.	Query	No.	Query
1.	$\widetilde{\rho}$	6.	$\int_a^b f(x)\,dx = F(b) - F(a).$
2.	$\bar{u} = (x, y, z)$	7.	$\left(1/6,\ \sqrt{1/28},\ -\sqrt{12/7},\ 0,\ 0,\ 0,\ 0,\ 0\right)$
3.	$1 + \tan^2\theta = \sec^2\theta$	8.	$\sum_{i=m}^{n} a_i = a_m + a_{m+1} + a_{m+2} + \cdots + a_{n-1} + a_n.$
4.	$cos(\theta_E) = e^{-TR/T_1}$	9.	$f(x;\mu,c) = \sqrt{\frac{c}{2\pi}}\ \frac{e^{-\frac{c}{2(x-\mu)}}}{(x-\mu)^{3/2}}$
5.	$a = g\frac{m_1 - m_2}{m_1 + m_2}$	10.	$D4\sigma = 4\sigma = 4\sqrt{\frac{\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} I(x,y)(x-\bar{x})^2\,dx\,dy}{\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} I(x,y)\,dx\,dy}}$



(a) Ratings by system

(b) Response times by rating score

Figure 3: Likert rating counts by system (a) and Response Times by Rating Score (b). There are 2000 evaluations in total (10 queries; 10 results/query; 20 participants/result).

Previously it was shown that search result ordering affects how likely hits are to be identified as relevant.[4] We presented search results one-at-a-time to avoid similarity ratings being affected by hit list position. Presenting hits one-at-a-time has the additional benefit of allowing expressions returned by multiple search engines to be evaluated once, reducing participant effort. After deduplication, the participants were asked to score 214 results, ten of which were for the familiarization exercise.

Before the experiment, participants were asked to fill out a short demographics survey. Then, they completed a short familiarization exercise, wherein they scored five results for each of two queries. After this task was complete and any questions were answered, they began the experiment. In addition to the Likert ratings, we recorded the time taken by the participant for each query. At the end of the experiment, participants were asked to rate the difficulty of the task and describe their scoring criteria. The task took approximately 30 minutes, and participants were paid $10 for their time.

4.1.1 Similarity Evaluation Results

Demographics and Surveys. 20 students and professors participated in the experiment. 15 (75%) of the participants were male, and 5 (25%) were female. 15 (75%) were between the age of 18 and 24, three (15%) were 25-34, and one (5%) was 35-44. All participants were from fields in science and technology, with 13 (65%) in computing, 5 (25%)in science, and 2 (10%) in mathematics.

1 participant (5%) found the task very difficult, 11 (55%) found it difficult, 7 (35%) were neutral, and 1 (5%) found it easy. When asked to describe how they evaluated the results, 17 (85%) mentioned using visual similarity and 10 (50%) mentioned semantic meaning. The additional comments mostly described how difficult the users found the task, which aligned with their ratings.

Similarity Ratings. Figure 3(a) provides the distribution of all ratings by system. We can see that the peaks for the Lucene and Tangent distributions shift from a pronounced peak for ratings of 2 ('Dissimilar') for Lucene, to a larger peak rating of 4 ('Similar') for all the Tangent variants. Further, the number of 5 ('Very Similar') ratings nearly match the number of 'Dissimilar' ratings in the Lucene system. To our surprise, the simplest ranking function for Tangent performed slightly better than the Distance and Largest Common Prefix rankers in terms of raw ratings (however, this difference is not statistically significant).

Likert scale data is ordinal rather than nominal (numeric), and whether it is valid to use Likert data as numeric values for statistical analysis (e.g. in an analysis of variance (ANOVA)) remains an open question. We instead binarize the scores to values of either relevant (scored similar or very similar) or not relevant (scored neutral to very dissimilar). After this, we can calculate precision-at-k for the top-10, top-5, and top-1 results.

A two-way ANOVA comparing the mean top-10 precision ratings for system vs. query shows a very strong effect for both the system ($p < 2.2 * 10^{-16}$) and the queries ($p < 8.9 * 10^{-16}$). There was no interaction effect found between the system and query ($p < 0.205$). Running pairwise t-tests (using the Bonferroni correction for

multiple comparisons), we see that there is a significant difference between Lucene and each of the three Tangent variants (all with $p < 10^{-13}$). The t-tests did not show a significant difference between any of the different ranking functions for Tangent. The three different rankings returned a very large set of common results for each query, which largely explains this.

In terms of precision-at-k, for the ten queries Lucene had a mean precision-at-10 of 39% ($\sigma = 18\%$), mean precision-at-5 of 47% ($\sigma = 18\%$) and mean precision-at-1 of 60.0% ($\sigma = 39\%$). Tangent using the F-measure ranking had mean precision-at-10 of 60% ($\sigma = 16\%$), mean precision-at-5 of 75% ($\sigma = 16\%$), and mean precision-at-1 of over 99% ($\sigma = 2\%$) (the first result always matched the query). At all three depths, the new system has higher precision, with lower variance, particularly for precision-at-1. Both systems performed poorly for query 10, which is a large expression (mean top-10 precisions of 26% for Lucene and 33% for Tangent). The top-10 results produced by each algorithm are available, along with the similarity ratings for each result.[28]

Response Times. As some hits were shared between systems and displayed to each user only once, the time taken to evaluate a hit was counted identically for each system producing the hit. An ANOVA test on timings by system showed that with high confidence ($p < 0.00007$) there is a significant difference between the systems. From a t-test post-hoc (with Bonferroni corrections), there is a difference between Lucene and each of the three Tangent systems, with p-values of 0.018, 0.0037, .000053 between Lucene and Distance, F-Measure, and Prefix respectively. The mean response time for Lucene was 5.84 seconds ($\sigma = 5.81$), whereas the mean response time for F-measure was 5.29 seconds ($\sigma = 4.68$). From this, we see that evaluating search results was slightly faster for Tangent than Lucene.

As can be seen in Figure 3(b), participants took longest to score the expressions that were not obviously similar or dissimilar. This intuitively makes sense, as less thought is required if the expressions are identical or vastly dissimilar.

4.2 System Performance (Time and Space)

Indexing time was tested using the full Wikipedia corpus. Retrieval time was tested using the ten queries from the experiment and the same corpus. We treated this as a binary test: either the system is fast enough to be used in real time (queries running in under approximately 3 seconds), or it is not. We measure execution speed using system clock time, which has the drawback of introducing noise into our timings and being hardware-dependent, but is sufficient for assessing response time. As indexing speed is only important for the initialization of the system, it is the less important of the two quantities being measured. All timing tests were performed on a server in our department with 2 Intel Xeon CPUs (2.93GHz) and 96GB RAM. In addition to execution time, we also wished to observe differences in index size between Lucene and our system.

As Lucene is a mature application, it is faster for both indexing and searching, and the index is much smaller than our system. Still, while Tangent is clearly slower than Lucene, it is not unusably slow. Indexing the Wikipedia dataset took 53 minutes - much slower than Lucene's 7 minutes 44 seconds, but workable for a one-time task. The average query time for the 10 queries used in the experiment is ~1.5 s with a standard deviation of ~1 s. This is well within our stated goal of being fast enough to be used in real time. Query time is dependent on the size of the query and number of matches, but the even the largest (slowest) queries we tested took under 3 seconds. Tangent's index for the Wikipedia dataset used 6.19 GB of memory while Lucene's on-disk index was a much smaller 107 MB. This is because our current indexing algorithm uses raw strings for storage (rather than enumerations), and we make no attempt to compress the index.

4.3 Discussion

From our results, we can conclude that overall the expressions returned by our system were rated by participants as significantly more similar to the queries than those returned by the Lucene system, and participants were able to make similarity assessments more quickly for our system than Lucene. Looking at the scores by query, we can place queries in two categories: five where Tangent and Lucene score roughly as well as each other (for example, see Table 3), and five where Tangent scores much higher. While Lucene sometimes can find good matches, Tangent's results are more consistent. Our findings are also in line with recent investigations that suggest structural similarity matching is better for query-by-expression than text-based retrieval models.[6,9]

Table 3: Top 10 results for $1 + \tan^2\theta = \sec^2\theta$ for Lucene and Tangent (F-Measure Ranking)

Rank	Lucene	Tangent	Rank	Lucene	Tangent
1.	$1 + \tan^2\theta = \sec^2\theta$	$1 + \tan^2\theta = \sec^2\theta$	6.	$\sin^2\theta + \cos^2\theta = 1$	$\sqrt{1 + \tan^2\theta_o}$
2.	$\tan^2\theta + 1 = \sec^2\theta$	$1 + \tan^2 y = \sec^2 y$	7.	$\cos^2\theta + \sin^2\theta = 1$	$\pm\sqrt{1 + \tan^2\theta}$
3.	$\sec^2\theta = 1 + \tan^2\theta$	$\frac{d}{d\theta}\tan\theta = \sec^2\theta$	8.	$1 + \cot^2\theta = \csc^2\theta$	$1 + \cot^2 A = \csc^2 A$
4.	$1 + \tan^2\theta = \sec^2\theta$ and $1 + \cot^2\theta = \csc^2\theta$.	$1 + \cot^2\theta = \csc^2\theta$	9.	$\cot^2\theta + 1 = \csc^2\theta$	$1 + \cot^2 y = \csc^2 y$
5.	$\cos^2\theta + \sin^2\theta = 1$,	$\sec^2\theta = 1 + \tan^2\theta$	10.	$x = r\cos\theta = 2a\sin^2\theta = \frac{2a\tan^2\theta}{\sec^2\theta} = \frac{2at^2}{1+t^2}$	$\tan^2\theta + 1 = \sec^2\theta$

Table 3 illustrates some differences in how the two systems perform matching. Tangent prefers tight matches, and does not (directly) consider term frequencies as the Lucene system does. This may partly explain the faster evaluation times for our system, as there are often larger structures from the query that can be visually matched in the results. The ranking functions for our system (F-Measure, Distance, Largest Common Prefix) largely returned the same expressions, with small differences. One possible explanation for the success of the simple F-Measure ranking is that counter to our intuitions, preferring local to distant symbol pair matches does not seem to have been beneficial. A good example of this are parentheses - matching parentheses at a fixed distance along a baseline matches expressions with a similar number of arguments between parentheses, even when the arguments differ (e.g. for Query 2). This structural matching allows us to find expressions that are identical (or very similar) to the query but have renamed symbols (second hit in Table 3).

While much slower than Lucene, our system is viable for real-world use without any optimizations. Both indexing and speed are sufficiently fast. The current high memory usage necessitates a modern computer that can address more than 4 GB of memory, but most current computers are capable of running Tangent well. The size of the index could be greatly reduced by removing the Largest Common Prefix ranker, as the paths for it comprise much of the memory usage, and its addition did not improve the quality of returned results. Index compression in particular could be a great boon to speed as well as memory usage, because the majority of the query time is spent transferring postings (matching symbol pairs) from the Redis server to the client. It is likely that an optimized version of our system could be comparable to Lucene in memory usage, indexing speed, and query speed, and we propose optimizations below. It is possible to informally compare our query speed with Kamali and Tompa's results.[10] In an index that is roughly twice the size as ours, their system is roughly twice as fast (~800ms vs ~1500ms on average per query). Sojka and Líška's MIaS[25] is even faster, at ~300ms. We believe that one could create a system that as fast as these other systems by optimizing our current framework.

Performance Optimizations. Storing the document references in a compressed form (e.g. using Elias' gamma code or a bytewise representation) could significantly reduce the index size, as would storing the differences between document identifiers rather than the identifiers themselves.[14] This would substantially improve query time, as the majority of the current query time is spent transferring postings from the server to the client. Another improvement would be to implement the system in a higher-performance language than Python. However, most query time is spent reading data from Redis (which calls C libraries), so the improvement would be minor. Another possibility is to store the index directly in memory, rather than indirectly through Redis. We briefly experimented with both approaches (using the Go language). Surprisingly, query times were similar between the current Python-Redis and in-memory Go implementations. Still, it is possible that implementation-specific speedups may be made.

Our method can also be parallelized/distributed. For an index the size of Wikipedia, a load-balanced set of servers running the entire index would allow the system to scale to more users, albeit without improving query time. For larger indexes, performance (query time) starts to become unacceptable, and more work would be needed to distribute the index itself across servers.

Retrieval Model Improvements. One limitation of Tangent is that it does not allow for substitution of symbols, although often surrounding structures can compensate for this (e.g. as seen in Table 3). A separate substitution index could be created where one or both of the symbols are replaced by a type (e.g. VARIABLE, OPERATOR, CONSTANT). A number of existing math retrieval systems use such information in their

indices,[5,13] including some incorporating query languages that support wildcards for variables and subexpressions.[8,12,15] However, the lists in such an index would be large, and the effect on performance unknown. A related improvement would be to add support for synonymns (e.g. cos and cosine).

Matching symbol distances precisely may not be helpful at large distances (e.g. eight or nine symbols apart). We might consider binning distances at larger intervals, or to allow matches of symbols within a range of distances (e.g. ± 1 symbol). Also, our baseline offset measurements are lossy. If an ABOVE relationship is followed by a BELOW relationship, the offset will be 0, the same as two ADJACENT relationships. This occurs for the (x, z) pair in x^{y_z}. We could store a list of baseline changes for each pair, but this increases the index size.

There are two issues with the way we generate the symbol pairs. First, we do not generate symbol pairs between pairs that are on separate branches of the SLT. For example, there is no pair between the 2 and y in $x^2 + y$. Fixing this would require major changes to the symbol pair representation, and these pairs tend to represent weak semantic relationships, and so might not be helpful. The second issue is that the number of pairs increases quadratically with the size of a (linear) subexpression. While the unintentional effect of preferring well-connected matches seems to be positive, it may not be ideal. We might for example weight each symbol pair inversely proportional to the number of other symbol pairs that use either of the symbols in the pair.

Currently, Tangent does not support tables or matrices. One simple solution would be to simply index each cell separately, as its own expression, and prefer matches that best cover the subexpressions. Another approach would be to modify the SLT and symbol pair definitions to represent this structure directly. We also currently do not support pre-superscripts or pre-subscripts, but this can be added by extending our SLTs to include these relationships (e.g. using negative distances in symbol tuples).[13]

5. CONCLUSION

Our novel approach to math expression retrieval yields high quality search results and does so efficiently. We have also introduced a new rigorous method for human evaluation of results in math expression retrieval. Potential improvements include optimization of the inverted index, modifications to incorporate matrices and other tabular structures, support for wildcards, and the integration of our query-by-expression technique with a text-based search.[18] In follow-on work, our group addressed each of these issues and produced a system that obtained the best results for the NTCIR-11 Math-2 Wikipedia subtask.[20]

ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Grant No. IIS-1016815.

REFERENCES

[1] A. Aizawa, M. Kohlhase, and I. Ounis. NTCIR-10 math pilot task overview. In *Proc. NII Testbeds and Community for Information access Research (NTCIR)*, pages 654–661, Tokyo, Japan, 2013.

[2] D. Archambault and V. Moço. Canonical MathML to simplify conversion of MathML to Braille mathematical notations. *Computers Helping People with Special Needs*, pages 1191–1198, 2006.

[3] P. Graf. *Term Indexing*, volume 1053 of *LNCS*. Springer, 1995.

[4] Z. Guan and E. Cutrell. An eye tracking study of the effect of target rank on web search. In *Proc. ACM SIGCHI*, pages 417–420, 2007.

[5] H. Hiroya and H. Saito. Partial-match retrieval with structure-reflected indices at the NTCIR-10 math task. In *Proc. NII Testbeds and Community for Information access Research*, pages 692–695, Tokyo, Japan, June 2013.

[6] X. Hu, L. Gao, X. Lin, Z. Tang, X. Lin, and J. B. Baker. Wikimirs: A mathematical information retrieval system for Wikipedia. In *Proc. Joint Conf.Digital Libraries (JCDL)*, pages 11–20, 2013.

[7] S. Kamali, J. Apacible, and Y. Hosseinkashi. Answering math queries with search engines. *Proc. Int. Conf. World Wide Web (WWW)*, pages 43–52, 2012.

[8] S. Kamali and F. W. Tompa. A new mathematics retrieval system. *Proc. Int. Conf. Information and Knowledge Management (CIKM)*, pages 1413–1416, 2010.

[9] S. Kamali and F. W. Tompa. Retrieving documents with mathematical content. In *Proc. SIGIR*, pages 353–462, Dublin, Ireland, Aug. 2013.
[10] S. Kamali and F. W. Tompa. Structural similarity search for mathematics retrieval. *Proc. Conf. Intelligent Computer Mathematics (CICM)*, pages 246–262, 2013.
[11] M. Kohlhase, B. A. Matican, and C.-C. Prodescu. MathWebSearch 0.5: Scaling an open formula search engine. In *Proc. AISC/MKM/Calculemus*, volume 7362 of *LNCS*, pages 342–357. Springer, 2012.
[12] M. Kohlhase and I. Sucan. A search engine for mathematical formulae. *Proc. Conf. Artificial Intelligence and Symbolic Computation (AISC)*, pages 23–24, 2006.
[13] P. Kumar, A. Agarwal, and C. Bhagvati. A structure based approach for mathematical expression retrieval. In *Multi-disciplinary Trends in Artificial Intelligence*, volume 7694 of *LNCS*, pages 23–34. Springer, 2012.
[14] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge Univ. Press, 2008.
[15] B. R. Miller and A. Youssef. Technical aspects of the digital library of mathematical functions. *Annals of Mathematics and Artificial Intelligence*, 38(1):121–136, 2003.
[16] J. Mišutka and L. Galamboš. Extending full text search engine for mathematical content. In *Towards Digital Mathematics Library (DML)*, pages 55–67, 2008.
[17] R. Munavalli and R. Miner. MathFind: a math-aware search engine. In *Proc. SIGIR*, page 735, 2006.
[18] T. T. Nguyen, K. Chang, and S. C. Hui. A math-aware search engine for math question answering system. In *Proc. Int. Conf. Information and Knowledge Management (CIKM)*, pages 724–733, 2012.
[19] T. T. Nguyen, S. C. Hui, and K. Chang. A lattice-based approach for mathematical search using formal concept analysis. *Expert Systems with Applications*, 39(5):5820 – 5828, 2012.
[20] N. Pattaniyil and R. Zanibbi. Combining TF-IDF text retrieval with an inverted index over symbol pairs in math expressions: The Tangent math search engine at NTCIR 2014. In *Proc. 1tth NII Testbeds and Community for Information access Research (NTCIR)*, Tokyo, Japan, 2014. (to appear).
[21] M. Reichenbach, A. Agarwal, and R. Zanibbi. Rendering expressions to improve accuracy of relevance assessment for math search. In *Proc. ACM SIGIR*, pages 851–854, Gold Coast, Australia, 2014.
[22] G. Salton and M. J. McGill. *Introduction to modern information retrieval*. McGraw-Hill, 1983.
[23] C. Sasarak, K. Hart, R. Pospesel, D. Stalnaker, L. Hu, R. Livolsi, S. Zhu, and R. Zanibbi. m_{in} : A Multimodal Web Interface for Math Search. In *Symp. Human-Computer Interaction and Information Retrieval*, Cambridge, MA, 2012. `http://ils.unc.edu/hcir2012/hcir2012_submission_11.pdf`.
[24] T. Schellenberg, B. Yuan, and R. Zanibbi. Layout-based substitution tree indexing and retrieval for mathematical expressions. *Proc. Document Recognition and Retrieval XIX*, pages OI: 1 –8, Jan. 2012.
[25] P. Sojka and M. Líška. Indexing and Searching Mathematics in Digital Libraries. In *Intelligent Computer Mathematics*, volume 6824 of *LNCS*, pages 228–243. Springer, 2011.
[26] P. Sojka and M. Líška. The art of mathematics retrieval. *Proc. Document Engieneering (DocEng)*, pages 57–60, 2011.
[27] C. Sombattheera, N. Loi, R. Wankar, T. Quan, P. Pavan Kumar, A. Agarwal, and C. Bhagvati. A structure based approach for mathematical expression retrieval. In *Multi-disciplinary Trends in Artificial Intelligence*, volume 7694 of *LNCS*, pages 23–34. Springer, 2012.
[28] D. Stalnaker. Math expression retrieval using symbol pairs in layout trees. Master's thesis, Rochester Institute of Technology, Rochester, NY, USA, Aug. 2013.
[29] K. Wangari, R. Zanibbi, and A. Agarwal. Discovering real-world use cases for a multimodal math search interface. In *Proc. ACM SIGIR*, pages 947–950, Gold Coast, Australia, 2014.
[30] A. Youssef. Roles of math search in mathematics. In *Proc. Mathematical Knowledge Management (MKM)*, volume 4108 of *LNCS*, pages 2–16. Springer, 2006.
[31] A. Youssef. Methods of relevance ranking and hit-content generation in math search. In *Towards Mechanized Mathematical Assistants*, volume 4573 of *LNCS*, pages 1–15. Springer, 2007.
[32] R. Zanibbi and D. Blostein. Recognition and retrieval of mathematical expressions. *International Journal on Document Analysis and Recognition (IJDAR)*, 15(4):331–357, 2012.
[33] R. Zanibbi and B. Yuan. Keyword and image-based retrieval of mathematical expressions. In *Proc. Document Recognition and Retrieval XVIII*, volume 7874 of *Proc. SPIE*, pages 0I:1–8, 2011.
[34] J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Computing Surveys*, 38(2), 2006.