Combining TF-IDF Text Retrieval with an Inverted Index over Symbol Pairs in Math Expressions: The Tangent Math Search Engine at NTCIR 2014

Nidhin Pattaniyil and Richard Zanibbi*
Document and Pattern Recognition Lab
Rochester Institute of Technology, NY, USA
(npatta01@gmail.com, rlaz@cs.rit.edu)

ABSTRACT

We report on the system design and NTCIR-Math-2 task results for the Tangent math-aware search engine. Tangent uses a federated search over two indices: 1) a TF-IDF textual search engine (Lucene), and 2) a query-by-expression engine. Query-by-expression is performed using a bag-of-words approach where expressions are represented by pairs of symbols computed from symbol layout trees (e.g. as expressed in LaTeX or Presentation MathML). Extensions to support matrices and prefix subscripts and superscripts are described. Our system produced the highest highly + partially relevant Precision@5 result for the main text/math query task (92%), and the highest Top-1 specific-item recall for the Wikipedia query-by-expression subtask (68%). The current implementation is slow and produces large indices for large corpora, but we believe this can be ameliorated. Source code for our system is publicly available.

Keywords

Mathematical Information Retrieval (MIR), Federated Search, Inverted Index, TF-IDF

1. INTRODUCTION

Math expressions found in on-line documents are often encoded in LaTeX or MathML. Conventional text search engines ignore the structure provided in those encodings, treating math as normal text. This may prevent users from locating relevant documents due to limited structural information about expressions in the search index. This text-based approach also poses problems for users. In terms of query formulation, there is an *intention gap* for non-experts unfamiliar with LaTeX or MathML (e.g. when trying to learn about '$\binom{n}{2}$' [21]). Also, a recent study confirms that presenting 'raw' math encodings in search results can adversely affect the accuracy of relevance assessment for search hits [16]. Multimodal search interfaces like m_{in} that support visual editing of query expressions may help with query formulation [21] (see Figure 1), but in general better integration of expression layout and/or content is needed.

Issuing a math expression as a query is known as *query-by-expression*, and systems supporting such queries may be beneficial to both non-experts looking up unfamiliar formulae as well as experienced researchers seeking to locate relevant papers based on formulae [22].

*Corresponding author. Telephone: +1 585-475-5023

Note however, that query-by-expression is insufficient in many cases due to publications across different topics and fields sharing formulae. Take for example Bayes' theorem below, which is found in documents from diverse fields like Machine Learning, Statistics, Medicine and History [12]. Without keywords to refine the query, Bayes' theorem will match a wide variety of documents for myriad topics.

$$P(A|B) = \frac{(B|A)P(A)}{P(B)}.$$

When individuals have a mathematical information need, the query they pose often consists of interleaved math and text as opposed to separated math and text, as can be seen in the titles of posts from question-and-answer (Q/A) forums such as MathOverflow and MathStackExchange.[1]

The field of Math Information Retrieval (MIR [23]) is still relatively young, with few benchmark datasets available. The NTCIR-Math tasks provide a valuable resource in this regard, both in collecting system results and disseminating labeled corpora for use in research. 2014 is the second year in which an MIR workshop has been held. The tasks in which we participated are the main text/math query task (Math-2) and the Wikipedia query-by-expression subtask.

For our group to successfully participate in NTCIR-11, we needed to modify our group's earlier query-by-expression system (*Tangent* [19, 20]) so that it could scale to cope

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

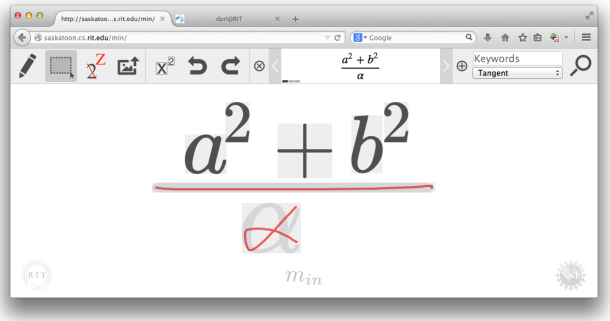


Figure 1: The m_{in} search interface [18]. Input via keyboard, mouse/touch and images is supported. Queries may be sent to different engines, including DLMF [14], Google, Wolfram Alpha, and Tangent

with the gigabytes of documents from the arXiv[2] used for the main task. We also needed to add support for additional structures (e.g. matrices) and wildcard variables. Our intended audience for Tangent was originally non-experts, whom we imagined submitting whole expressions from a pdf document or web page [3, 9] to learn more about them.

Our system performed well in both NTCIR-11 Math-2 tasks, achieving a Precision@5 of over 92% in the main task (when counting both highly and partially-relevant hits), and the strongest result for the Wikipedia expression retrieval task, with a specific-item recall at Top-1 of 68% and 74% at Top-10. However, there are performance issues that need to be addressed. While the original Tangent system was fast enough for real-time use, the current implementation of the expression retrieval engine is slow in terms of indexing and retrieval time, and produces relatively large index files. We remain confident that our basic approach is sound, and that efficient implementations and/or approximations can be created. Source code for our system is available.[3]

We continue with a discussion of related work in text and math retrieval in the next Section.

2. RELATED WORK

In this Section we briefly summarize related work, and in particular work that has influenced the design of Tangent.

Text Search. Text-based informational retrieval is one of the oldest areas of research in Computer Science, and a number of approaches have been devised over the years [11, 17]. Methods based on 'bag-of-word' representations, where the frequency of terms in documents are used to define a vector space remain dominant, with variations of the Term Frequency-Inverse Document Frequency model (TF-IDF) being the most popular (e.g. the Pivot TF-IDF technique [11]). Boolean and Bayesian approaches have also been well-studied and find use in practice.

An important data structure for supporting text search is the *inverted index* [24]. In an inverted index, words or other tokens are mapped to documents that contain them. A widely-used scalable full text inverted index library is the Lucene Java library.[4] We adapt a TF-IDF-based scoring metric provided with Lucene, as discussed in Section 3.2.

Math Search and Query-by-Expression. Mathematical Information Retrieval (MIR) is a comparatively new area of research [23]. An important early development was the creation of the NIST Digital Library of Mathematical Functions (DLMF) which supports LaTeX queries [14]. Currently systems supporting query-by-expression accept queries in LaTeX or Presentation MathML representing the layout of symbols in an expression (see Figure 2b) or Content MathML which represents the underlying mathematical semantics via an operator tree for the expression. Both the layout and operator representations are rooted trees.

Techniques for query-by-expression may be categorized into methods that use operator structure vs. the layout of symbols in an expression, and methods that use sequence-based matching (e.g. by tokenizing LaTeX and then modifying a conventional text retrieval system [14]) vs. those that use the rooted trees representing expression semantics or symbol placement directly [19]. At the present time, it isn't clear whether layout, mathematical content or some combination of both are best to use for math search. However, tree-based methods appear to be proving more effective for query-by-expression than adapting sequential text-based techniques [4, 9, 20].

It is common in math search engines to create an inverted index where symbols, subexpressions and/or complete expressions are mapped to the documents that contain them. Two MIR systems that generate tokens from Presentation MathML are Sojka and Liska's MIaS system [10] and Kristianto et al.'s MCAT system [7]. Tokens are generated differently in both systems, but both store tokens in the Lucene implementation of an inverted index. In the MIAS system, during tokenization an expression is decomposed into individual symbols (e.g. number, operator, function) and subexpressions. The tokenized form of the expression is a generalization, where for example all numbers are represented using one symbol, and all identifiers another. Subexpressions are also generated from these generalized expressions. All generated tokens are assigned a weight capturing a difference/distance from the original formula. When an expression is retrieved, Lucene's TD-IDF scoring algorithm is adapted to use these token weights.

In the case of the MCAT system, an expression is broken into tokens containing symbols, operators, and subexpressions. Sub-expressions generate two token types: the ordered symbols found in the subexpression, and the unordered symbols. Unlike MIaS, when storing tokens in the Lucene index, no additional weight is assigned to tokens.

An alternative to retrieving matched tokens and subexpressions in an inverted index is to instead use a unification-based approach where a hierarchical index of expressions is searched for similar expressions. This is the approach used in Kohlhase and Sucan's Math original WebSearch [6]. MathWebSearch makes use of a substitution index tree, originally developed to unify terms for automated theorem proving. Content MathML is stored in a substitution index tree, and a search for expressions with similar operator structure and operands starts from the dominating/lowest-precedence operators. Nodes in the tree correspond to expressions with common structure at the top of their operator trees. Moving from the root to the leaves of the substitution tree yields increasingly concrete expressions.[5]

The query-by-expression technique we employed adapts Stalnaker's original Tangent system [20]. Tangent represents expressions using an inverted index over pairs of symbols in an expression and their relative positions, effectively using a Boolean bag-of-words (now, a bag-of-symbol-pairs) approach to matching the structure of symbol layout trees (e.g. as represented in LaTeX and Presentation MathML). Candidate expressions are scored by the ratios of query and candidate symbol pair sets that are matched. Somewhat surprisingly, this method tends to handle variable substitutions and locate subexpressions of the query well, due to using both local and non-local symbol relationships.

The original Tangent used a memory-resident hash table implemented in Python and Redis to store expressions in the English Wikipedia corpus. The system was fast enough for use in real-time, and produced strong results in terms of the similarity of returned hits to the query expression [20]. Details of our extensions and integration with text retrieval

[2] http://arxiv.org/

[3] http://www.cs.rit.edu/~dprl/Software.html

[4] http://lucene.apache.org/

[5] Updated MCAT [8] and MathWebSearch systems [5] participated in NTCIR-11 Math-2.

are provided in Section 3.

Indexing Text and Mathematics. If math and text are stored in the same index, such as in MIAS and MCAT, a single function may be used to compute the similarity score between two documents. However, an issue is that math tokenization can generate many tokens, and there are different methods used to tokenize math and text.

Using separate indices for text and math features has been used in systems such as MaTeSearch [2] and Nguyen's system [15]. The issue raised when using separate text and math indices is how to combine their rankings. The common approach is to use a linear combination of a documents' math and text similarity scores. In MaTeSearch, parameters for the weight function were set empirically based on observations about the number of documents typically returned for math and text queries. Nguyen et al. instead used a learning algorithm to set the combination weights.

3. METHODOLOGY

The Tangent system used for NTCIR-Math-2 combines a text retrieval system with a query-by-expression system. It is a generalized version of Stalnaker's original Tangent system [20]. In this section we summarize how scores from the math and text sub-systems are combined, followed by details of the retrieval and scoring mechanisms used by each.

3.1 Combining Math and Text Scores

Determining an optimal manner in which to combine scores from separate subsystems for text and math retrieval is challenging. Nguyen's MIR system [15] used a Large Margin Perceptron trained on pairs of documents from Math Stack Exchange to learn the weights for his linear scorer. A similar approach could be used for Tangent, but we instead varied the combination weight over multiple runs in the main task to observe the effect of the text weight (see Section 4).

We use a simple linear combination of the retrieval scores for the text (Lucene/Solr) and math (symbol pair-based) subsystems. If $\alpha \in [0, 1]$ is the weight for the text portion of the system, the final score for a document d is given by:

$$score(d) = \alpha\ lucene(d) + (1 - \alpha)\ expr(d)$$

where $expr(d)$ is the maximum score for a formula in d, and $lucene(d)$ is the score for d output by Lucene.

3.2 Text Retrieval using TF-IDF/Lucene

Most common approaches to text retrieval use the term frequency-inverse document frequency model (TF-IDF [17]). TF-IDF represents a document as a vector of word frequencies, with each frequency logarithmically scaled by the inverse ratio of documents containing the term (IDF). The IDF scaling gives preference to rarer, and thus often more discriminating terms: $idf(t) = \log(\frac{N}{d \epsilon D : t \epsilon d})$ where N is the number of documents in corpus D, t is a term, and $d \epsilon D : t \epsilon d$ is the number of documents containing t.

The number of vector elements corresponds to the number of pre-selected terms from the corpus included in the model (e.g. extremely frequent, non-discriminative 'stopwords' such as 'the' are excluded). This representation allows documents to be compared within a vector space, most commonly using the dot product of normalized document vectors. The dot product is proportional to the angular difference between two document vectors, arising from relative differences in TF-IDF term values.

A key data structure for text search is the *inverted index*, which maps terms to the documents that contain them. In our system we have used Lucene, which provides an inverted index and TF-IDF scoring algorithm. The text similarity score used in our system is provided by Lucene's Disjunction query, which applies TF-IDF to multiple fields of a document.

For a single field, Lucene scores a document d for query $q = \{t_1, \ldots, t_n\}$ by:

$$s(q, d) = c(q, d)\ idn(q) \sum_{t \in q} \sqrt{freq(t, d) \cdot idf(t)^2} \cdot norm(t, d)$$

where c is the ratio of query terms (t_i) matched in the document, $idn(q) = \sqrt{\sum_{t \in q} idf(t)^2}^{-1}$ normalizes the squared idf values, $freq(t, d)$ the number of times t appears in d, and $norm(t, d)$ is the normalized number of tokens in the field (in our case, body text or title).

To emphasize title text, the score for the title field is doubled, and then the maximum of the title and body text scores is returned as the final text score.

3.3 Math Retrieval using Symbol Pair Sets

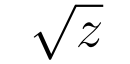
Symbol Normalization. To avoid treating identical symbols with different codes (e.g. unicode vs. LaTeX codes) as separate symbols, it is important to map these symbols to common codes in a normalization step. We use integers for this purpose. Unfortunately, in the current implementation this step is working improperly, meaning for example that '>' and '`>`' are treated as different symbols. Our reported results may be slightly lower and our indices larger due to these issues.

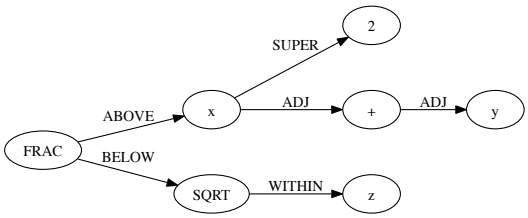
After the competition we noticed that the LaTeXML tool used to convert the arXiv data for the main NTCIR-11 retrieval task inserts 'implicit multiplication' operators between most adjacent identifiers and operators, leading to the representation of relationships with and between invisible operators in our index. Due to the consistency of the mapping, we expect that this has increased our index size without significantly impacting the relevance of returned hits. Earlier results suggest that effective retrieval is obtained by Tangent without implicit operators [20].

Symbol Pair Tuple Generation. The input expression is translated from Presentation MathML to a symbol layout tree [23] that describes the relative placement of symbols in an expression (see Figure 2b). Next, symbol pairs are generated from the layout tree using a depth-first traversal from the root of the tree toward the leaves. All symbols generate a pair with each of their descendants, with symbols at the end of baselines generating a relationship pair with the placeholder symbol *None*.

For NTCIR we extended our earlier algorithm for generating pairs [20] to accommodate matrices and prefix subscripts and superscripts. Matrices are handled by treating whole matrices as symbols represented by their dimensions along with LaTeX for the subexpression located in each cell (see Figure 3). The expression on the main baseline of the expression is indexed in the normal way, treating matrices as individual symbols, and each matrix subexpression cell is converted to symbol pairs.

Prefix subscripts and superscripts have also been added, through negating the distance between symbols if the child

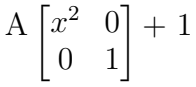


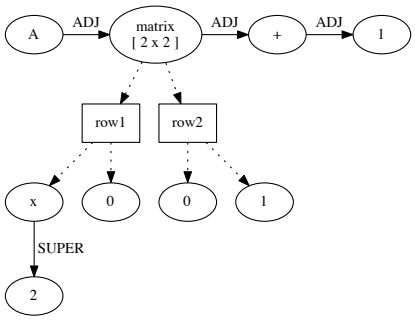


Parent	Child	Dist.	Vert.
FRAC	x	1	1
FRAC	2	2	2
FRAC	+	3	1
FRAC	y	3	1
FRAC	SQRT	1	-1
FRAC	z	2	-1
x	2	1	1
2	**None**	**0**	**0**
x	+	1	0
x	y	2	0
+	y	1	0
y	**None**	**0**	**0**
SQRT	z	1	0
z	**None**	**0**	**0**

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

Figure 2: Symbol layout representations. Tuples are defined for every descendant of a symbol (c) in the symbol layout tree (b). Symbols without children produce a tuple with child symbol 'None.' In (c), *Dist.* is the path length from the parent symbol to the child symbol in the symbol layout tree, and *Vert.* is a sum representing vertical displacements along this path: +1 for each superscript/above edge, -1 for each subscript/below edge, and 0 for each horizontally adjacent/within edge.





Matrix Structure			
Parent	Child	Row	Column
matrix	dimensions	2	2
matrix	'x^2'	1	1
matrix	'0'	1	2
matrix	'0'	2	1
matrix	'1'	2	2
Subexpressions			
Parent	Child	Dist.	Vert.
A	**matrix2x2**	1	0
A	+	2	0
A	1	3	0
matrix2x2	+	1	0
matrix2x2	1	2	0
+	1	1	0
1	None	0	0
x	2	1	1
2	None	0	0
0	None	0	0
0	None	0	0
1	None	0	0

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

Figure 3: Matrix handling. At the topmost level of the expression, matrices are treated as single symbol labeled by their dimensions (e.g. 'matrix2x2'). This topmost expression along with all subexpressions in matrix cells are represented as in Figure 2 (under Subexpressions in (c)). Additional tuples are used to represent matrix dimensions, and the contents of matrix cells.

symbol is at left of the parent symbol, rather than at right.[6] For example, the symbol pair from C to n in ${}_nC_2$ ('n choose 2') would be represented by $(C, n, -1, -1)$.

Inverted Index and Retrieval. Figure 4 provides a summary of the expression retrieval algorithm used by Tangent. Our inverted index is broken into two parts, one lookup table mapping symbol pairs to expressions (*pairIndex* in Fig. 4), and a second lookup table mapping expressions to the document containing an expression, and a list of (pair, count) entries to represent symbol pairs in the expression (*exprIndex* in Fig. 4). To save space, symbol names are replaced by integer codes, and another table is used to map expressions to their LaTeX representation.

[6]First suggested by P. Pavan Kumar et al., University of Hyderabad, India.

Using the inverted index in this way, our basic approach is for *specific* symbol pairs and structures. However, in both the main and Wikipedia subtasks, expressions in queries could contain (enumerated) wildcards replaceable by any symbol, e.g. $a^{?i} + b^{?i} = ?j^2$, where $?i$ and $?j$ represent two different wildcard symbols, with a and b having the same exponent. To address this, we have added entries in the symbol pair index where *either* the parent or child symbol is a wildcard. We then add a greedy search in our ranking algorithm that assigns concrete symbols to wildcards such that the largest number of unmatched symbol pairs associated with *one* wildcard symbol is matched incrementally (see Figure 4, Step 5). To avoid overwhelming the system, we ignore pairs of wildcard symbols (e.g. $(?i, ?j, 0, 0)$) and wildcard symbols at the end of a baseline (e.g. $(?w, None, 0, 0)$). This means that those pairs are treated as unmatched in the

```
retrieveExpression( query, topK, pairIndex, exprIndex ):

Let H be the search hits, an empty list.
Let C, L, and R be hash tables from expr. ids to symbol pairs
 with counts, e.g. C: eid -> ((pair1,count1), ... ,(pairN, countN))
Let l* and r* be hash tables of candidates for wildcards
  in an expression. l*,r*: (eid, ?w) -> (list of symbols)

1. Normalize the query, and build a symbol layout tree (T)

2. Generate symbol pairs (Q) for query tree T using a depth-first
   traversal

3. Symbol pair lookup:
   a. Find symbol pairs from Q without wildcards in pairIndex,
      update C to record matching pairs in pairIndex
   b. Find each pair (?p, [any symbol]) from Q in pairIndex,
      update L; add symbols matching ?p to l* for each expression
   c. Find each pair ([any symbol], ?c) from Q in pairIndex,
      update R; add symbols matching ?c to r* for each expression

4. Filter: Sort expressions in C by matched pair count, then
   keep only the topK expressions in C

5. Match wildcard symbols and score expressions:

   For each expression (eid) in C:
    a. Let U be a set containing unmatched symbol pairs with counts.
       Obtained by removing matched pairs in C[eid] from all pairs
       in the expression, stored in exprIndex[eid]
    b. Let W be the set of unique wildcard symbols in Q,
       and wildcard match count M be 0
    c. Until W is empty, or no wildcard match is found:
        i. Using substitution candidate tables l* and r*, find the
           (symbol, ?w in W) substitution that is most frequent
       ii. Remove matching pairs from U, and matched wildcard ?w
           from W. Increment M by the number of matched pairs
    d. Let I be the sum of M and the number of matched pairs
       in C. Let Recall be I/|Q| and Precision I/|eid|, where
       |eid| is the number of symbol pairs in the expression
    d. Add (eid,score) to hit list H, where score is the F-measure
       for query and expression symbol pair matches, defined
       by F = (2 * Recall * Precision) / (Recall + Precision)

6. Sort search hits H by F-measure, then return H
```

Figure 4: Tangent expression retrieval algorithm. *?p, ?c* and *?w* represent wildcard symbols in queries

query, but also means that when there are a large number of wildcards, many parts of the structure of the expression may not be represented.

Matching expressions are ranked by the harmonic mean of the ratio of matched query expression pairs and ratio of matched candidate expression pairs. We return both the specific expression along with the document with which the expression is associated.

Multiple Expressions. The math score for a document d is given by a weighted sum of the top-1 match scores for each query expression e_i ($t_1(d, e_i)$). Individual top-1 expression match scores are weighted by the ratio of pairs in a query ($|e_i|$) to all pairs in queries:

$$m(d, e_1, ..., e_n) = \frac{|e_1|}{\sum_{i=1...n} |e_i|} t_1(d, e_1) + \ldots + \frac{|e_n|}{\sum_{i=1...n} |e_i|} t_1(d, e_n)$$

4. RESULTS

Our group participated in two tasks for NTCIR-Math-2: 1) the 'main' Math-2 task, and 2) the Wikipedia subtask for formula retrieval using query-by-expression. These corpora are quite different. The English Wikipedia corpus contains 387,947 unique expressions,[7] while the arXiv corpus used for the main Math-2 task contains 30,008,971 unique expressions. The Wikipedia corpus contains roughly 35,000 encyclopedia entries, each treated as a single document. The arXiv corpus contains 100,000 scientific articles (e.g. from physics) split into fragments, producing 8,301,578 'documents.' These fragment 'documents' range in size from a couple of words to complete derivations with accompanying text. The uncompressed arXiv collection occupies 173GB on disk.

Formula Query: $\mathbb{P}[\boxed{X} \geq \boxed{t}] \leq \frac{\mathbf{E}[\boxed{X}]}{\boxed{t}}$

Keyword: Markov inequality

a) Math-2 #39

$$\mu(A) = \begin{cases} 1 & \text{if } 0 \in A \\ 0 & \text{if } 0 \notin A. \end{cases}$$

b) Wikipedia #49

Figure 5: Sample Queries. Query a) contains four wildcard symbols (shown in boxes), and two keywords. Queries for the Wikipedia subtask were single expressions. Query b) has no wildcards and includes a tabular/matrix layout

Queries differ in the two tasks; Figure 5 provides examples. In the main task, most queries are a single expression along with keywords. One main task query contained four query expressions, and two contained two query expressions. The number of keywords in each query was between one and six, with roughly three keywords on average ($\mu = 3.1, \sigma = 1.27, mode = 3$). For the Wikipedia task, 100 expressions were selected randomly from the English Wikipedia articles, and then had variables replaced by wildcards uniformly at random. The original names of wildcard variables were provided in the main task, but enumerated as $x_1, \ldots, x_n$ in the Wikipedia task. In the Wikipedia task, 36 of the 100 queries contained wildcards (36%), while for the Math-2 task 40 of 50 queries had an expression containing wildcards (80%), with an average of 2.9 wildcards per expression ($\mu = 2.9, \sigma = 2.3, mode = 2$), and between 0 and 10 wildcards in query expressions.

The main task (Math-2) was evaluated using hits judged by two human evaluators. Evaluator ratings were combined into a single Likert scale rating between 0 (irrelevant to the query) and 4 (highly relevant). Unjudged hits were treated as irrelevant. Evaluations were performed two ways: 1) **highly-relevant condition**: judged hits with a rating of 3 or 4 are treated as relevant, and 2) **partially-relevant condition**: judged hits with a rating greater than 0 are treated as relevant. The *treceval* evaluation tool (version 9.0) was used to compute Precision@5 and Precision@10 metrics.

Due to the large number of submitted runs for the main task, a 'round-robin' sampling method was used to select hits for ranking, trying to cover as many of the submitted top-k hits as possible. Most top-5 hits were evaluated for systems, but coverage of the top-10 was less consistent (for example, on average 6 of our system's top 10 hits were judged). As a result, we focus our analysis on top-5 results.

For the Wikipedia subtask, a form of specific-item-recall was used to evaluate systems. The topmost position where the original article from which a query was taken appears were compared. Due to this, it is possible to match a different expression in the article, and have the returned ex-

[7] computed based on the number of unique LaTeX strings

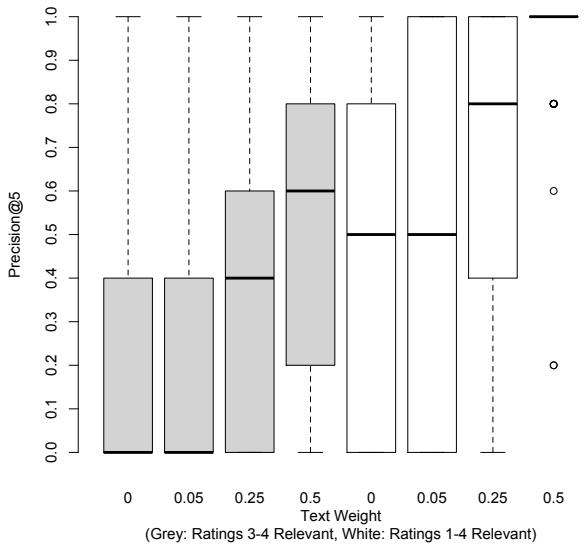


Figure 6: Tangent Precision@5 (Main Task) for 50 queries combining one or more math expressions with keywords. The collection holds roughly 100,000 arXiv articles broken into fragments serving as the retrieval units ('documents'). Grey: highly relevant hits; White: partially + highly relevant hits

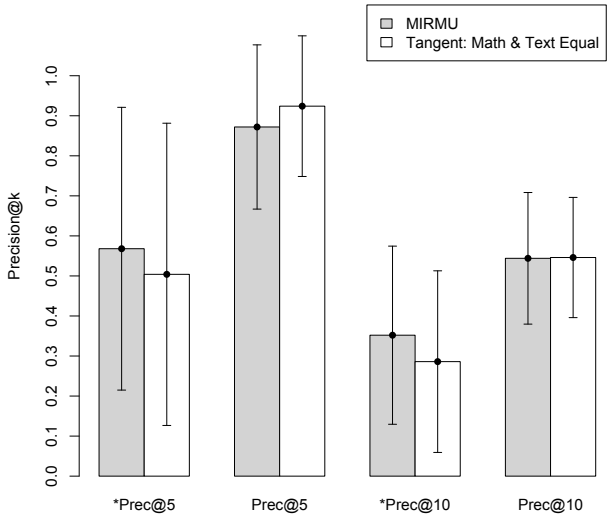


Figure 7: MIRMU System vs. Tangent (Main Task). *Prec@ indicates precision for high-relevance hits (rated 3-4), and Prec@ using hits rated higher than 0. Note that unevaluated hits are treated as misses. For Precision@5 at most 1 hit is unevaluated for each of the 50 queries (10 for MIRMU, 7 for Tangent). For Precision@10 the average number of evaluated top-10 hits in both systems is only 6, (i.e. a maximum possible Precision@10 of 60% on average)

pression and article treated as a valid hit. It may also be possible to match the query expression identically, but find it in an article differing from the query and treated as a miss. However, this seems to be a rare occurrence in the Wikipedia corpus.

4.1 Tangent Submissions and Retrieval Results

Our group submitted 4 runs to the main task, each of which used a different weight for text retrieval in the final ranking of hits: $\{0, 0.05, 0.25, 0.5\}$. In the following, weight 0 for text is termed 'math-only,' and 0.5 'equally-weighted.' For the Wikipedia subtask, only the query-by-expression subsystem was used in a single run.

Math-2 (main) Task. For the main task, looking at the boxplots in Figure 6 we see that increasing the weight of text results substantially increases the Precision@5 ratings for both the highly-relevant and partially + highly relevant conditions. The average Precision@5 for the partially-relevant condition with text and math weighted equally is over 92% ($\mu = 0.92, \sigma = 0.18$). Observing hits returned by the math-only and math/text equally weighted conditions, some trends emerge. Returning documents based on the best expression match appears to work best when: 1) expressions are not tiny; expressions such as '(D)' produce exact matches, but with little relevance to the query topic as they are not distinctive, 2) query expressions have neither too few nor too many wildcards. Queries with many wildcards obtain poor results due to wildcard symbol pairs being ignored by our retrieval algorithm, while one or two wildcards seems to provide useful flexibility, and 3) rare symbols and/or relationships tend to produce relevant matches in the math-only condition (e.g. for expressions using $\bigoplus$ as

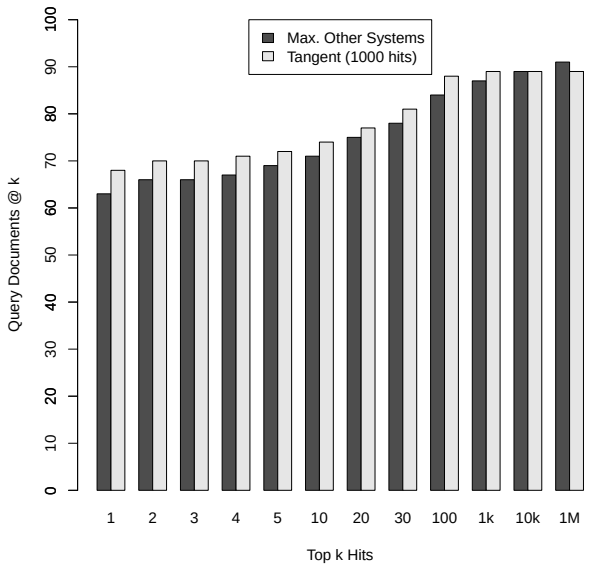


Figure 8: Wikipedia Math Search Subtask Results. 100 English Wikipedia articles were chosen at random. Query expressions were produced by selecting one math expression at random from each article, replacing variables in these expressions by wildcards (qvar) at random. 'Query Documents @k' is a specific-item recall measure, giving the percentage of query articles returned in the first k hits

an indexed operator), precisely because they are infrequent.

For the main task, the 'math only' Tangent system produced no hits in the top-5 results for 26 queries for the high-relevance condition, but only 15 complete misses for the partially-relevant condition. In contrast, the equal-weight Tangent had 7 queries with no hit in the top-5 in the high-relevance condition, and no complete misses in the partially-relevant condition.

The high partially-relevant score for the equally weighted math and text condition is in part because the Lucene hits were often much smaller document excerpts than those for the math index hits. In some cases the Lucene hits were single words, whereas math queries often returned large excerpts with complete derivations and surrounding text. For example, in one query the term 'Lisboa' is used, which then matches a number of addresses contained in the database (which are rated as partially relevant by judges), but doesn't appear alongside a mathematical concept. We believe this occurs because Lucene prefers tight matches, whereas Tangent's math retrieval engine ranks documents by highest similarity of a single expression in the excerpt, ignoring other content in the document.

In cases where both the math-only and math/text equal conditions produce primarily highly-relevant hits in the top-5, the hit diversity was high (for queries 18, 22, 36 and 47, containing 2, 2, 2 and 0 wildcards, respectively). For the highly-relevant condition, there were in fact six queries where the math-only condition had higher Precision@5 than math/text equal (queries 2, 10, 11, 38, 48, and 50). Three of these queries contain 6 keywords. It is possible that somehow this led to weaker text retrieval results, due to the number and variety of query terms. In some cases the math-only condition performed better because the text retrieval system was returning small excerpts (a single sentence or less) with little context, and so were not rated as being highly relevant.

A comparison of results from the math/text equally weighted condition and the system (MIRMU [13]) with the highest 'highly relevant' ratings obtained for the main task are shown in Figure 7. Tangent with equally-weighted math and text produced the highest partially relevant Precision@5 result, and the second-highest highly-relevant Precision@5 result for the main task.

Wikipedia Subtask. Results for Tangent vs. the highest result from other runs at different ranks are provided in Figure 8. At all ranks up to 1000 Tangent has the highest metric value. Tangent returned 68 of the 100 query articles as the first hit, with the next-highest top-1 result 5% lower (63 hits). With 36 queries containing wildcards, it seems likely that most participating systems were able to match the original query expression in the top-1 when no wildcards were used ($63 + 36 = 99$). Additional hits near the top of the rankings most likely occur because some variant(s) of the query expression are ranked above the query expression itself (e.g. when a wildcard variable is bound to y, rather than x in the original query). Tangent behaves as one expects, preferring exact matches and structurally similar matches with few additional or missing symbols [19, 20].

4.2 System Configuration and Performance

We constructed our search engine using Amazon Elastic Compute Cloud (Amazon EC2), a web service. We used an EC2 memory-optimized configuration (r3.4xlarge) with the following specifications: 16 vCPUs, 2.5 GHz, Intel Xeon E5-2670v2, 122 GB memory, 1 x 320 GB Disk. A snapshot was configured to include MySQL, Tangent, python packages, Java, and Solr (which includes Lucene). The document collection itself was large, consuming 173GB of disk space.

Table 1: Database table sizes for query-by-expression systems. Sizes shown for the arXiv main task are for 1/9th of the complete collection. For the main task 81,774,641 symbol pair entries are defined across all nine indices (including repetitions).

Table	Rows	Size(MB)	Index(MB)
arXiv (main)	**Shown: 1 of 9 Indices**		
symbol pairs	14,791,465	2600	692
expression-docs	5,927,284	183	147
expression	5,636,077	313	78
symbol-ids	195,960	6	10
Wikipedia	**Shown: Complete Index**		
symbol pairs	3,002,881	305	141
expression-docs	387,975	12	9
expression	387,947	775	6
symbol	56,437	2	3

For the main task, expressions were indexed in a divide-and-conquer fashion, with 9 EC2 machines with the above configuration indexing 1/9th of the collection apiece, and one additional instance used for Solr/Lucene. Because all 9 folds are built independently, the same expression may be present in different folds, with a different expression identifier. Similarly there is one symbol index per server. However, no document occurs in more than one database. The sizes of the tables for symbol pairs, expression documents, expressions and symbols are shown in Table 1.

An additional instance was used to issue queries and then compile results from the nine math indices and the Lucene index. This instance was configured to allocate 65% of its available RAM for MySQL. Results from the nine servers were then pooled to produce the final 1000 hits output by the expression index, which were then combined with the Lucene hits using the weight parameter α.

As can be seen in Table 2, both indexing and retrieval were quite slow for the main task. This is partly due to the number of unique symbol pairs, and for retrieval the large number of pairs that may match symbol pairs containing a wildcard. We suspect that there may be some issues with how the database was constructed and the database tables were organized, which may also be slowing things down.

In contrast, the Wikipedia corpus was small enough to be indexed and stored using a single machine (see Table 1). Execution time for all 100 Wikipedia queries is eight minutes (Table 2). This is a bit slower than the original Tangent would take at most 3-5 seconds for a large query to finish [20], but wildcards were not supported in that version of the system. For faster execution, one might delay retrieving symbol pairs with wildcards until after concrete symbol pairs have been located, or reduce the index size by reducing the granularity of symbol distances (e.g. treating symbols at distance three or higher as one value [20]).

5. CONCLUSION

We have extended the Tangent query-by-expression search engine to support new structures and wildcard matching, scale up to larger collections, and integrate with TF-IDF-based text search. The resulting system performed well

Table 2: Indexing & retrieval times for query-by-expression. Search times shown are for 50 main task queries, and 100 Wikipedia subtask queries.

Collection	Time (minutes) Index	Search
NTCIR-main (arXiv)	$420 \times 9 \approx 3380$	150
Wikipedia	33	8

in both the Math-2 main task and Wikipedia query-by-expression subtask. However, a number of performance issues need to be resolved, in terms of both space and time.

For future work, perhaps the most pressing need is to improve the handling of wildcards. Currently results are poor for queries with many wildcards because we ignore relationships between pairs of wildcard symbols, as well as wildcards at the end of a baseline. The current ranking of documents in the expression index by the best matching expression is brittle. Type information may also be helpful, for example to prevent wildcard variables from matching constants. We might also rank matches where keywords and expressions are close to one another higher (i.e. by locality of the match) to improve retrieval for mixed math and text queries.

Finally, we believe that our expression retrieval method can be adapted to work with operator trees (e.g. Content MathML), and we are interested in combining layout and operator tree information in a query-by-expression system.

Acknowledgements

We wish to thank David Stalnaker and Frank Tompa for helpful discussions, as well as Akiko Aizawa and Moritz Schubotz for their assistance with obtaining results. This material is based upon work supported by National Science Foundation (USA) under Grant No. IIS-10161815.

6. REFERENCES

[1] Anca, S., and Kohlhase, M. MaTeSearch: A combined math and text search engine. Tech. rep., Jacobs University, Bremen, Germany, 2007.

[2] Baker, J., Sexton, A. P., and Sorge, V. Extracting precise data on the mathematical content of PDF documents. In *Proc. Digital Mathematics Libraries* (Birmingham, UK, July 2008), pp. 75–79.

[3] Kamali, S., and Tompa, F. W. Retrieving documents with mathematical content. In *Proc. ACM SIGIR* (New York, USA, 2013), pp. 353–362.

[4] Kohlhase, M., Hambasan, R., and Prodescu, C.-C. MathWebSearch at NTCIR-11. In *Proc. NTCIR Workshop 11 Meeting* (2014).

[5] Kohlhase, M., and Sucan, I. A search engine for mathematical formulae. vol. 4120 of *LNAI*. Springer, 2006, pp. 241–253.

[6] Kristianto, G. Y., Nghiem, M.-Q., and Aizawa, A. The MCAT math retrieval system for NTCIR-10 math track. In *Proc. 10th NTCIR Conference* (Tokyo, Japan), pp. 680–685.

[7] Kristianto, G. Y., Topic, G., Ho, F., and Aizawa, A. The MCAT math retrieval system for NTCIR-11 math track. In *Proc. NTCIR Workshop 11 Meeting* (2014).

[8] Lin, X., Gao, L., Hu, X., Tang, Z., Xiao, Y., and Liu, X. A mathematics retrieval system for formulae in layout presentations. In *Proc. ACM SIGIR* (Gold Coast, Australia, 2014), pp. 697–706.

[9] Líška, M. Evaluation of Mathematics Retrieval (MSc Thesis). Diplomovprce, Masarykova Univerzita, Fakulta Informatiky, 2013.

[10] Manning, C. D., Raghavan, P., and Schütze, H. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[11] McGrayne, S. B. *The Theory That Would Not Die: How Bayes' Rule Cracked the Enigma Code, Hunted Down Russian Submarines, and Emerged Triumphant from Two Centuries of Controversy*. Yale University Press, 2012.

[12] Michal Růžička, P. S., and Líška, M. Math indexer and searcher under the hood: History and development of a winning strategy. In *Proc. NTCIR Workshop 11 Meeting* (2014).

[13] Miller, B., and Youssef, A. Technical aspects of the digital library of mathematical functions. *Annals of Mathematics and Artificial Intelligence* (2003), 1–19.

[14] Nguyen, T. T., Hui, S. C., and Chang, K. A lattice-based approach for mathematical search using Formal Concept Analysis. *Expert Systems with Applications 39*, 5 (Apr. 2012), 5820–5828.

[15] Reichenbach, M., and Zanibbi, R. Rendering expressions to improve accuracy of relevance assessment for math search. In *Proc. ACM SIGIR* (July 2014), pp. 851–854.

[16] Salton, G., and McGill, M. J. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.

[17] Sasarak, C., Hart, K., Pospesel, R., Stalnaker, D., Hu, L., Livolsi, R., Zhu, S., and Zanibbi, R. m_{in}: A multimodal web interface for math search. In *Symp. Human-Computer Interaction and Information Retrieval (HCIR)* (Cambridge, MA, Oct. 2012).

[18] Stalnaker, D. Math Expression Retrieval Using Symbol Pairs in Layout Trees. Master's thesis, Rochester Institute of Technology, NY, USA, August 2013.

[19] Stalnaker, D., and Zanibbi, R. Math expression retrieval using an inverted index over symbol pairs. In *Proc. Document Recognition and Retrieval XXII* (San Francisco, USA, 2015). (to appear).

[20] Wangari, K., and Zanibbi, R. Discovering real-world usage scenarios for a multimodal math search interface. In *Proc. ACM SIGIR* (July 2014), pp. 947–950.

[21] Youssef, A. Roles of math search in mathematics. In *Proc. Mathematical Knowledge Management*, no. 4108 in LNAI. Springer, Wokingham, UK, 2006, pp. 2–16.

[22] Zanibbi, R., and Blostein, D. Recognition and retrieval of mathematical expressions. *Int'l. J. Document Analysis and Recognition 15*, 4 (Dec. 2012), 331–357.

[23] Zobel, J., and Moffat, A. Inverted files for text search engines. *ACM Computing Surveys 38*, 2 (July 2006), 56 pp.