The Abject Failure of Keyword IR for Mathematics Search: Berkeley at NTCIR-10 Math

Ray R Larson
University of California, Berkeley
School of Information
Berkeley, CA 94720-4600
+1-510-642-6046
ray@ischool.berkeley.edu

Chloe J Reynolds
University of California, Los Angeles
IT Services | 3327 Murphy Hall
Los Angeles, CA 90095-1434
+1-310-206-1621
creynolds@it.ucla.edu

Fredric C Gey
University of California, Berkeley
Inst. for Study of Societal Issues
Berkeley, CA 94720-5670
+1-510-292-8421
gey@berkeley.edu

ABSTRACT
This paper demonstrates that classical content search using individual keywords is inadequate for mathematical formulae search. For the NTCIR10 Math Pilot Task, the authors used a standard indexing by content word for search coupled with search for components of mathematical formulae. This was followed by formula extraction from the top ranked documents. Performance was terrible, even for partial relevance. The further inclusion of some manual reformulation of topics into queries did not improve retrieval performance.

Team Name
BRKLY

Categories and Subject Descriptors
H.3.3 [**Information Systems**]: Information Search and Retrieval—retrieval models, search process**.**

General Terms
Experimentation, Performance, Measurement

Keywords
Mathematics Search, MathML, Scientific Search.

1. INTRODUCTION
Search through scientific documents according to the mathematical formulation of equations is a emerging and challenging research area in the science of search. The availability through the web of large collections of scientific documents either in LaTeX or MathML makes this research valuable to the advancement of scientific knowledge. However there are many levels of abstraction which need to be introduced to move from traditional keyword search of segmented XML documents to matching of mathematical formulae in which the basic semantic units may be expressed in an almost infinitude of lexical formats. For example the quadratic formula

$ax^2 + bx + c$

should also match

$3x^2 + 1$

or

$qx^2 + rx + s$

as well as

$x_i\alpha^2 + x_j\alpha + \beta$

Because hundreds of thousands of documents are in LaTeX source, software must be written to convert them into the MathML format because the original authors cannot be expected to manually reconvert their document into the new format. The application of translation software has its limitations as NTCIR10 Math task topic NTCIR10-FS-7 will show below.

2. BAG OF WORDS SEARCH
The search methodology most commonly used in information retrieval is to take the information need, expressed as a verbal phrase or sentence and extract the 'meaningful' set of words and search for these words in a similar representation for each document. For scientific documents stored in the LaTeX source format, it may be theoretically possible to take formulae components and use them in search. However since the alterative format, MathML (Mathematics Markup Language) allows for segmentation of scientific documents into an xml structure more concerned with the layout of the elements on the page than with the semantic content, the application of transformation software may yield unpleasant results. For example, NTCIR10-FS-7, expressed in LaTeX as:

<TeXquery>sin(\qvar{x})/\qvar{x}</TeXquery>

the translation takes the mathematical operation for the sine function (expressed as sin) and splits each letter off into an individual xml component, changing it to:

xml:id="m10.1.1" ref="m10.1.1.pmml">s</m:ci>
xml:id="m10.1.2" xref="m10.1.2.pmml">i</m:ci>
xml:id="m10.1.3" ref="m10.1.3.pmml">n</m:ci>

A keyword approach using this split retrieves documents with each of the individual letters instead of the actual trigonometric function which is a distinct semantic unit. Since this has occurred in one of the limited number of search topics for NTCIR10 Math, it is unclear how many documents in the test collection may have been semantically corrupted by the translation software.

In normal XML retrieval, the searchable content is almost always the text nodes of the XML tree structure, and the

XML structure itself is used only as way of segmenting elements of content into more or less focused components. This notion is turned on its head when considering MATHML row layout (<pquery>) or the other more semantic variant (<cquery>). In the former, for example an integral often appears as just a literal integral symbol in a given row, while in the latter it is a separate XML tag with with no content "<m:int/>" (with id and xref attributes removed for clarity). In normal XML retrieval such "EMPTY" elements lacking significant text content or attributes are totally ignored, but in this task they take on the utmost significance.

One approach for turning this XML markup into searchable text is to treat each of these "EMPTY" elements as an index entry that simply records that a given document contains such elements. We implemented this as an extension of Cheshire II bitmap indexes, that record a single bit to represent whether or not a document has a given XML tag [6]. Indexes of this type were created for "EMPTY" tag symbols including m:sum, m:int, m:infinity, m:times, m:interval, m:plus, m:minus, m:divide, m:root, m:eq, m:neq, m:leq, m:geq, m:abs, m:in, m:list, m:cos, m:mfrac, m:log, etc.

In addition to such symbols, conventional text indexes were used to capture all of variable elements expressed as the text nodes for a given row, a given m:subsup, a given math operator m:mo, numeric value m:mn, or variable m:mi, etc.

For the automatic querying the same kind of element identification and extraction used for the indexing process, yielding extremely complex queries. For example:

"*search topic @@ {{ Multiple integral computation, axis commutativity formula }} and math @@ {{ ∫ 0 ∞ d □x □∫ x ∞ F □ x , y □d □y = ∫ 0 ∞ d □y □∫ 0 y F □ x , y □d □x superscript subscript 0 normal-d x superscript subscript x F x y normal-d y superscript subscript 0 normal-d y superscript subscript 0 y F x y normal-d x }} and times 1 and eq 1 and integral 1 and infinity 1 and interval 1*"

These automatic searches usually provided only a few results for each topic, since all of the "EMPTY" elements resulted in Boolean restrictions on the result sets. But even with such restriction there was no expression of order or structure in the queries – matching results had the same "bag of symbols" as the query and we used a post-search process to extract formulae and compare more closely to the query specification. Naturally, if a document had a variant form of a formula, it was most likely rejected in the Boolean aspects of the search and never considered in further extraction.

BRKLY submitted three automatic runs, R1, R2, and R3 to NTCIR-10 Math [1]. These were differentiated by using the full topic including the topic name, the <cquery> and <pquery> contents as "R1", only the <pquery> contents as "R2" and only the <cquery> contents as "R3".

3. FORMULA EXTRACTION

3.1 Formula Extraction Technique Overview

The first stage (bag of words) of our information retrieval process identified potential match documents for a given query. Let us call the results from this stage the *preliminary result set*. The second stage identified a formula's identifier within a document, for each of the first 100 documents retrieved in the preliminary result set, for each query.

If no exact match to the original query was found in a given document by the second process, then the final search result would be a document identifier without a formula identifier for that query and document pair. Otherwise the final result would be both a document and formula identifier.

The formula extraction technique addresses only queries of the formula search and cannot find identifiers of non-formula search queries. Formulas in the query document are extracted through this regular expression:

pattern = re.compile('<cquery>\s+<m:math>(.?)</m:math',re.S)*

while the corresponding formulas in the scientific document collection. Documents are extracted through this regular expression:

pattern = re.compile('<m:annotation-xml id="id\d+" encoding="MathML-Content">(.?)</m:annotation-xml>',re.S)*

Details about the formula extraction algorithm can be found in the Appendix.

3.2 Formula Extraction Performance

Formula extraction was used to hone the bag of words preliminary results set. Since the bag of words outcome yielded few appropriate document results, the formula search technique also yielded few correct results. Limited small-scale manual tests indicated 100% accurate results of formula ID extraction for exact matches of query for-mulas. However, the performance of the feature extrac-tion technique - assuming a large theoretical test set of 100% correct preliminary results - is unknown.

Yet it would be infeasible to skip the bag of words stage and instead run the formula extraction technique directly against the 100,000 documents for each query. One reason is that the formula extraction technique has many loops and is run on a single laptop. Those features make the technique a slow if impossible task to run against a base of 100,000 documents.

A second reason is that the formula extraction process only addresses formula queries, whereas the bag of words technique can retrieve document results for queries con-taining words and sentences (i.e. that are not formatted as formulae), yielding at least a document ID for our team's results for those non-formulae queries.

4. MANUAL QUERY REFORMULATION

Manual reformulation of topics into queries with additional semantic content has been an often used methodology for enriching the judgment pool in developing test collections. The approach was most successfully used by University of Central Florida [3] in the first TREC Spanish retrieval track . In that evaluation the experimenters spent over 40 hours per topic developing an entire semantic structure around the original query words, for example the term 'Mexican jewelry' was expanded to include 'bracelets, rings, necklaces" and "gold, silver, copper, turquoise." The resulting retrieval produced a convex recall-precision curve instead of the usual concave curve usually found.

The BERKLY team spent some small time an effort developing a manual reformulation of a few of the NTCIR-10 Math topics. This formulation was then applied as a Boolean search against the document collection. This approach, taken on Formula Search for eight Formula Search Topics NTCIR10-FS-1, NTCIR10-FS-2, NTCIR10-FS-4, NTCIR10-FS-5, NTCIR10-FS-6, NTCIR10-FS-7, NTCIR10-FS-8 and NTCIR10-FS-13

In most cases, particular keywords were taken from the topic description and searched for in the title section of the document. For example, for NTCIR10- FS-7, the word 'sin' was used, retrieving 14 such documents, including

docid 11464|rank 5| rawrel 1.0|filename /projects/metadata2/ntcir10-math/data/NTCIR-sandbox/39/f015329.xhtml

<title>TUM-HEP-285/00MPI-TH/2000-30 "A Lower Bound on sinâ¡2â¢Î²2Î² \sin 2\beta from Minimal Flavour Violation"

</title>

Formulas were then extracted from the list of documents. Manual reformulation was also done for Full Text Search Topics but because of a processing oversight the formula extraction was not run.

5. RESULTS

Results below are reported only for the FS formula search runs because the single manual FT full text search submission was not included in the judgment set because formula extraction was not performed for that run.

Table 1. Relevance Level >= 3 (Relevant)

	BRKLY. R1	BRKLY. R2	BRKLY. R3	BRKLY. R4
MAP	0.024	0.000	0.000	0.024
P@10	0.010	0.000	0.000	0.019
P@5	0.005	0.000	0.000	0.010
Precision (count)	0.004 (3/815)	0.000 (0/898)	0.001 (1/911)	0.062 (2/32)

Table 2. Relevance Level >= 1 (Partially Relevant)

	BRKLY. R1	**BRKLY. R2**	**BRKLY. R3**	**BRKLY. R4**
MAP avg	0.029	0.001	0.002	0.024
P-5 avg	0.076	0.010	0.010	0.019
P-10 avg	0.048	0.005	0.010	0.010
Precision (count)	0.016 (13/815)	0.003 (3/898)	0.004 (4/911)	0.062 (2/32)

Another way to evaluate the retrievals is to examine performance for individual topics as in the table below for mean average precision (map).

Table 3Individual Topics (Relevance Level = 1)

Topic	**BRKLY. R1.map**	**BRKLY. R2.map**	**BRKLY .R3.map**	**BRKLY .R4.map**
FS-1	0.0039	0	0	0
FS-2	0	0	0	0
FS-3	0.0093	0.0015	0.0278	NA
FS-4	0	0	0	0
FS-5	NF	0	NF	0
FS-6	0	0	0.0050	0
FS-7	0	0	0	0
FS-8	0	0	0	0.0098
FS-9	0	0	0	NA
FS-10	0	0	0	NA
FS-11	0	0	0	NA
FS-12	NF	0	NF	NA
FS-13	0.5	0	0	0.5
FS-14	0.0087	0.0071	0.0020	NA
FS-15	0	0	0	NA
FS-16	0.0245	0	0.0040	NA
FS-17	NF	0	NF	NA
FS-18	0.0408	0.0009	0	NA
FS-19	0	0	0	NA
FS-20	NF	NF	NF	NA
FS-21	0.0125	0	0	NA
FS-22	NF	0	NF	NA
all	0.0286	0.0056	0.0018	0.0243

The table presents results for MAP for individual topics. NF means that no formulas found in submitted docu-ments, NA means (for the manual run BRKLY.R4) Not Applicable, in the

sense that nothing was run for that topic. Results show that only a few queries submitted by the BRKLY team found any relevant formulae.

6. FUTURE WORK

6.1 Formula search incorporated into document indexing

For the limited situations in which a finite body of search documents will be known ahead of time (such as for this workshop), one option would be to parse the for-mula parts for all documents ahead of time. Then index them based on number of formula parts and perhaps also index them on number of operators or boolean existence of operators, or even a list of operators (stored as a sys-tematically ordered list). This might allow a more rela-tively quicker and therefore feasible execution of the for-mula extraction technique directly on the full set of docu-ments rather than requiring a stage 1 preliminary results set on which to operate. Of course, this would find results for queries of the formula search type.

Another structural change could be to reduce the amount of looping in the overall technique.

6.2 Automated Query Expansion

A third type of improvement would be to run the original query through a pre-search process that acts as a sort of thesaurus for math formulae. Special formulas may have specific terms associated with them that would strengthen stage 1 searching. One example is e = mc2. E = mc2 is associated with terms like "mass-energy equivalence" and "speed of light." Augmenting the original search query with these additional query expansion terms before running the query through our stage 1 process would likely improve stage 1/preliminary results and therefore the final results. A separate run could be performed using this pre-search process and compared to the original search method to verify the amount of utility of a query expansion pre-search. An automated source for performing query expansion is Wolfram Alpha's website, which allows mathematical searching and returns results that sometimes includes query expansion terms..

6.3 Improvement in Precision

Each formula contains many identifiers, allowing a searcher to specify with great precision the exact location in a document suspected to match a query. Our feature extraction technique, however, always returns the outer-most identifier as a formula's formula identifier. That is, our technique returns the digits that follow "sid=id" in a given document. As stated above, the regular expression for formula ID extraction is:

```
pattern = re.compile('<m:\w*\sid="id(\d+)"',re.S)
```

6.4 Variant formulas

6.4.1 Letter substitution –

Ideally both stages would address variant formats of formulas. One variation would be allowing any English or Greek letter in place of another English or Greek letter, with the exception of e (Euler's number) and i, which often have special meaning. In the case of the variable i, perhaps our technique should attempt to determine whether i stands for an imaginary number; if not, allow letter substitution, if so, disallow letter substitution. If not enough information (such as a negative number under a square root sign) is present in a query to enable an educated guess, the variable i should be assumed to have special meaning and no letter substitution should be allowed. As with the treatment of i, information searchers would need to choose whether or to disallow letter substitution of Greek letters since those can sometimes have special meaning (alpha can stand for the first angle in a triangle, beta can stand for one square of a root, etc). Actual trials comparing results when allowing or disallowing substitution for these (or other letter) characters would confirm which retrieval practice performs better (statistically having more accurate results more often) in each case.

6.4.2 Term ordering –

The formulas 3x + 2 is equivalent to 2 + 3x or even 2 + x3, or for that matter possibly 3x + 1 + 1 or 3x + 4 - 2. Our technique did not capture such variations.

6.4.3 Other conceptual equivalence –

X times x is the same as x^2, but our approach would not capture that. Equivalencies such as this will be difficult to codify since each type of scenario will need to be both thought of and allowed for.

6.5 A Graph Structured Model

Several of the improvements described above to capture formula variation suggest a more fundamental look at formula structure as an directed acyclic graph. With proper modeling, formula variants should be able to be expressed as a series of homomorphic transformations from a query structure to a document structure. Missing nodes in the full equation structure (such as omission of the linear part of a quadratic equation) should also be accommodated by allowing for matching empty nodes in the structure. The advantage of developing a graph structured model for equations might enable application of the growing number of algorithms for graph matching [2,4].

6.6 Equation Grammars

Before our attempt at working on the NTCIR-10 Math Pilot Task, the authors were unable to find any papers in the literature on mathematics formulae search. However we have more recently found a paper on approaching equation structure as a structured grammar problem [5]. We will be studying whether there is other research along these lines which have techniques to be applied to math search.

7. SUMMARY

This paper has presented the approach and results of a keyword information retrieval approach to search for mathematical formulae in the NTCIR-10 Math Pilot Task. Both automatic and manual query development using this retrieval approach produced unsatisfactory results. It is clear that math search is both qualitatively and quantitatively different from ordinary search and will require significant development of new ideas and approaches to be able to achieve the goal of information retrieval for information needs in seeking scientific articles on the basis of formulas contained therein.

8. REFERENCES

[1] Akiko Aizawa, Michael Kohlhase and Iadh Ounis "NTCIR-10 Math Pilot Task Overview," in *this proceedings,* 2013.

[2] Li Chen, Amarnath Gupta, and M. Erdem Kurul. 2005. Efficient Algorithms for Pattern Matching on Directed Acyclic Graphs. In *Proceedings of the 21st International Conference on Data Engineering (ICDE '05)*. IEEE Computer Society, Washington, DC, USA, 384-385.

[3] James R. Driscoll, G. Theis, G. Billings "Using Database Schemas to Detect Relevant Information," *in Proceedings of TREC-3,* 1994

[4] Xinbo Gao, Bing Xiao, Dacheng Tao, Xuelong Li, "A survey of graph edit distance," Pattern Anal Applic (2010) 13:113–129.

[5] R. Nigel Horspool and John Aycock, "Analysis of Equation Structure using Least Cost Parsing," in *Proceedings of Sixth Intl. Workshop on Parsing Technologies (IWPT'2000*), Trento, Italy, Feb. 2000, pp. 307-308.

[6] Ray R. Larson, Ralph Moon, Jerome McDonough, Lucy Kuntz and Paul O'Leary. Cheshire II: Design and Evaluation of a Next-Generation Online Catalog System" IN: *ASIS '95: Proceedings of the 58th ASIS Annual Meeting*. Medford, NJ: Information Today, 1995. (p. 215-225).

9. APPENDIX -- Formula Extraction Details

The specific steps to identify the formula location within a document are as follows:

1. Read in the query document (an XML file) of multiple queries
2. Extract, parse and store the MathML content part of the query and query id number, using Python's regular expressions, for each query present in the queries document
    - Extract the queries.

pattern = re.compile('<cquery>\s+<m:math>(.?)</m:math',re.S)*

    - Extract the query IDs.

pattern = re.compile('<num>(.?)</num>',re.S)*

3. Read in the preliminary result which is a list in that contains entries of filenames, such as f000001.xhtml, and query IDs, such as FS-1, among other things.
4. For each query ID in the query file:
    - Get the query parts. For each line in the query:

pattern1 = re.compile('^(<m:\w)?.*?',re.S) # begins with an open tag <m:...*

pattern2 = re.compile('(.?)</m',re.S) # anything before an open tag*

pattern3 = re.compile('(</m:\w)',re.S) # begins with a close tag </m:*

These parts are each appended to a single list of all query parts, line by line, for each given query ID.

    - In the preliminary results, find all filenames matching the current query ID
    - For each found filename:
        - Find all formulas in that file.

pattern = re.compile('<m:annotation-xml id="id\d+" encoding="MathML-Content">(.?)</m:annotation-xml>',re.S)*

        - For each formula:
            - Get the formula id. *pattern = re.compile('<m:\w*\sid="id(\d+)"',re.S)*
            - Get the formula parts (using the same three regular expression patterns as were used for extracting query parts).
            - Compare each formula part to each query part, in order.
            - If all parts match between a query formula and a preliminary result file formula, use the current result file's formula ID as the formula ID for our final result.
            - Write the result to a final results file.
5. Repeat the above steps for each run