Indexing and Searching Mathematics in Digital Libraries

Architecture, Design and Scalability Issues

Petr Sojka and Martin Líška

Masaryk University, Faculty of Informatics, Botanická 68a, 602 00 Brno, Czech Republic
sojka@fi.muni.cz, 255768@mail.muni.cz

Abstract. This paper surveys approaches and systems for searching mathematical formulae in mathematical corpora and on the web. The design and architecture of our MIaS (Math Indexer and Searcher) system is presented, and our design decisions are discussed in detail. An approach based on Presentation MathML using a similarity of math subformulae is suggested and verified by implementing it as a math-aware search engine based on the state-of-the-art system, Apache Lucene.

Scalability issues were checked based on 324,000 real scientific documents from arXiv archive with 112 million mathematical formulae. More than two billions MathML subformulae were indexed using our Solr-compatible Lucene extension.

Keywords: math indexing and retrieval, mathematical digital libraries, information systems, information retrieval, mathematical content search, document ranking of mathematical papers, math text mining, MIaS, WebMIaS.

I do not seek. I find.
Pablo Picasso

1 Introduction

The solution to the problem of mathematical formulae retrieval lies at the heart of building digital mathematical libraries (DML). There have been numerous attempts to solve this problem, but none have found widespread adoption and satisfaction within the wider mathematics community. And as yet, there is no widely accepted agreement on the math search format to be used for mathematical formulae by library systems or by Google Scholar.

MathML standard by W3C has become the standard for mathematics exchange between software tools. Almost no MathML is written directly by authors—they typically prefer a compact notation of some TEX flavour such as LATEX or $\mathcal{AMS}$-LATEX. The designer of a search system for mathematics is thus faced with the task of converting data to a unifying format, and allowing DML users to use their prefered notation when posing queries. [$\mathcal{AMS}$]LATEX or other TEX flavour are the typical preferences; Presentation MathML or Content MathML are used only when available as outputs of a software system.

J.H. Davenport et al. (Eds.): Calculemus/MKM 2011, LNAI 6824, pp. 228–243, 2011.

During the integration of existing DMLs into larger projects such as EuDML [15], the unsolved math search problem becomes evident—D*M*L without math search support is an oxymoron. As our subject matter search has not lead to a satisfactory solution, we have designed and implemented [7] *new* robust solutions for retrieval of mathematical formulae.

Section 2 explores published facts about research done in the area of mathematics retrieval. Pros and cons of existing approaches are outlined, most of them being neither applicable nor satisfactory for digital library deployment. In Section 3 we present our design of scalable and extensible system for searching mathematics, taking into account not only inherent structure of mathematical formulae but also formula unification and subformulae similarity measures. Our evaluation of prototypical implementation above the Apache Lucene open source full-featured search engine library is presented in Section 4. The paper closes listing future work directions in Section 5 and a conclusion is summarised in Section 6.

> Computers are useless. They can only give you answers.
> Pablo Picasso

2 Approaches to Searching Mathematics

A great deal of research on has been already undertaken on searching mathematical formulae in digital libraries and on the web. Several such Mathematical Search Engines (MSE) have been designed in the past: MathDex, EgoMath, LaTeXSearch, LeActiveMath or MathWebSearch. In this section, we will briefly comment on each of these.

MathDex[1] (formerly MathFind [9]) is a result of a NSF-funded project headed by Robert Miner of Design Science[2]. It encodes mathematics as text tokens, and uses Apache Lucene as if searching for text. Using similarity with search terms, ranked results are produced by the search algorithm, matching *n*-grams of presentation MathML. The creators of MathDex report that most of the work was due to a necessary and extensive normalization of MathML—because of the fact that it uses several converters and filters to convert to XHTML + MathML—HTML (jtidy), TeX/LaTeX (blahtex, LaTeXML, Hermes), Word (Word+MathType), PDF (pdf2tiff+Infty). The algorithm of *n*-gram ranking has several drawbacks. For one thing, it cannot take many kinds of elementary mathematical equivalences into account, and it puts undue weight on variable names.

Contrary to its intentions, MathDex has not become a sustainable service to the mathematical community, although it has fueled research in the area of mathematics searching [16,17,1].

EgoMath[3] is being developed by Josef Mišutka as an extension of a full text websearch core engine Egothor (by Leo Galamboš, MFF UK Prague) [8] licenced under GPL. It uses presentation MathML for indexing and develops generalization algorithms and

[1] www.mathdex.com/
[2] www.ima.umn.edu/2006-2007/SW12.8-9.06/activities/Miner-Robert/index.html
[3] egomath.projekty.ms.mff.cuni.cz/egomath/

relevancy calculation to cope with normalization. As part of EgoThor evaluation, an MSE evaluation dataset is also being developed[4].

LaTeXSearch[5] is a search tool offered by Springer in SpringerLink. It searches directly in the TeX math string representations as provided by the authors of papers submitted to Springer in LaTeX sources. Some kind of text similarity matching is probably used. Since it is not open source, one can only guess the strategy for posing queries. Our experiments typically lead to a very low precision. Neither is there any definition of the article dataset available.

LeActiveMath[6] search has been developed as part of the ActiveMath-EU project. It is Lucene based, indexing string tokens from OMDoc with an OpenMath semantic notation. The document database format is internal since only documents authored for LeActiveMath learning environments are indexed.

MathWebSearch[7] is an MSE developed in Bremen/Saarbrücken by Kohlhase et al. [2] It is not based on full text searching, rather it adopts a semantic approach: it uses substitution trees in memory. Both presentation and content MathML is supported, together with OpenMath. It is exceptional in the fact that it primarily deals with semantics and uses its own engine, not being built on the Lucene engine, for math. Further development is now being pursued under LaMaPun architecture [6].

The comparison of math search systems is summarized in Table 1. All of the MSEs reviewed had some drawbacks regarding their employment in a digital mathematical library such as EuDML. This was our main motivation for designing a new one, primarily for the use in large scale libraries, such as EuDML or ArXiv.

Everything you can imagine is real.
Pablo Picasso

3 Design of MIaS

We have developed a math-aware, full-text based search engine called *MIaS* (Math Indexer and Searcher). It processes documents containing mathematical notation in MathML format. MIaS allows users to search for mathematical formulae as well as the textual content of documents.

Since mathematical expressions are highly structured and have no canonical form, our system pre-processes formulae in several steps to facilitate a greater possibility of matching two equal expressions with different notation and/or non-equal, but similar formulae. With an analogy to natural language searching, MIaS searches not only for whole sentences (whole formulae), but also for single words and phrases (subformulae down to single variables, symbols, constants, etc.). For calculating the relevance of the matched expressions to the user's query, MIaS uses a heuristic weighting of indexed terms, which accordingly affects scores of matched documents and thus the order of results.

[4] egomath.cythres.cz/dataset.py
[5] www.latexsearch.com/
[6] devdemo.activemath.org/ActiveMath2/
[7] search.mathweb.org/index.xhtml

Table 1. Comparison of math search systems

System	Input documents	Internal representation	Approach	α-eq.	Query language	Queries	Indexing core
MathDex	HTML, TEX/LATEX, Word, PDF	Presentation MathML (as strings)	syntactic	✗	?	text, math, mixed	Apache Lucene
LeActiveMath	OMDoc, OpenMath	OpenMath (as string)	syntactic	✗	OpenMath (palette editor)	text, math, mixed	Apache Lucene
LATEXSearch	LATEX	LATEX (as string)	syntactic	✗	LATEX	titles, math, DOI	?
MathWeb Search	Presentation MathML, Content MathML, OpenMath	Content MathML, OpenMath (substitution trees)	semantic	✔	QMath, LATEX, Mathematica, Maxima, Maple, Yacas styles (palette editor)	text, math, mixed	Apache Lucene (for text only)
EgoMath	Presentation MathML, Content MathML, PDF	Presentation MathML trees (as strings)	mixed	✗	LATEX	text, math, mixed	EgoThor
MIaS	any (well-formed) MathML	Canonical Presentation MathML trees (as compacted strings)	math tree similarity/ normalization	✔	$\mathcal{AMS}$-LATEX or MathML	text, math, mixed	Apache Lucene/Solr

3.1 System Workflow

The top-level indexing scheme is shown in Figure 1. A detailed view of the mathematical part is shown in Figure 2 on the next page.

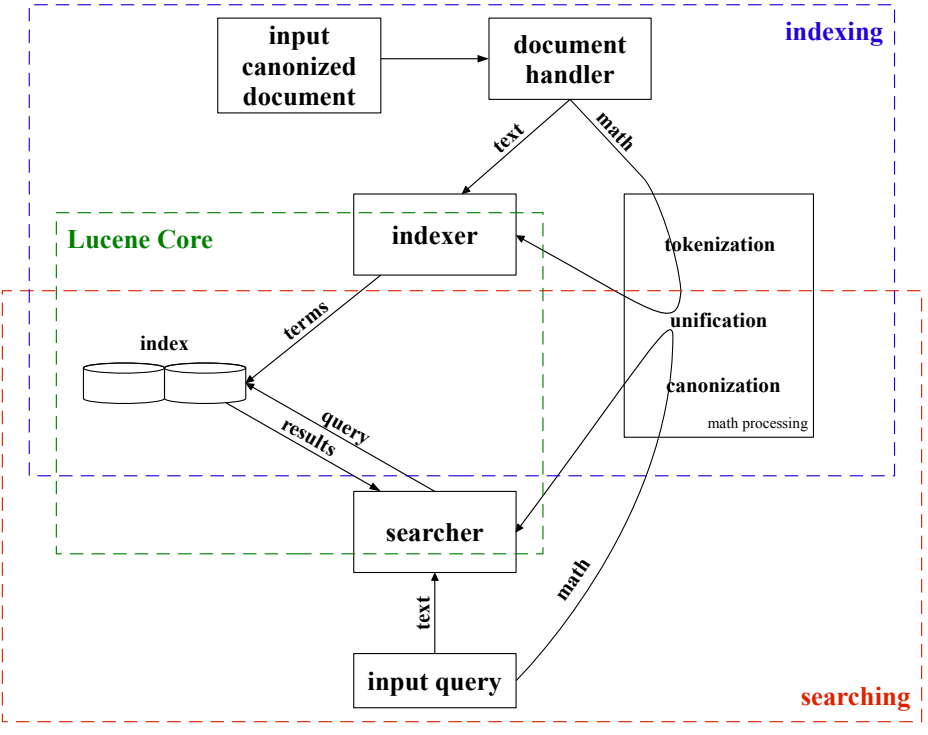


Fig. 1. Scheme of the system workflow

3.2 Indexing

MIaS is currently able to index documents in XHTML, HTML and TXT formats. As Figure 1 shows, the input document is first split into textual and mathematical parts. The textual content is indexed in a conventional way.

Mathematical expressions, on the other hand, are pre-analyzed in several steps to facilitate searches not only for exact whole formulae, but also for subparts (tokenization) and for similar expressions (formulae modifications). This addresses the issue of the static character of full-text search engines and creates several representations of each input formula all of which are indexed. Each indexed mathematical expression has a weight (relevancy score) assigned to it. It is computed throughout the whole indexing phase by individual processing steps following this basic rule of thumb—the more modified a formula and the lower the level of a subformula, the less weight is assigned to it.

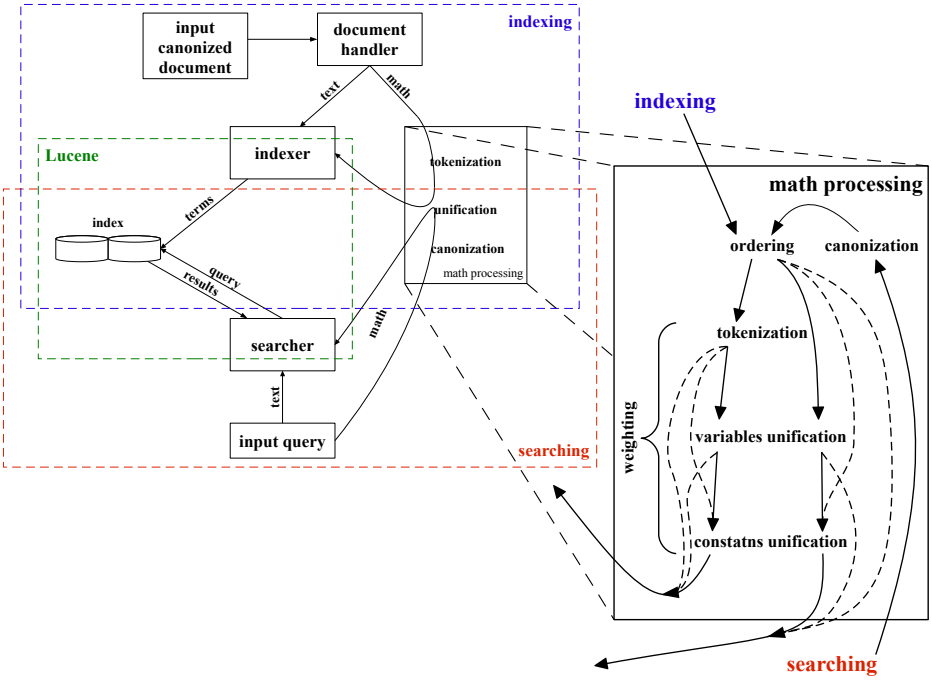


Fig. 2. Scheme of the MIaS workflow of math processing

At the end of all processing methods, formulae are converted from XML nodes to a compacted linear string form, which can be handled by the indexing core. Start and end XML tags are substituted by the tag name followed by an argument embraced by opening and closing parentheses. This creates abbreviated but still unambiguous representation of each XML node. For example, formula $a + b^2$, in MathML written as:

```
<math xmlns="http://www.w3.org/1998/Math/MathML">
  <mrow>
    <mi>a</mi>
    <mo>+</mo>
    <msup> <mi>b</mi><mn>2</mn></msup>
  </mrow>
</math>
```

is converted to "`math(mrow(mi(a)mo(+)msup(mi(b)mn(2))))`" and this string is then indexed by Lucene.

3.3 Tokenization

Tokenization is a straightforward process of obtaining subformulae from an input formula. MIaS makes use of Presentation MathML markup where all logical units are

enclosed in XML tags which makes obtaining all subformulae a question of tree traversal. The inner representation of each formula is an XML node encapsulating all the member child nodes. This means the highest level formula—as it appears in the input document—is represented by a node named "math". All logical subparts of an input formula are obtained and passed on to modification algorithms.

3.4 Formulae Modifications

MIaS performs three types of unification algorithms, the goal of which is to create several more or less generalized representations of all formulae obtained through the tokenization process. These steps allow the system to return similar matches to the user query while preserving the formula structure and α-equality.

3.5 Ordering

Let us take a simple example: $a + 3$ and the query $3 + a$. This would not match even though it is perfectly equal. This is why a simple ordering of the operands of the commutative operations, addition and multiplication, is used. It tries to order arguments of these operations in the alphabetical order of the XML nodes denoting the operands whenever possible—it considers the priority of other relevant operators in the formula. The system applies this function to the formula being indexed as well as to the query expression. Applied to the example above, the XML node denoting variable a is named "mi", the node denoting number 3 is named "mn". "mi"<"mn" therefore $3 + a$ would be exchanged for $a + 3$ and would match.

3.6 Unification of Variables

Let us take another example: $a + b^a$ and $x + y^x$. Again, these would not match even though the difference is only in the variables used. MIaS employs a process that unifies variables in expressions while taking bound variables into account. All variables are substituted for unified symbols (ids) in both the indexing and searching phases. Applied to the example, both expressions would unify to $\mathrm{id}_1 + \mathrm{id}_2^{\mathrm{id}_1}$ and would match. This process is not applied to single symbols—this would lead to the indexing of millions of ids and searching for any symbol would end up matching all of the documents containing it.

3.7 Unification of Constants

This is a strightforward process of substituting all the numerical constants for one unified symbol (const). This obviates the need for the exact values of constants in user queries. In some situations however, this can be too much of a generalization. As well as in the case of the variables, stand-alone numerical constants are not unified for the same obvious reason.

3.8 Formulae Weighting

During the searching phase, a query can match several terms in the index. However one match can be more important to the query than another, and the system must consider

this information when scoring matched documents. For mathematical formulae the system makes use of the processing operations described above since they all produce expressions more generalized than the input ones.

It is impossible to assemble a weighting function that is exactly right. Such a function should consider a document base on which the system will run as well as the established customs in a particular scientific field. We tried to create a complex and robust weighting function that would be appropriate to many fields.

The original unchanged untokenized formula should of course have the greatest weight, but the precision of the ordered representation is not compromised at all, so it should have the same weight. In fact, if the ordering process changes the order of some members in an expression, the original formula is not indexed at all. The starting weight for such a representation is 1.

The tokenization process should naturally lower the weight of the subformulae since they are deeper in the structure and therefore less important to the overall formula. When a user who is searching for $a + b$ finds two documents, the first containing $a + b$ and the second containing $\frac{2}{a+b}$, the first should score more and appear higher in the results, as it matches in higher *level* of MathML expression tree. Hence the tokenization process reduces the weight of the subformulae according to the level coefficient $l < 1$.

Both unification algorithms produce representations that are more generalized than their input expressions. They have a higher probability of matching, and should therefore score less. The unification of variables alters the weight of the result formula by coefficient $v < 1$, unification of number constants uses coefficient $c < 1$.

Theoretically, two equally unified subformulae matched on the same level of differently complex parent formulae would have the same score. For example

$$a + b^{3+a}$$

and

$$\frac{\int_0^\infty 25b^2\,db}{3+a} + \frac{d-e}{b} + 100ab$$

with the query $3 + a$. Both matches are not unified, and both are found on the third level. Analogously to conventional full-text engines which discriminate documents with more tokens than others, we use information about the complexity of parent formulae. More specifically, an initial weight of 1 is multiplied by the inverse number of nodes of a whole parent expression.

According to this model, each formula has a weight attribute indexed alongside itself, which belongs to the interval $(0, 1\rangle$. Weight w of the subformula contained on a certain level in a parent formula with the number of nodes (n) can be calculated in particular situations as follows:

- no changes made: $w = \frac{l^{\text{level}}(1+v+c+vc)}{n}$
- unified variables: $w = \frac{l^{\text{level}}(v+vc)}{n}$
- unified constants: $w = \frac{l^{\text{level}}(c+vc)}{n}$
- unified both variables and constants: $w = \frac{l^{\text{level}}(vc)}{n}$.

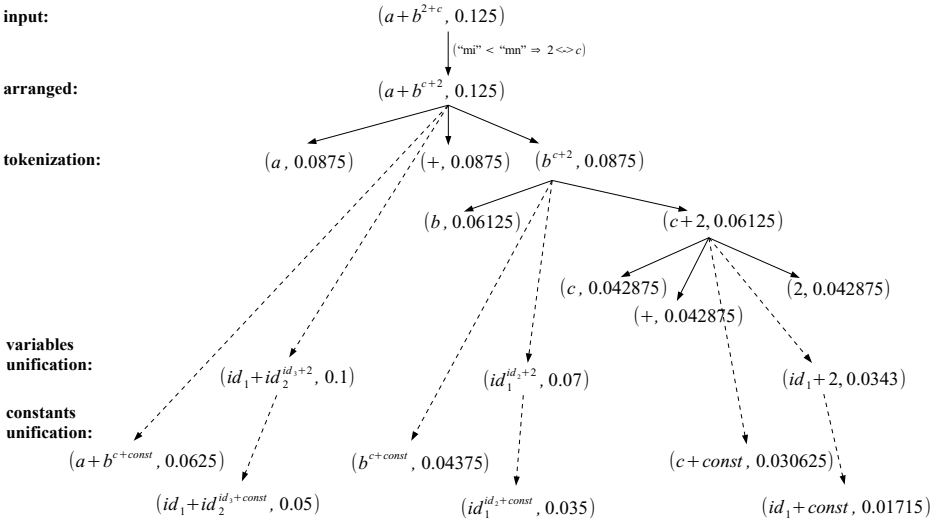


Fig. 3. Example of formula preprocessing. Ordered pairs are (<expression written naturally>, <it's weight>). All expressions as shown are indexed, except for the original one.

See Section 3.9 for details.

To fine tune the weighting parameters, we developed a tool with verbose output in which the behavior of the model can be observed and tested. A sample from the tool mentioned above is shown in Table 2 on the next page.

We have come to the conclusion that the unification of variables interferes less with original formula meaning than the unification of number constants. For this reason, its coefficient should be higher—i.e., less discriminating. The main question then became, how discriminating the level coefficient should be. Our empirical deduction is that going deeper in a structural tree should be discriminating, the precise match on a lower level should still score more than any unified formula on the level above, as could be seen in Table 2: $\frac{1}{a+3}$ (row 5) is an exact match on the second level and its score is higher than unified expressions matched on the first level (rows 2, 3 and 4).

This led us to the valuation of level weighting coefficient $l = 0.7$, unification weighting coefficient $v = 0.8$ and constant weighting coefficient $c = 0.5$.

In Figure 3 the whole formula preprocessing process is illustrated together with its subformulae weightings.

3.9 Searching

In the search phase, user input is again split into mathematical and textual parts. Formulae are then reprocessed in the same way as in the indexing phase, except for tokenization—which we doubt that users are likely to query, for example $\frac{a+b}{c}$ wanting to find documents only with occurrences of variable c. That means the queried

Table 2. Example of weighting function on several formulae. Original query is $a + 3$—all queried expressions are $a + 3$, $\mathrm{id}_1 + 3$, $a + \mathrm{const}$, $\mathrm{id}_1 + \mathrm{const}$.

Formula	Indexed Expressions	Score	Matched
$a + 3$	$0.25=[a + 3]$, $0.2=[\mathrm{id}_1 + 3]$, $0.175=[a, 3, +]$, $0.125=[a + \mathrm{const}]$, $0.1=[\mathrm{id}_1 + \mathrm{const}]$	2.7	$0.1[\mathrm{id}_1 + \mathrm{const}] + 0.25[a + 3] + 0.2[\mathrm{id}_1 + 3] + 0.125[a + \mathrm{const}]$
$b + 3$	$0.25=[b + 3]$, $0.2=[\mathrm{id}_1 + 3]$, $0.175=[b, +, 3]$, $0.125=[b + \mathrm{const}]$, $0.1=[\mathrm{id}_1 + \mathrm{const}]$	1.2	$0.1[\mathrm{id}_1 + \mathrm{const}] + 0.2[\mathrm{id}_1 + 3]$
$a + 5$	$0.25=[a + 5]$, $0.2=[\mathrm{id}_1 + 5]$, $0.175=[a, +, 5]$, $0.125=[a + \mathrm{const}]$, $0.1=[\mathrm{id}_1 + \mathrm{const}]$	0.9	$0.1[\mathrm{id}_1 + \mathrm{const}] + 0.125[a + \mathrm{const}]$
$c + 10$	$0.25=[c + 10]$, $0.2=[\mathrm{id}_1 + 10]$, $0.175=[c, +, 10]$, $0.125=[c + \mathrm{const}]$, $0.1=[\mathrm{id}_1 + \mathrm{const}]$	0.4	$0.1[\mathrm{id}_1 + \mathrm{const}]$
$\frac{1}{a+3}$	$0.16667=[\frac{1}{a+3}]$, $0.13334=[\frac{1}{\mathrm{id}_1+3}]$, $0.11667=[1, a + 3]$, $0.09334=[\mathrm{id}_1 + 3]$, $0.08334=[\frac{\mathrm{const}}{a+\mathrm{const}}]$, $0.08167=[+, 3, a]$, $0.06667=[\frac{\mathrm{const}}{\mathrm{id}_1+\mathrm{const}}]$, $0.05833=[a + \mathrm{const}]$, $0.04667=[\mathrm{id}_1 + \mathrm{const}]$	1.26	$0.04667[\mathrm{id}_1 + \mathrm{const}] + 0.11667[a + 3] + 0.09334[\mathrm{id}_1 + 3] + 0.05833[a + \mathrm{const}]$
$\frac{1}{b+3}$	$0.16667=[\frac{1}{b+3}]$, $0.13334=[\frac{1}{\mathrm{id}_1+3}]$, $0.11667=[b + 3, 1]$, $0.09334=[\mathrm{id}_1 + 3]$, $0.08334=[\frac{\mathrm{const}}{b+\mathrm{const}}]$, $0.08167=[b, 3, +]$, $0.06667=[\frac{\mathrm{const}}{\mathrm{id}_1+\mathrm{const}}]$, $0.05833=[b + \mathrm{const}]$, $0.04667=[\mathrm{id}_1 + \mathrm{const}]$	0.56	$0.04667[\mathrm{id}_1 + \mathrm{const}] + 0.09334[\mathrm{id}_1 + 3]$
$\frac{1}{a+5}$	$0.16667=[\frac{1}{a+5}]$, $0.13334=[\frac{1}{\mathrm{id}_1+5}]$, $0.11667=[1, a + 5]$, $0.09334=[\mathrm{id}_1 + 5]$, $0.08334=[\frac{\mathrm{const}}{a+\mathrm{const}}]$, $0.08167=[a, 5, +]$, $0.06667=[\frac{\mathrm{const}}{\mathrm{id}_1+\mathrm{const}}]$, $0.05833=[a + \mathrm{const}]$, $0.04667=[\mathrm{id}_1 + \mathrm{const}]$	0.42	$0.04667[\mathrm{id}_1 + \mathrm{const}] + 0.05833[a + \mathrm{const}]$
$\frac{1}{c+10}$	$0.16667=[\frac{1}{c+10}]$, $0.13334=[\frac{1}{\mathrm{id}_1+10}]$, $0.11667=[1, c + 10]$, $0.09334=[\mathrm{id}_1 + 10]$, $0.08334=[\frac{\mathrm{const}}{c+\mathrm{const}}]$, $0.08167=[+, c, 10]$, $0.06667=[\frac{\mathrm{const}}{\mathrm{id}_1+\mathrm{const}}]$, $0.05833=[c + \mathrm{const}]$, $0.04667=[\mathrm{id}_1 + \mathrm{const}]$	0.19	$0.04667[\mathrm{id}_1 + \mathrm{const}]$

expressions are first ordered, then unified. This produces several representations which are connected to the final query by the logical OR operator.

Textual query terms are connected to the final query by the logical AND operator. Therefore by specifying a text term we can narrow down the results, because each returned document must have the term contained. When more than one text term is specified, they are implicitly connected to the text query by the OR operator which means at least one term should occur in the result. We can also explicitly state preferences about each text term—whether it needs to occur in the result or not.

As stated above, the final query, without having explicitly stated occurrences of text terms, is in the logical form of $(\mathrm{formula}_1 \vee \ldots \vee \mathrm{formula}_n) \wedge (\mathrm{term}_1 \vee \ldots \vee \mathrm{term}_n)$.

In order to counterbalance the weight of the textual and mathematical parts of the query, the score of the matched formulae are additionally multiplied by number of nodes

the matching query consists of. This results in more complex mathematical queries scoring more.

A very positive value has its price in negative terms...
the genius of Einstein leads to Hiroshima.
Pablo Picasso

4 Evaluation

For large scale evaluation, we needed an experimental implementation and a corpus of mathematical texts.

4.1 Implementation

The Math Indexer and Searcher is written in Java. The role of full-text indexing and searching core is performed by Apache Lucene 3.1.0. The mathematical part of document processing can be seen as a standalone pluggable extension to any full-text library, however it would need custom integration for each one. In the case of Lucene, a custom Tokenizer (MathTokenizer) has been implemented.

For the textual content of documents, Lucene's StandardAnalyzer is employed. In MathTokenizer, TermAttributes are used for carrying strings of math expressions and PayloadAttribute for storing weights of formulae.

The question now is, how should the weights of formulae be taken into consideration in the overall score of matched documents. Lucene's practical scoring function for every hit document d by query q with each query term t is as follows:

$$score(q,d) = coord(q,d) \cdot queryNorm(q) \cdot \sum_{t\ in\ q} \Big(tf(t \text{ in } d) \cdot idf(t)^2 \cdot t.getBoost() \cdot norm(t,d) \Big)$$

It is described in detail at `http://lucene.apache.org/java/3_1_0/api/core/index.html?org/apache/lucene/search/Similarity.html`.

When searching for mathematical formulae, their weights need to be considered in the final score of the document. The resulting MIaS scoring function adds another parameter to the basic function—weight w of one matched formula:

$$score(q,d) = coord(q,d) \cdot queryNorm(q) \cdot \cdot \sum_{t\ in\ q} \Big(tf(t \text{ in } d) \cdot avg(w) \cdot idf(t)^2 \cdot t.getBoost() \cdot norm(t,d) \Big) \quad (1)$$

If a document contains the same formula more than once (each occurrence can have different weight assigned), the average value of all the weights is taken into consideration ($avg(w)$).

Let's take a simplified version of the function (1). Specifically, let us not to consider normalization factor $queryNorm(q)$, inverse document frequency $idf(t)^2$ and document/field boost and length factor $norm(t,d)$:

$$score(q,d) = coord(q,d) \cdot \sum_{t\ in\ q} (tf(t \text{ in } d) \cdot avg(w) \cdot t.getBoost()) \quad (2)$$

and follow the example in Table 2 on page 237. Let's consider we query a document containing only two formulae $b + 3$ and $\frac{1}{a+3}$ (rows 2 and 5). During indexing time, preprocessing creates several more representations, all of which are indexed (shown in the second column). The query is $a + 3$ which is expanded by query preprocessing to the final query that takes the form of $a + 3 \vee \mathrm{id}_1 + 3 \vee a + \mathrm{const} \vee \mathrm{id}_1 + \mathrm{const}$. Column 4 shows which actual expressions will match the query for each particular input formula. $coord(q, d)$ will be $\frac{4}{4}$ because all four of the four query terms found a match. Query terms $a + 3$ matched only one indexed term and its weight is 0.11667; query term $a + \mathrm{const}$ also matched only one indexed term and its weight 0.05833; query term $\mathrm{id}_1 + 3$ matched two indexed terms with weights 0.2 and 0.09334 so its average is 0.14667; finally the last query term $\mathrm{id}_1 + \mathrm{const}$ matched two indexed expressions with weights 0.1 and 0.04667 so its average is 0.07335. *t.getBoost*() is a query time boosting factor and as stated in Section 3.9, we use the number of XML nodes of the original query formula—in this example it is 4. The resulting score of the whole document is then

$$\frac{4}{4} \cdot \left((1 \cdot 0.11667 \cdot 4) + (1 \cdot 0.05833 \cdot 4) + (2^{\frac{1}{2}} \cdot 0.14667 \cdot 4) + (2^{\frac{1}{2}} \cdot 0.07335 \cdot 4)\right)$$

4.2 Corpus of Mathematical Documents MREC

A document corpus MREC with 324,060 scientific documents (version 2011.3.324) was initially used to evaluate the behaviour of the system we modelled. The documents come from the arXMLiv project that is converting document sets from arXiv into XHTML + MathML (both Content and Presentation) [13]. At the time of testing, our system was not yet able to process mixed MathML markup so preprocessing in the sense of filtering out unwanted markup was needed. The resulting corpus size was 53 GB uncompressed, 6.7 GB compressed. Documents contained 112,055,559 formulae in total, of which 2,129,261,646 mathematical expressions were indexed. The resulting index size was approx. 45 GB.

We were able to gather even greater amount of documents in MREC corpus version 2011.4.439 to test our indexing system. This corpus consists of 439,423 arXMLiv documents containing 158,106,118 mathematical formulae. 2,910,314,146 expressions were indexed and the resulting size of the index is 63 GB. Sizes of uncompressed and compressed corpora size are 124 GB and 15 GB, respectively.

Mentioned MREC corpora are available to the community for download from MREC web page `http://nlp.fi.muni.cz/projekty/eudml/MREC/` so that other math indexing engines could be compared with MIaS on the same data.

4.3 Results

Math Indexer and Searcher demonstrated the ability to index and search a relatively vast corpus of real scientific documents. Its usability is highly elevated thanks to its preprocessing functions together with formulae weighting model. The ability to search for exact and similar formulae and subformulae, more so with customizable relevancy computation, demonstrates an unquestionable contribution to the whole search experience.

Table 3. Scalability test results (run on 32 GB RAM, quad core AMD Opteron™ Processor 850 driven machine)

Documents	Input formulae	Indexed formulae	Indexing time [min]	Average query time [ms]
10,000	3,450,114	65,194,737	39.15	32
50,000	17,734,342	334,078,835	201.68	178
200,000	70,102,960	1,316,603,055	889.28	576
324,060	112,055,559	2,129,261,646	1,292.16	789

It is very difficult, if not impossible, to completely verify the correctness of the theoretical considerations made in the previous sections and thus correctness of search results. For this purpose, a sufficiently large corpus of documents with fully controlled content would be needed. For any assembled query, there should exist beforehand a complete list of the documents ordered by their relevance to the query to compare the actual results to.

We have applied an empirical approach to the evaluation so far. For these purposes we have created a demo web interface WebMIaS which is publicly available on the MIaS web page `http://nlp.fi.muni.cz/projekty/eudml/mias/`. It works over MREC corpora discussed in the Section 4.2 Additionally, for the latest MREC corpus we have implemented and added demanded snippet generation and mathematical match highlighting in hit list. Preliminary version of this functionality is available.

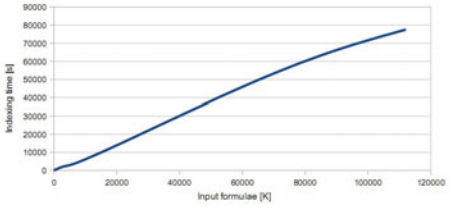
Our WebMIaS interface supports queries in two different notations—in $\mathcal{AMS}$-LaTeX and MathML. Mathematical queries are additionally canonized using XSLT transformations from UMCL library [4,3] to improve the query and to avoid notation flaws restraining proper results retrieval. Portability of the interface is increased by using MathJax for rendering of mathematical formulae in snippets.

4.4 Scalability Testing and Efficiency

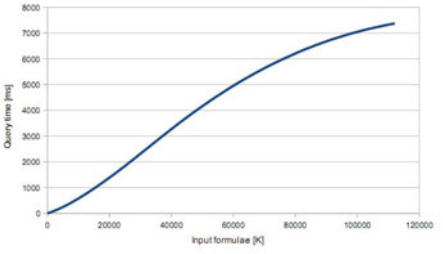
We have devised a scalability test to see how the system behaves with an increasing number of documents and formulae indexed. Subsets containing 10,000, 50,000, 200,000 and the complete 324,060 documents were gradually indexed and several values were measured: the number of input formulae, the number of indexed formulae, the indexing time and the average query time.

The number of input formulae indicate how rich a particular subset was in formulae; the number of indexed formulae should illustrate their complexity. Moreover both should indicate whether indexing and query time are dependent on the number of documents or specially on the formulae they contain. For measuring the average query time, we queried each created index with the same set of differently complex queries (mixed, non-mixed, more/less complex single/multiple formulae) computing the average time. The results are shown in Table 3 and in the form of diagrams in Figure 4 on the next page.

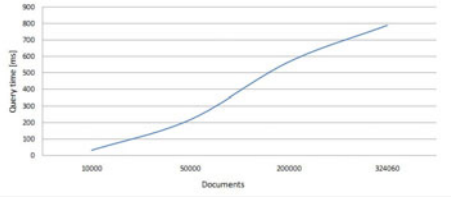
MREC version 2011.4.439 was indexed using improved and optimised algorithms and ran on a different machine. Therefore it cannot be compared to measured values



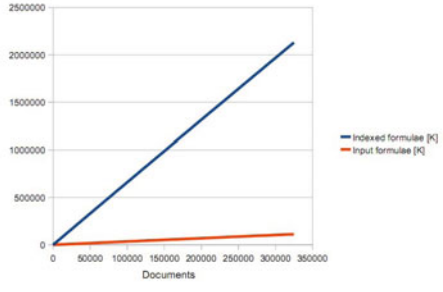
(a) Number of input formulae vs. indexing time



(b) Number of input formulae vs. query time



(c) Number of documents vs. query time



(d) Number of formulae vs. number of documents

Fig. 4. Scalability diagrams

shown in tables 3 on the facing page and 4. Indexing time of this corpus was 1378.82 ms, e.g. almost 23 hours.

He can who thinks he can, and he can't who thinks he can't.
This is an inexorable, indisputable law.
Pablo Picasso

5 Open Issues, Future Work

We are now awaiting heterogeneous MathML data collected by the EuDML project, that has been generated from born-digital [meta]data [10], from born-digital PDFs [5] or from math OCR [14].

It is evident that some kind of normalization of MathML will be a necessity. We have opted for Canonical MathML [4,3] as normalization MathML format and are using software library UMCL supporting it. Our latest experiments with canonical form of MathML generated by UMCL shows that it not only increases fairness of similarity ranking, but also helps to match a query against the indexed form of MathML. We are also working hard on snippets generation and on matched formulae visualization.

Another area of long-term research planned is supporting Content MathML, in a way similar to the current handling of Presentation MathML. The architectural design

is open to it, but as most of math within EuDML will be in Presentation MathML taken from PDFs, this is not currently a high priority.

> I am always doing that which I can not do, in order that I may learn how to do it.
> Pablo Picasso

6 Conclusions

We have presented an approach to mathematics searching and indexing—the architecture and design of the MIaS system. The feasibility of our approach has been verified on large corpora of real mathematical papers from arXMLiv. Scalability tests have confirmed that the computing power needed for fine math similarity computations is readily available; this would allow the use of this technology for projects on a European or world-wide scale.

Acknowledgements. This work has been in part financed by the European Union through its Competitiveness and Innovation Programme (Information and Communications Technologies Policy Support Programme, "Open access to scientific information", Grant Agreement no. 250,503). We thank anonymous reviewers for their improvement and future work suggestions, Michal Růžička for help with figure drawings and web form of MIaS interface, and Peter Mravec for collecting MREC.

References

1. Altamimi, M., Youssef, A.S.: A Math Query Language with an Expanded Set of Wildcards. Mathematics in Computer Science 2, 305–331 (2008), http://dx.doi.org/10.1007/s11786-008-0056-4
2. Anca, Ş.: Natural Language and Mathematics Processing for Applicable Theorem Search. Master's thesis, Jacobs University, Bremen (August 2009), https://svn.eecs.jacobs-university.de/svn/eecs/archive/msc-2009/aanca.pdf
3. Archambault, D., Berger, F., Moço, V.: Overview of the "Universal Maths Conversion Library". In: Pruski, A., Knops, H. (eds.) Assistive Technology: From Virtuality to Reality: Proceedings of 8th European Conference for the Advancement of Assistive Technology in Europe AAATE 2005, Lille, France, pp. 256–260. IOS Press, Amsterdam (September 2005)
4. Archambault, D., Moço, V.: Canonical MathML to Simplify Conversion of MathML to Braille Mathematical Notations. In: Miesenberger, K., Klaus, J., Zagler, W., Karshmer, A. (eds.) ICCHP 2006. LNCS, vol. 4061, pp. 1191–1198. Springer, Heidelberg (2006), http://dx.doi.org/10.1007/11788713_172
5. Baker, J.B., Sexton, A.P., Sorge, V.: Extracting Precise Data on the Mathematical Content of PDF Documents. In: Sojka [11], pp. 75–79, http://dml.cz/handle/10338.dmlcz/702535
6. Grigore, M., Wolska, M., Kohlhase, M.: Towards context-based disambiguation of mathematical expressions. Math-for-Industry Lecture Note Series, vol. 22, pp. 262–271 (December 2009)
7. Líška, M.: Vyhledávání v matematickém textu (in Slovak), Searching Mathematical Texts. Bachelor Thesis, Masaryk University, Brno, Faculty of Informatics (advisor: Petr Sojka) (2010), https://is.muni.cz/th/255768/fi_b/?lang=en

8. Mišutka, J., Galamboš, L.: Extending Full Text Search Engine for Mathematical Content. In: Sojka [11], pp. 55–67, http://dml.cz/dmlcz/702546
9. Munavalli, R., Miner, R.: MathFind: A Math-Aware Search Engine. In: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2006, pp. 735–735. ACM, New York (2006), http://doi.acm.org/10.1145/1148170.1148348
10. Růžička, M., Sojka, P.: Data Enhancements in a Digital Mathematics Library. In: Sojka [12], pp. 69–76, http://dml.cz/dmlcz/702575
11. Sojka, P. (ed.) Towards a Digital Mathematics Library, Birmingham, UK. Masaryk University (July 2008), http://www.fi.muni.cz/~sojka/dml-2008-program.xhtml
12. Sojka, P. (ed.) Towards a Digital Mathematics Library, Paris, France. Masaryk University (July 2010), http://www.fi.muni.cz/~sojka/dml-2010-program.html
13. Stamerjohanns, H., Kohlhase, M., Ginev, D., David, C., Miller, B.: Transforming Large Collections of Scientific Publications to XML. Mathematics in Computer Science 3, 299–307 (2010), http://dx.doi.org/10.1007/s11786-010-0024-7
14. Suzuki, M., Tamari, F., Fukuda, R., Uchida, S., Kanahori, T.: INFTY — An integrated OCR system for mathematical documents. In: Vanoirbeek, C., Roisin, C., Munson, E. (eds.) Proceedings of ACM Symposium on Document Engineering 2003, Grenoble, France, pp. 95–104. ACM, New York (2003)
15. Sylwestrzak, W., Borbinha, J., Bouche, T., Nowiński, A., Sojka, P.: EuDML—Towards the European Digital Mathematics Library. In: Sojka [12], pp. 11–24, http://dml.cz/dmlcz/702569
16. Youssef, A.S.: Roles of Math Search in Mathematics. In: Borwein, J., Farmer, W. (eds.) MKM 2006. LNCS (LNAI), vol. 4108, pp. 2–16. Springer, Heidelberg (2006), http://dx.doi.org/10.1007/11812289_2
17. Youssef, A.S.: Methods of Relevance Ranking and Hit-Content Generation in Math Search. In: Kauers, M., Kerber, M., Miner, R., Windsteiger, W. (eds.) MKM/CALCULEMUS 2007. LNCS (LNAI), vol. 4573, pp. 393–406. Springer, Heidelberg (2007)