MathWebSearch 0.4
A Semantic Search Engine for Mathematics

Michael Kohlhase[1,2], Ştefan Anca[1,2], Constantin Jucovschi[1], Alberto González Palomo[1,3], and Ioan A. Şucan[4]

[1] Safe and Secure Cognitive Systems, DFKI Lab Bremen
$firstname.$lastname@dfki.de
[2] Computer Science, Jacobs University Bremen
$initial.$lastname@jacobs-university.de
[3] Computer Science, Saarland University
Alberto.Gonzalez@matracas.org
[4] Computer Science, Rice University,
isucan@rice.edu

Abstract. We present a search engine for mathematical formulae. The MathWebSearch system harvests the web for content representations of formulae and indexes them with substitution tree indexing. In version 0.4 we have parallelized and distributed the search server and augmented the web interface with a new JavaScript-based visual editor for content math formulae. Furthermore, we have extended the query language by generalization, variants, unification, and text search facilities, which can also be mixed.
Our experiments show that this architecture results in a scalable application.

1 Introduction

As the world of information technology grows, being able to quickly search data of interest becomes one of the most important tasks in any kind of environment, be it academic or not. We present the MathWebSearch system; a search engine that addresses the problem of searching mathematical formulae from a semantic point of view i.e. finds formulae by their structure and meaning not via their presentation (as a standard engine like Google does).

In [KŞ06] we have presented the motivation, query language, and web front end of MathWebSearch 0.2. In [KK07] we have re-examined the value proposition of semantic search for mathematical knowledge homing in on the benefits and sacrifices for the user induced by the various search approaches [You06,MM06,LM06]. The result of this analysis is MathWebSearch 0.4 which we describe in this paper. The new version features significant efficiency gains, new management features, advanced searching capabilities, and a new user interface. The MathWebSearch system (see [Mat07] for details) is released under the Gnu General Public License [Fre91]. MathWebSearch 0.3 which features some of the extensions reported in this paper is available at `http://search.mathweb.org`, an

integrated prototype of MathWebSearch 0.4 will be available for testing at `http://betasearch.mathweb.org` in April 2008.

In the next section we will introduce the new search capabilities by looking at the types of queries MathWebSearch supports, provide a use case for each and show how naturally one can express the queries. Then we (section 2) show how the new user interface supports formula editing in such queries and in section 3 we get more into the the technical side of MathWebSearch. For that, a brief description of the new MathWebSearch infrastructure will be given which now supports distribution and work balance among several computers, hence becoming more scalable. Afterwards, we will describe the process which a mathematical expression undergoes before it is inserted in the index. Further, we have a brief look at the data structure used to index the terms and at the basics of performing search operations. It will then continue with the part of result generation and combining these with the searches over natural language.

1.1 A Unification-Based Query Language for Mathematics

As we have already mentioned, the power of MathWebSearch consists in its capability of indexing mathematical content by using formulae structure and semantics. Even more power is gained if we combine it with a standard search engine of natural language. This fusion makes the search engine more expressive and context driven. More details about this fusion is given in the section 4.

Very often one finds himself in the position that he remembers parts of a formula but does not remember details like coefficients or arguments. This is the typical use case where **instantiation queries** might come of use as they find indexed terms which are obtained after replacing the parts marked by the user as unknown with some terms i.e. they become instantiated. For example consider an engineer trying to solve a mathematical problem such as finding the power of a given signal $s(t)$. Of course our engineer is well-versed in signal processing and remembers that a signal's power has something to do with integrating its square, but has forgotten the details of how to compute the necessary integrals. He will therefore call up MathWebSearch to search for something that looks like

$$\int_{\boxed{max}}^{\boxed{min}} \boxed{f}(x)^2 dx \tag{1}$$

Here and everywhere we mark query variables as named boxes[5]. One of the search results will be the page containing the term (Parseval's theorem)

$$\frac{1}{T}\int_0^T s^2(t)dt = \sum_{k=-\infty}^{\infty} \|c_k\|^2 \tag{2}$$

[5] Note that this alredy gives us a more expressive query language than e.g. regular expressions supported by some text-based search engine, since we can use variable co-occurrences to query for co-occurring subterms.

about the energy of a signal s. This not only confirms the definition of the energy (in the surrounding text), but also gives a way to compute it via the Fourier coefficients c_k of s. Note that here the default setting pays off that instructs the search engine to also report subterms of indexed terms which match the given search expression.

Let us now consider a student who encounters $\int_{\mathbb{R}^2} |\sin(t)\cos(t)|dt$ and wishes to know if there are any mathematical statements (like theorems, identities, inequalities) that can be applied to it. Indeed, there are many such statements (for example Hölder's inequality), they can be found using **generalization queries**. The idea behind answering generalization queries is that the index marks universal[6] variables in subterms as generalization targets. Hence the search engine looks for terms in the index which after instantiating the universal identifiers become equal to the query. For our example, we have in the index the term (we reuse the box notation for generalization targets in the index)

$$\int_{\boxed{D}} \left|\boxed{f}(\boxed{x})\boxed{g}(\boxed{x})\right| d\boxed{x} \leq \left(\int_{\boxed{D}} \left|\boxed{f}(\boxed{x})\right|^p d\boxed{x}\right)^{\frac{1}{p}} \left(\int_{\boxed{D}} \left|\boxed{g}(\boxed{x})\right|^q d\boxed{x}\right)^{\frac{1}{q}} \tag{3}$$

which the search engine instantiates $\boxed{x} \mapsto t, \boxed{f} \mapsto sin, \boxed{g} \mapsto cos, \boxed{D} \mapsto \mathbb{R}^2$ in order to find the generalization query. Note that the variant query $\int_{\mathbb{R}^2} |\sin(t)\cos(2t)|dt$ will not find Hölder's inequality since that would introduce inconsistent substitutions $\boxed{x} \mapsto t$ and $\boxed{x} \mapsto 2t$.

A very similar idea is used in **variation queries** where the indexed terms are searched to match the seach expression but only renamings of generic terms are allowed. This type of queries prove to be helpful when the structure of the term needs to be maintained.

Sometimes, however, one is in the position that the searching criteria is somewhere between instantiation queries (i.e. parts are unknown) and generalization queries (parts are probably instantiated already). In this case we give the possibility to pose **unification queries**. As the name suggests, the query just finds terms which are unifiable with the search expression. A query like $g^2\cos(\boxed{x}) + b\sin(\sqrt{y})$ would match the term $\boxed{a}\cos(\boxed{t}) + \boxed{b}\sin(\boxed{t})$ as we can substitute $\boxed{x} \mapsto \sqrt{y}, \boxed{t} \mapsto \sqrt{y}, \boxed{a} \mapsto g^2, \boxed{b} \mapsto b$ to get the term $g^2\cos(\sqrt{y}) + b\sin(\sqrt{y})$.

It is easy to see that unification queries include but are not limited to the results of the instantiation, generalization and variation queries. Hence one might use it to combine the other three search queries. Sometimes, however, it might be challenging to see the substitutions that the search engine applied to both indexed term and search term for unification. For that reason, we introduce in a query language that supports all four previously mentioned query types and

[6] We consider an identifier as universal, if it can be instantiated without changing the truth value of the containing expression. In formal repreentations like first-order logic, such variable occurrences can be effectively computed, but in semi-formal settings like mathematical textbooks, they have to be approximated by heuristical methods; see the discussion in the conclusion for details

supports "*where*" clauses. An example for such a query would be:

$$\text{select instance } \left(\int_{\boxed{DOM}} \frac{1}{\sqrt{2\pi}} \exp\{B\} \right) \text{ where } B{=}\text{variation}(x^2 + jy^2) \quad (4)$$

In the current implementation of MATHWEBSEARCH, *where* clauses are incorporated in the original query and are evaluated in one run i.e. there is no efficiency loss in compared to simple queries.

2 The MathWebSearch Web User Interface

One of the results of the value-centered analysis in [KK07] was that the power of formula search engines like MATHWEBSEARCH will only become useful to a user, if the sacrifices involved in using it are small.

Therefore we have experimented with integrating MATHWEBSEARCH into editing environments (e.g. the `emacs` editor [Pes06]) and as a search plugin for the FireFox 2 or Internet Explorer 7 search bar. The input in this case is the term representation (called "string representation" in [KŞ06] and documented there); this includes the internal representation of terms (prefix notation) and any expression that Mathematica is able to parse. To require terms to be only interpreted as prefix notation, prepend a `#` to them (for example `#eq(id(x),@value)`).

But these approaches still require the user to learn a specific notation for MATHWEBSEARCH queries, and are therefore geared more for users who have already seen experienced the benefits of semantic search. For new users, we need a web-based interface that is simple and intuitive enough to encourage experimentation without a learning curve. In fact, we expect many of our users to arrive by chance and lack the patience to do more than a few clicks before abandoning the web page. For this clientele we have developed a novel formula editor that interfaces to MATHWEBSEARCH, which will act as the primary web interface to the system. Our formula editor is a typical linear/palette input hybrid with the difference that the internal representation is very loosely coupled to the input and output notations, which enables switching them easily according to the user's preferences. The internal representation is either OPENMATH or Content MATHML depending on the application. It would be possible to use some other XML encoding.

The input syntax is selectable so that users already familiar with some of the supported ones can type directly as they are used to and also paste some expressions in those notations. We do not intend to parse any arbitrary formula originating from those external systems (which would be the task of a complete OpenMath "phrasebook"), but rather avoid making the user stop to think how to type simple things such as $\sin 3x$ which can be input as `sin(3*x)`, `Sin[3x]`, `sin(3x)`, `Sin(3*x)` depending on the selected input notation. As a side effect, the formula editor also works as a translator: when switching to another syntax, the internal representation is translated back (unparsed) to the new linear syntax.

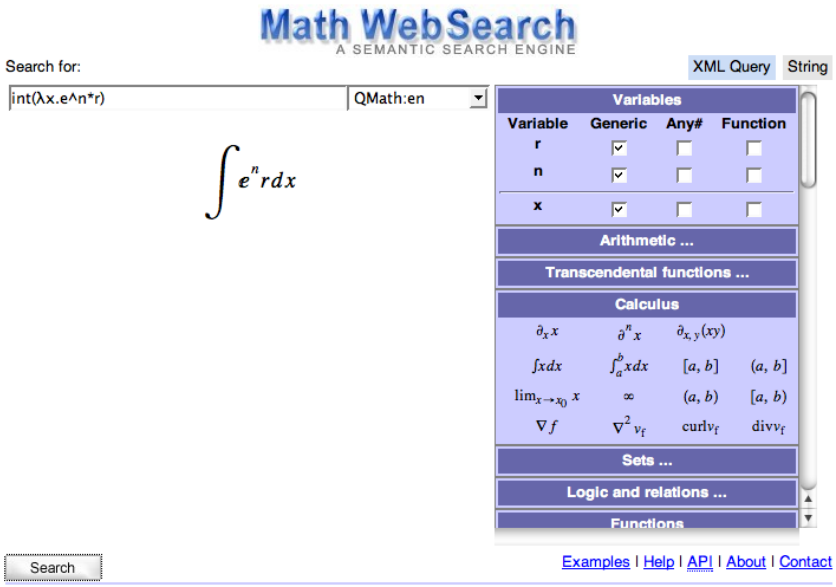

Fig. 1. Searching for an integral: the variables n and r are set to be generic by the user, and the bound variable x is generic by default. Most of the palettes are collapsed, with only the "Variables" and "Calculus" ones open and visible here.

The palettes contain the content encoding of the formulas, which is unparsed for insertion in the input field at the current cursor location. This way, using the palettes serves as documentation for the linear input syntax, enabling a smooth transition from palette-based entry to direct typing. In contrast to other palette-based editors, the placeholders in the formulas have default content so that just clicking once in a button inserts a complete formula.

Undo/redo is provided by the browser through the linear input text field. This is enough because it is translated immediately after each change to the internal representation, and from that the Presentation MathML form is generated for display.

The top palette contains the variables used in the current formula, and is updated dynamically as the formula changes. Free variables are listed above the horizontal line, and bound variables under it. Each list is ordered by the position of the last usage of each variable so that when writing a new formula, the last typed variables appear on top to keep them visible even if the variable list grows past the bottom of the page.

The first column is the variable name, and the rest are flags specifying whether it is a generic variable, whether it might match a sequence of subexpressions, and finally whether it is a function or not. Deleting a variable from the input does not reset those flags: if it is reintroduced they will be recovered.

The function flag is unrelated to MathWebSearch and needed only by the parser, as we allow constructions like $x(y+2)$ to mean the multiplication of an implicit variable x by $(y+2)$.

Single letters are always considered variables if not defined, while undefined words are treated as either products of single-letter variables or new implicit variables depending on the grammar for the current input syntax.

Syntax errors are reported by tinting the input field red, but this is unsatisfactory as finding the problem requires a familiarity with the input syntax grammars that can not be expected from most users, and we are investigating ways of giving them useful feedback, finding a balance between overly generic indications (like the current coloring) and complete but cryptic grammar descriptions.

There are however many errors that a grammar check does not uncover but can be realized by the user when seeing the final presentation display.

For instance, intending f to be a function in $f(x)$ but getting a multiplication instead: when f is interpreted as a function variable, there is a bigger separation between it and the next term than if it was an object variable being multiplied. In general, any symbols for which the input syntax and the presentation are unique and sufficiently distinct benefit from this effect (e.g. `absx` $\rightarrow |x|$) since the double translation linear $\rightarrow$ content $\rightarrow$ presentation amplifies the difference between expected and actual result.

We implemented the formula editor using only web standards: JavaScript, XHTML, XSLT, MathML and CSS. This way much code can be shared between the web interface for MathWebSearch and the FireFox extension Sentido [Pal06]. We intend to put as much as possible of the algorithms into XSLT so that they can be reused in software systems implemented in different programming languages: XSLT processors are available as libraries for many languages, but the other technologies we use are not so portable. We could then reuse the formula editor server-side to render the formulas as images for browsers that don't support MathML, and specifically the parser for indexing plain text documents.

3 The MathWebSearch Server

The MathWebSearch server is a distributed application consisting of

Search Nodes that run search servers, index builders, and web crawlers. Some of the nodes contain "meta-servers" that act as gateways to others; they forward queries to a specified set of nodes and merge the received results. Thus the collection of search nodes is organized as a tree for efficient query distribution when using a large number of nodes. In case of failure of a node, the only effect is that the results that would have been produced by that node are not received by the web server.

Database Servers that store the indexed documents here realized in MySQL.

A Web Server to Communicate with Browser Clients that combines search results from the root meta-server with the documents from the databases.

An Admin Server that allows to assign the different tasks to the available nodes, or to add automatic load balancing if needed. The admin server also monitors the search nodes for node failures and re-directs search.

A search server is able to run multi-threaded searches. In the age of multi-core CPUs, this becomes an important feature. The number of threads used in the search is chosen at the index building step. By construction, the theoretical speedup is linear. However, depending on the query, one of the threads may have to find all results and then no speedup is achieved. If the threads have balanced amount of work the achieved speedup is linear – or even super-linear due to cache effects. To avoid memory allocations and cache misses, the same area of memory – a memory pool – is being used in different steps of the search.

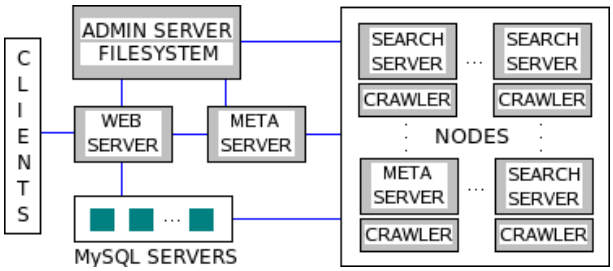

Fig. 2. System Architecture

We have tested our implementation on `http://functions.wolfram.com`, which contains 90,000 mathematical formulae and yields an index with 1.5 million subterms (memory footprint 250MB). Typical query execution times are in the range of milliseconds. The search in (1) takes about 1 ms for instance. Performed experiments indicate that index sizes have little effect on search times, even for more complex searches.

The distributed architecture and multi-threaded implementation of the search nodes is the main contribution of MATHWEBSEARCH 0.3 over the previous version presented in [KŞ06]. It leads to significant efficiency gains (reducing memory footprint to a third and speeding up searches by an order of magnitude) and makes the system scalable to larger indexes.

3.1 Input Processing

MATHWEBSEARCH can process any XML-based content representation of mathematical formulae: MATHML [ABC+03] and OPENMATH [BCC+04] are supported directly, other formats e.g. Wolfram Research's MATHEMATICA are supported if a converter for them is provided.

Consider the example on the right: We have the standard mathematical notation of an equation (1), its Content MATHML representation (2), and the term we extract for indexing (3) Note that in the internal (MATHML$^{\mathcal{Q}}$) syntax, the query variables and the generalization targets in the index are represented with identifiers starting with the `@` character. As previously stated,

(1) Mathematical expression: $f(x) = y$

(2) Content MATHML:

```
<apply><eq/>
  <apply>
    <ci>f</ci>
    <ci>x</ci>
  </apply>
  <ci>y</ci>
</apply>
```

(3) String representation: $eq(@f(@x), @y)$

Fig. 3. Converting to MATHML$^{\mathcal{Q}}$

any mathematical construct can be represented in a similar fashion. See [Mat07] for more details and syntax of search queries.

As we can see, terms like the one in the figure are not stored in their original form in the index. MATHWEBSEARCH 0.4 stores formulas with universal variables in their generalized form. In the example, we identify f, x and y as universals and store the formula in generalized representation in the index. Thus, the above formula is stored, for example, as *@1(@2)=@3*, where the @ variables can be later matched against any term at search time. The step that takes care of identifying the universal variables in the mathematical formulas is done as a pre-processing step by a **variable spotter**. This utility goes through the input MATHML formulas and annotates all the generic (universal) variables with unique identifiers, before these terms are stored in the index. The **variable spotter** does not discard bound variable information. For example, $f(x) + 2 = x * y$ will be transformed into *@10(@11) + 2 = @11 * @13*, before being turned into prefix notation. This enables search queries where these generic terms stored in the index can be instantiated or unified at search time.

An extensible framework for approximate search is also provided. For example, the user can easily perform a search for $eq(x, 5)$ but specify that numbers within range of 1 also match. $eq(x, 4)$ or $eq(x, 6)$ will then also be considered results. With the same extension we can catch typos like $fibonaci(x)$ and return $fibonacci(x)$ as well.

3.2 Indexing Mathematical Formulae

For indexing mathematical formulae on the web, we will interpret them as first-order terms and index them with a form of tree-based indexing called *substitution-tree indexing* [Gra96] As the name already suggests, the tree has substitutions in each node and satisfies the following conditions:

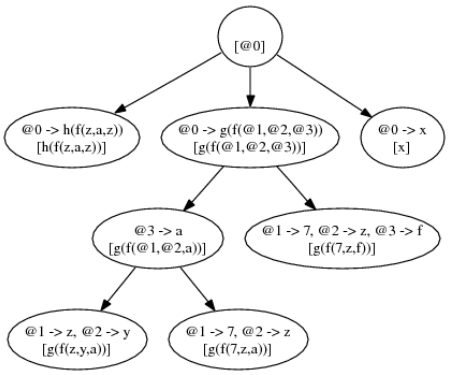

Fig. 4. An Index with Five Terms

- Each node is either leaf or has at least two sons.
- The substitution at the root of the tree is of from $\{\mathcal{Q} \rightarrow \tau\}$ where $\mathcal{Q}$ is some distinguished element of alphabet and τ is some term.
- Let σ_1, σ_2, ... σ_k represent substitutions along the path from root to a leaf, σ_1 being the substitution at the root of the tree. Then
  - successive application of substitutions from root to a leaf results in one of the indexed terms i.e. $\sigma_1 \circ \sigma_2 \circ ... \circ \sigma_k = \{\mathcal{Q} \rightarrow \sqcup\}$ where t is one of the indexed terms.
  - a substitution σ_i substitute different elements i.e. $DOM(\sigma_i) \neq DOM(\sigma_j)$ for $i \neq j$.

Figure 4 shows a typical index for the terms $h(f(z,a,z))$, x, $g(f(z,y,a))$, $g(f(7,z,a))$, and $g(f(7,z,f))$. For clarity we also present the term we get if we apply all the substitution starting from the root of the tree. The subterms `@integer` are the generic subterms; for details see [KŞ06]. To each of the indexed terms we attach an identifier that relates the term to its XPointer [GMMW03] reference and other relevant data (see below).

When building indexes for multi-threaded search, we in fact build multiple indexes, equal to the number of desired threads. This is transparent to the application using the index data-structure and it is guaranteed that the different threads will have approximately equal numbers of terms in their indexes. This is needed for having approximately equal search times the search threads and thus limit the synchronization time.

3.3 Reporting Results

For a search engine for mathematical formulae we need to augment the set of result items (usually page title, description, and page link) reported to the user for each hit. As typical pages contain multiple formulae, we need to report the exact occurrence of the hit in the page. We do this by supplying an XPointer reference where possible. Concretely, we group all occurrences into one page item that can be expanded on demand. Within this we order the groups by number of contained references.

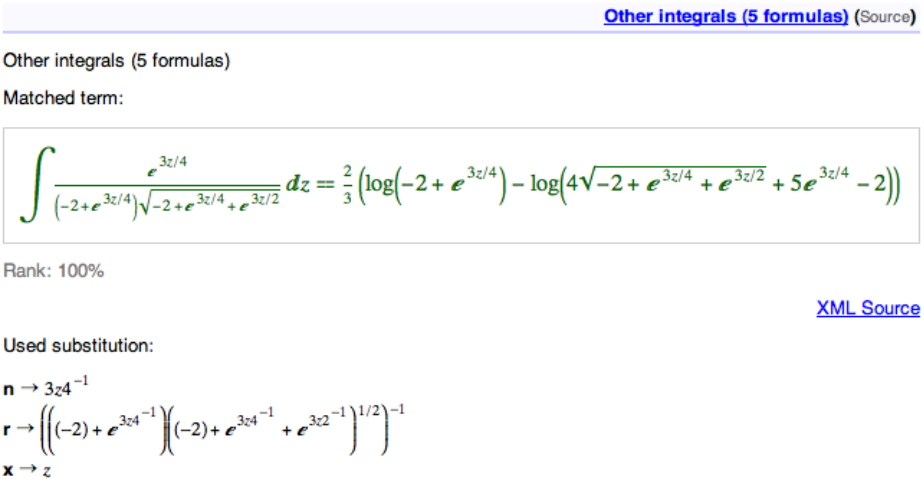

Fig. 5. Result showing the substitutions (matches) for each generic variable.

For any given result a detailed view is available (see Figure 5). This view shows the exact term that was matched and the used substitution (a mapping from the query variables specified by generic terms) to match the found result. A

more serious problem comes from the fact that — as mentioned above — content representations are often the source from which presentations are generated. If MathWebSearch can find out the correspondence between content and presentation documents, it will report both to the user. Wherever possible we present two links as results: one is the *source link*, a link to the document we actually index, and the *default link*, a link to the more esthetically pleasing presentation document.

4 Combined Search

The MathWebSearch engine is also enriched with the option of performing parallel text-search. The search capabilities described above become more powerful by allowing a query of a string and a formula at the same time, thus leading to more precise results. For our example, let us consider the use case where an engineer who has graduated a few years back from college, needs the formula for the probability density function (PDF) of two random variables $Y = X_1 + X_2$ on his new project. The formula that the engineer is looking for is actually $f(y) = \int f(y|x_1)f_1(x_1)$, using marginal probabilities. But our engineer only remembers something about needing the joint PDF of the sum and one of the variables to calculate $f(y)$. Since the engineer doesn't remember the exact formula for the joint PDF $f(y, x_1) = f(y|x_1)f_1(x_1)$ either, she would like to enter the search query $\boxed{f}(\boxed{x}, \boxed{y})$, which would match a large part of the formulae in the index and is therefore unsuited for searching[7]. With the text search functionality, the engineer adds the string queries `"random"` and `"variable"`, to help narrow down the search. Of course, there are many documents which contain the word *variable* in them, but only few that will also contain the specified formula. The returned intersection results all fit in one page, with the document entitled *"Sums of Random Variables"* listed near the top of the first page of results. We see that even if both the formula and string queries are very vague the intersection result set is narrowed down to a handful of documents, which can be easily browsed over in order for their relevance to be determined for the user.

The combined math and text search facility is realized in MathWebSearch by combining it with the Nutch system, a text-based search engine built on the open-source Lucene [The06] architecture by supplying a crawler, a link-graph database, parsers for HTML and other document formats. MathWebSearch 0.4 keeps an additional Lucene-based text index and connects formula and text search and to these components in order to obtain two different sets of results which are merged by intersection and then presented as output. A combined math+text search works as follows:

[7] In fact, in earlier versions of MathWebSearch, such a query would have been akin to a "denial of service attack" on the system, since the search engine would have computed all matches before returning (the whole index as) a result. In MathWebSearch 0.4 the search engine only computes and returns a limited number of answers per query; additional ones can be queried by giving a result offset.

- Use the math query on MathWebSearch and get a ranked set of results R^M
- Use the text query on Nutch and get a ranked set of results R^N
- Intersect the two result sets $R^M \cap R^N$ by ranking heuristic and supply the result list

The question of ranking search results for formula queries is largely unexplored territory, especially in the context of combined math + text search. We are currently exploring several approaches for the ranking of formula search results: match frequency, substitution size, familiarity of substituted constants, formula-class (prefer definition/theorem/...), formula-rank (prefer formulae that are frequently re-used/referenced). The score of query `q` for document `d` in Nutch correlates to the cosine-distance or dot-product between document and query vectors in a *Vector Space Model (VSM) of Information Retrieval*. When considering the combined results, the question of blending in the rankings of the two different sets of results to create a single consistent one gets even more complicated. The ranking heuristic needs to look at the relevance of the results by analyzing the result sets from the two individual searches with respect to size and distribution. In our simulations attempts so far, we are focusing on Gaussian-modelled distributions.

5 Conclusions and Future Work

We have presented a scalable search engine for mathematical formulae on the Internet. In contrast to other approaches, MathWebSearch uses the full content structure of formulae, and is easily extensible to other content formats. We will continue developing MathWebSearch into a production system.

A current weakness of the system is that it can only search for formulae that match the query terms up to α-equivalence or some previously defined approximation like small edit distance. Many applications would benefit from stronger equalities for instance. Our search in the running example might be used to find a useful identity for $\int_{\infty}^{0} f(x) \cdot g(x)dx$, if we know that $s(x) \cdot s(x) = s^2(x)$. MathWebSearch can be extended to a *E-Retrieval* engine (i.e. search modulo a an equational theory E or logical equivalence) without compromising efficiency by simply E-normalizing index and query terms (see [NK07] for a first implementation).

In the long run we plan to extend MathWebSearch, so that it can take more document context information into account, i.e. not just keywords from the text around the formulae but e.g. the topology of theories in the OMDoc format [Koh06]: It would be very useful, if we could restrict searches to formulae that are consistent with current (mathematical) assumptions.

Finally we would like to allow specification of content queries using more largely known formats, like LaTeX: strings like `\frac{1}{x^2}` or `1/x^2` could be processed as well. We are currently working on translating the ca. 450.000 TeX/LaTeX articles on Physics, Mathematics, and Computer Science in the Cornell ePrint archive (see `http://www.arXiv.org`) to content MathML, to make

MathWebSearch accessible for a larger group of users [SK]. We estimate that this collection contains in the order of 10^8 non-trivial formulae, making this collection a real scalability challenge for MathWebSearch. In such a setting the variable spotter for identifying universal variables becomes a nontrivial piece of the document processing infrastructure. We need to utilize (shallow?) linguistic technologies to reliably analyze phrases like "*Let f and g be functions from $\mathbb{N}$ to $\mathbb{R}$...*" that mark the identifiers f and g as universal and to retrieve the associated sortal restrictions. Note that the linguistic capabilities have of the variable spotter have to be considerable to detect the difference between "*... where c is a natural number*" and "*... where x is the number between 1 and n, such that...*" (only c universal) or to detect that universals in a negative scope are indeed existential.

References

[ABC+03] Ron Ausbrooks, Stephen Buswell, David Carlisle, Stéphane Dalmas, Stan Devitt, Angel Diaz, Max Froumentin, Roger Hunter, Patrick Ion, Michael Kohlhase, Robert Miner, Nico Poppelier, Bruce Smith, Neil Soiffer, Robert Sutor, and Stephen Watt. Mathematical Markup Language (MathML) version 2.0 (second edition). W3C recommendation, World Wide Web Consortium, 2003. Available at `http://www.w3.org/TR/MathML2`.

[BCC+04] Stephen Buswell, Olga Caprotti, David P. Carlisle, Michael C. Dewar, Marc Gaetano, and Michael Kohlhase. The Open Math standard, version 2.0. Technical report, The Open Math Society, 2004. `http://www.openmath.org/standard/om20`.

[Fre91] Free Software Foundation. Gnu general public license. Software License available at `http://www.gnu.org/copyleft/gpl.html`, 1991.

[GMMW03] Paul Grosso, Eve Maler, Jonathan Marsh, and Norman Walsh. Xpointer framework. W3C recommendation, World Wide Web Consortium W3C, 25 March 2003.

[Gra96] Peter Graf. *Term Indexing*. Number 1053 in LNCS. Springer Verlag, 1996.

[ICW06] Tetsuo Ida, Jacques Calmet, and Dongming Wang, editors. *Proceedings of Artificial Intelligence and Symbolic Computation, AISC'2006*, number 4120 in LNAI. Springer Verlag, 2006.

[KK07] Andrea Kohlhase and Michael Kohlhase. *Re*examining the MKM Value Proposition: From Math Web Search to Math Web *Re*Search. In Kauers et al. [KKMW07], pages 266–279.

[KKMW07] Manuel Kauers, Manfred Kerber, Robert Miner, and Wolfgang Windsteiger, editors. *MKM/Calculemus 2007*, number 4573 in LNAI. Springer Verlag, 2007.

[Koh06] Michael Kohlhase. OMDoc – *An open markup format for mathematical documents [Version 1.2]*. Number 4180 in LNAI. Springer Verlag, 2006.

[KŞ06] Michael Kohlhase and Ioan Şucan. A search engine for mathematical formulae. In Ida et al. [ICW06], pages 241–253.

[LM06] Paul Libbrecht and Erica Melis. Methods for Access and Retrieval of Mathematical Content in ActiveMath. In N. Takayama and A. Iglesias, editors, *Proceedings of ICMS-2006*, number 4151 in LNAI. Springer Verlag, 2006. `http://www.activemath.org/publications/Libbrecht-Melis-Access-and-Retrieval-ActiveMath-ICMS-2006.pdf`.

[Mat07] Math Web Search. Web page at `http://kwarc.info/projects/mws/`, seen June 2007.

[MM06] Rajesh Munavalli and Robert Miner. Mathfind: a math-aware search engine. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 735–735, New York, NY, USA, 2006. ACM Press.

[NK07] Immanuel Normann and Michael Kohlhase. Extended formula normalization for ϵ-retrieval and sharing of mathematical knowledge. In Kauers et al. [KKMW07], pages 266–279.

[Pal06] Alberto González Palomo. Sentido: an authoring environment for OMDoc. In OMDoc – *An open markup format for mathematical documents [Version 1.2]* [Koh06], chapter 26.3.

[Pes06] Darko Pesikan. Coping with content representations of mathematics in editor environments: nOMDoc mode. Master's thesis, Computer Science, Jacobs University, Bremen, 2006.

[SK] Heinrich Stamerjohanns and Michael Kohlhase. Transforming the arχiv to xml. submitted to MKM 208.

[The06] The Apache Software Foundation. Lucene. `http://lucene.apache.org/`, 2000–2006.

[You06] Abdou Youssef. Roles of math search in mathematics. In Jon Borwein and William M. Farmer, editors, *Mathematical Knowledge Management, MKM'06*, number 4108 in LNAI, pages 2–16. Springer Verlag, 2006.