MathWebSearch 0.5: Scaling an Open Formula Search Engine

Michael Kohlhase and Corneliu C. Prodescu

Computer Science, Jacobs University Bremen
http://kwarc.info

Abstract. MathWebSearch is an open-source, open-format, content-oriented search engine for mathematical formulae. MathWebSearch is a complete system capable of crawling, indexing, and querying expressions based on their functional structure (operator tree) rather than their presentation.[1] EdNote:1
In version 0.5, we concentrate on scalability issues in MathWebSearch to take advantage of corpora in the giga-formula range. We re-implemented the index to make it distributable and made all the APIs web standards conformant. 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. Here we tackle the problem of finding information that is given in the form of (mathematical) formulae. Standard search engines like Google cannot deal with formulae at all, severely limiting the reach and utilization of technical, scientific, and engineering documents.

In this paper we present new work in the context of the MathWebSearch system; a search engine that addresses the problem of searching mathematical formulae from a semantic point of view, it finds formulae by their structure and meaning not via their presentation.

In [KŞ06] we have presented the motivation, query language, and web front end of MathWebSearch 0.1. In [KK07] we have re-examined the value proposition of semantic search for mathematical knowledge homing in on the benefits and sacrifices induced by the various search approaches [You06b; MM06; LM06], from a user's perspective. The result of this analysis is MathWebSearch 0.5, which we describe in this paper. The new version features significant efficiency gains (space efficiency increased by a factor of five), new management features, advanced searching capabilities, and a new user interface. The MathWebSearch system (see [MWS] for details) is released under the Gnu General Public License.

The motivation for the work reported in this paper is the availability of large[1] corpora, such as the arXMLiv corpus [SK08] with almost three quar-

[1] EdNote: Review2: MathWebSearch as start for the first two sentences
[1] We deem a corpus as *large* if it has more than 20 million expressions

ters of a million scientific articles and an estimated giga-formula. This has not only re-kindled interest in formula search[2], but also severely taxes the scalability of systems. Scalability issues for presentation-based search engines have been addressed in [SL11]. Such engines map formulae to "special words" which can then be indexed by conventional bag-of-word information retrieval engines, which have become extremely scalable over the last years. The case for MathWebSearch is completely different, since the content-based unification queries it offers require an index data structure that reflects the inner structure of formulae (rather than just pointers to words). Even with the space efficiency gains in MathWebSearch 0.5, the indices will surpass the main memory of most machines. Therefore, we have laid the foundations for distributing the MathWebSearch in this version.

Before we present MathWebSearch 0.5 from a technical perspective in Section 3, we will recap unification-based querying. We evaluate the system on a large corpus in Section 4 and see that we need to distribute MathWebSearch to cope with linear RAM usage. Section 5 presents the necessary extensions of the indexing. Section 6 concludes the paper and discusses future work.

2 Querying Mathematics by Unification

Retrieval of mathematical knowledge and information via unification-based queries for content-encoded mathematical formulae is very natural. In [KŞ06] we have already discussed **instantiation queries**, which can be used to retrieve partially remembered formulae, e.g. the query for the formula for energy of a given signal $s(t)$ in Figure 1. Note that instantiation queries are more expressive as a 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.

Query (query variables marked as named boxes)	Result (Parseval's Theorem)
$\int_{\boxed{min}}^{\boxed{max}} \boxed{f}(x)^2 dx$	$\frac{1}{T}\int_0^T s^2(t)dt = \sum_{k=-\infty}^{\infty} \Vert c_k \Vert^2$

Fig. 1. An Instantiation Query

To see the full power of unification-based querying consider a student who encounters $\int_{\mathbb{R}^2} \vert\sin(t)\cos(t)\vert dt$ 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) and they can be found using **generalization queries**. The idea behind answering generalization queries is that the index marks universal[3] variables in subterms

[2] The next NTCIR-10 Challenge in spring 2013 will have a "math track". NTCIR evaluates information access technologies in a series of competition events in Japan

[3] We consider an identifier as universal if it can be instantiated without changing the truth value of the containing expression. In formal representations like first-order

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) in Figure 2, 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 $\boxed{x} \mapsto 2t$.

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

Fig. 2. A Formula with Universal Variables in the Index

A very similar idea is used in **variation queries** where the indexed terms are searched to match the search 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 $g^2\cos(\sqrt{y}) + b\sin(\sqrt{y})$.

3 The MathWebSearch System, Version 0.5

The MathWebSearch system consists of the three main components pictured in Figure 3. The *crawler subsystem* collects data from the corpora[4]. It transforms the mathematical formulae in the corpus into *MWS Harvest*s (XML files that contain formula-URIreference pairs) and feeds them into the core system. The *core system* (the MathWebSearch daemon mwsd) builds the search index and processes search queries: it accepts the MathWebSearch input formats (*MWS Harvest* and *MWS Query*; see [**KohPro:MWSmanual**]) and generates the MathWebSearch output format (*MWS Answer Set*). These are communicated through the *RESTful interface* restd which provides a public HTTP API conforming to the REST paradigm.

logic, such variable occurrences can be effectively computed, but in semi-formal settings like mathematical textbooks, they have to be approximated by heuristic methods; see the discussion in the conclusion for details.

[4] Note that we envision essentially one crawler per corpus. The crawlers are specialized to the respective formula representation, the organization and access methods to the corpus, etc. We have only implemented a crawler for the arXiv (see Section 4), but additional crawlers can be patterned after this (see Section 6.1).

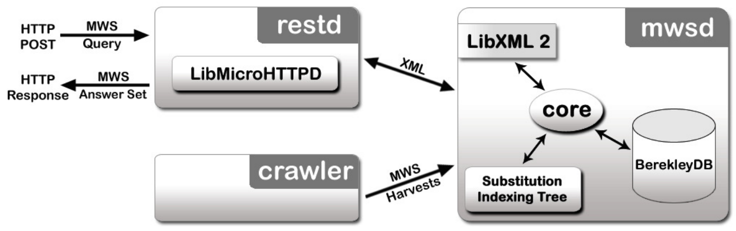


Fig. 3. MWS-0.5 System Structure

These components have been implemented using POSIX-compliant [Pos] C^{++}. We use the MicroHTTPd library [Mic] API for handling HTTP, and LibXML2 [Vei] API for XML parsing. The meta-data accompanying the internal index is stored using an external database system. As we are dealing mainly with key-value retrieval, the BerkeleyDB [Ber] API was preferred.

The system supports two main workflows:

1. The crawler sends an *MWS Harvest* to mwsd. The XML is parsed and an internal representation is generated. This is used to update the Substitution Indexing Tree and consequently the database.
2. The user sends an *MWS Query* to mwsd. The XML is parsed, an internal query is generated. Using an efficient traversal of the index tree, formulas matching the search term are retrieved and aggregated into a result. This is translated to an *MWS Answer Set* and sent back to the user.

3.1 Substitution Tree Indexing in MathWebSearch

As we are interested in indexing mathematical formulae at a large scale (document archives, text corpora), repetitive content is expected. After all, theorems are built on top of other theorems and terms on top of subterms. With this in mind, we chose a space-efficient internal representation based on substitutions.

In the previous version of MathWebSearch, we used a technique borrowed from Automated Theorem Proving called Substitution Indexing [Gra96]. It involves indexing expressions in a tree based on generality. The root is a generic variable and, as we go from a node to one of its children, one or more substitutions occur. For this version, we kept the substitution tree model and performed a few changes to better fit our design goals.

As such, we improved query times, by imposing a fixed substitution ordering. Hence, the query term describes a deterministic path through the index. The chosen ordering instantiates the left-most variable first, equivalent to DFS traversal of the operator tree. An example index, containing the terms $f(a)$, $f(b)$ and b, is presented in Figure 4. Note that the edges in the tree are labeled with the operators

$$\begin{tikzcd}
 & @0 \arrow[dl, "@1(@2)"'] \arrow[drr, "b"] & & \\
@1(@2) \arrow[d, "f"'] & & & b \\
f(@2) \arrow[d, "a"'] \arrow[dr, "b"] & & & \\
f(a) & f(b) & &
\end{tikzcd}$$

Fig. 4. Example DFS Substitution Tree

and operations: @1(@2) stands for the application operation.

Additionally, we save space by performing two steps of pre-processing for inserted terms, as well as query terms.

Firstly, we detect and reduce identical subterms. This is done by breaking the term into all possible subterms and detect equivalent subterms. Following this, the term is rewritten using only unique subterms (repeated matches are replaced through references to the first match in DFS order). For example the term $f(a, g(b), g(b))$ can be rewritten to $f(a, g(b), @[4])$, where @[4] represents the 4th term in DFS order: the subterms in DFS order are: $f(a, g(b), @[4])$, f, a, $g(b)$, g, b.

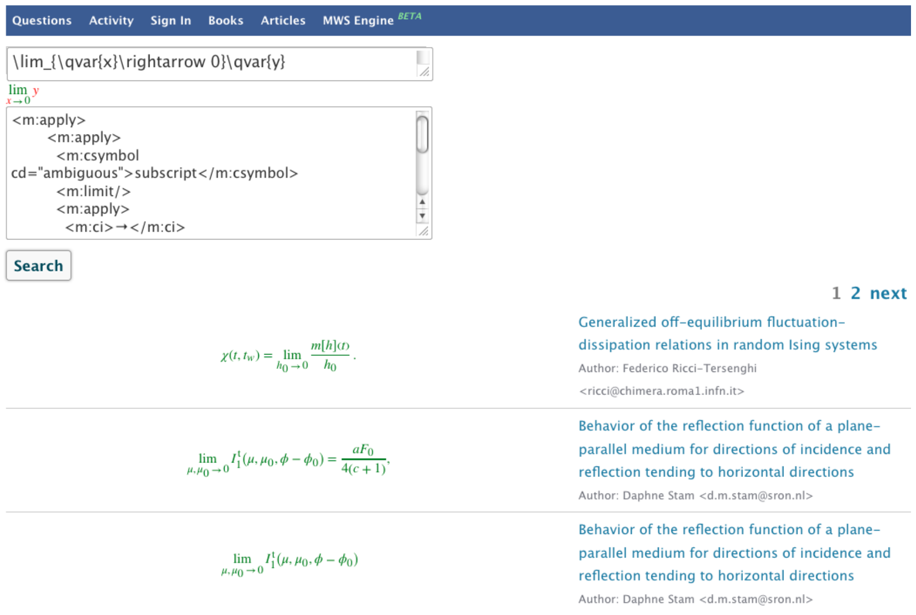
Furthermore, query terms with repeating identical query variables are reduced and handled as query terms with no repeating query variables. More importantly, this makes the search process stateless, as no previously matched query variable instantiations need to be stored.

Secondly, we hold an internal dictionary which maps symbols (in Content-MathML, represented by element name, attributes and text content) to integer IDs. The encoding relies on the fact that there are relatively few distinct tokens (compared to the number of expressions, for example). This achieves significant memory savings at a small price, since each (inserted or query) term is encoded exactly once.

3.2 Search Frontends and Embeddings

For practical applications, mwsd serves as a search back-end that needs to be embedded into a front-end system, which hides some of the complexities of writing MathWebSearch queries from the user. One example of a front-end system we are experimenting with is given in Figure 5[5] Here the user can enter queries in the LaTeX extended with the `\qvar` macro for query variables. This is then transformed into the content-MathML-based MathWebSearch queries by the LaTeXML daemon [GSK11] (the formula is also presented to the user with the query variables colored red). Upon receiving the resulting URIs, the frontend assembles a list of formulae and paper titles which link to the original paper. In this situation we are making use of the fact that TeX/LaTeX is a lingua franca for technical communication in the Mathematics community. For other communities, leveraging the MS Office equation editors might be an attractive option. For active document settings (e.g. in semantic publishing systems like Planetary [Koh+11]), formulae might be instrumented with a "search similar formulae" interaction. The same holds for integrated semantic development system such as Mathematica.

[5] The demo is temporarily available at `http://arxivdemo.mathweb.org/index.php?p=/article/MWS`; we will provide a more permanent location for the final version of the paper.



```
\lim_{\qvar{x}\rightarrow 0}\qvar{y}
```

$\lim_{x \to 0} y$

```
<m:apply>
      <m:apply>
        <m:csymbol
cd="ambiguous">subscript</m:csymbol>
        <m:limit/>
        <m:apply>
          <m:ci>→</m:ci>
```

$\chi(t, t_w) = \lim_{h_0 \to 0} \frac{m[h](t)}{h_0}$.

Generalized off-equilibrium fluctuation-dissipation relations in random Ising systems
Author: Federico Ricci-Tersenghi <ricci@chimera.roma1.infn.it>

$\lim_{\mu, \mu_0 \to 0} I_1^t(\mu, \mu_0, \phi - \phi_0) = \frac{aF_0}{4(c+1)}$,

Behavior of the reflection function of a plane-parallel medium for directions of incidence and reflection tending to horizontal directions
Author: Daphne Stam <d.m.stam@sron.nl>

$\lim_{\mu, \mu_0 \to 0} I_1^t(\mu, \mu_0, \phi - \phi_0)$

Behavior of the reflection function of a plane-parallel medium for directions of incidence and reflection tending to horizontal directions
Author: Daphne Stam <d.m.stam@sron.nl>

Fig. 5. MWS-0.5 arXivDemo Search Interface

4 Evaluation

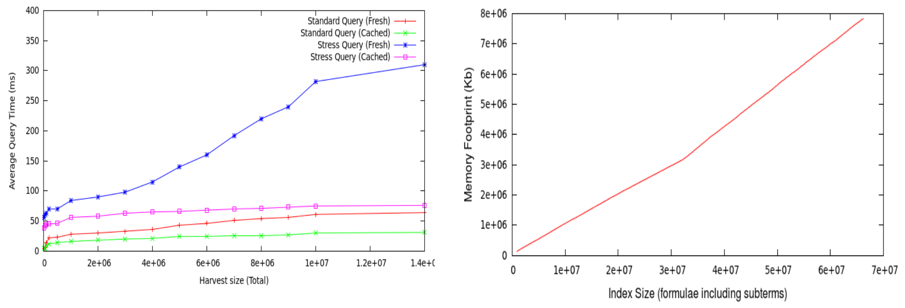
We evaluate the MathWebSearch implementation on a large corpus of mathematical formulae:

The arXMLiv Corpus Our group is working on the translation of the almost 750.000 TEX/LATEX articles on Physics, Mathematics, and Computer Science in the Cornell ePrint archive (see `http://www.arXiv.org`) to MathML [SK08]. **The arXMLiv corpus** is the result of translating ∼72% of the arXiv papers. For our evaluation we have harvested ca 65% (the fragment that have been converted without errors), resulting in a total of 115 million expressions. A trivial estimation suggests that the full arXMLiv corpus would contain approx. 245 million formulae. To harvest these, the arXMLiv crawler goes recursively through the pages of [arXMLiv] extracting the content MathML[6] elements, combines them with URI references, and reports them to MathWebSearch. We will now report on a performance analysis for MathWebSearch parametrized on **harvest size** (see Figure 6). As our index also indexes subformulae, we include them in the harvest size. Note that in the arXMLiv corpus a formula has 5.6 proper subformulae on average[7] so we estimate the number of indexable formula

[6] The result of the transformation contains both Content and Presentation MathML representations in parallel markup.

[7] This rather low number comes from the fact that roughly 2/3 of the formulae in the arXMLiv corpus consist of only one letter; these are largely irrelevant for search purposes.

occurrences in the arXiv corpus to be $6.6 \times 245 \times 10^8$ or 1.6 billion. Note that many of these formulae will actually be identical, leading to space savings in the index: recall that the URIs of the subformula occurrences are stored in a data base indexed by the leaves of the index (see Figure 3). Thus the index grows with the formula, whereas the database grows with the formula occurrences.



(a) Query Times (b) Memory Usage

Fig. 6. Experimental Performance Analysis

[2] EdNote:2

Average query times We'll start with the time efficiency aspect, as this is highly relevant for a search system. The average query times [8] are presented in Figure 6(a). This experiment involves measuring several query response times. A standard query has answsize = 30 and limitmin < 9000. Response times are measured for standard queries (fresh and cached) from 0 up to 5 qvars. Additionally, stress queries with answsize = 10000 are used.

As one can see, the query response times are fairly constant as the harvest data increases. This fits with the theory, as the querying process will follow the same paths in the index (because the query data remains constant). The small increase is due to the slightly higher density of the index tree (which affects retrieval of the right path).

The gap between the fresh and cached stress queries is expected, due to the fact that the current bottleneck is retrieving the meta-data from the external database. Hence, caching significantly improves the results.

In comparison to MathWebSearch 0.1 [KŞ06], which reported query times below 50 ms for simple queries and 200 ms for stress queries on a harvest size of 1.6 million, we see that the query times have not increased (even after normalization for hardware effects).

Memory usage The graph in Figure 6(b) presents the memory footprint of the mwsd process, as the system indexed 11.5 million expressions (67 million includ-

[2] EdNote: Review2:Figures 5, 6a, 6b: too small

[8] Note that the queries were sent from the local network, to eliminate any channel delays

ing subterms) harvested from the arXMLiv corpus. In comparison to MathWebSearch 0.1, which reported a memory footprint of 770 MB for a harvest size of 1.6 million, we see a space efficiency improvement by a factor of five. Contrary to our expectations of logarithmic increase, we see a fairly linear graph; the fact that the gradient became steeper after ∼33 million expressions is particularly unexpected. We are still investigating this.

Nevertheless, the experimental results are valuable to estimate the memory necessary to index the entire arXiv corpus. Assuming linear scaling across the 245 million formulae estimated for the arXiv, the memory necessary to index all the formulae would be $245 \times 8/11.5 \approx 170Gb$ according to our experiment. As this transcends the RAM of most machines, we have extended mwsd so that it can be distributed: a reasonable size computer cluster could easily accommodate the entire arXMLiv corpus and thus provide content-based formula search for `arXiv.org`.

5 Distributing MathWebSearch

We are currently implementing a distributed version of MathWebSearch. The core components like the RESTful interface, the data formats, and the indexing data structures remain the main building blocks, but we implement data persistency, distribution and migration on top of these. We will now present the extended index structure and data migration.

5.1 A Distributable Substitution Tree

As presented in Section 3, the main indexing data structure is a DFS substitution tree. To represent the tree in a manner which supports cross-machine links, we use three types of index nodes:

Internal Index Nodes They are used to navigate through most parts of the tree. Their data stores mappings from token ID to the corresponding index node.

Leaf Index Nodes They represent the end of a particular formula and its corresponding ID in the URL+URI database.

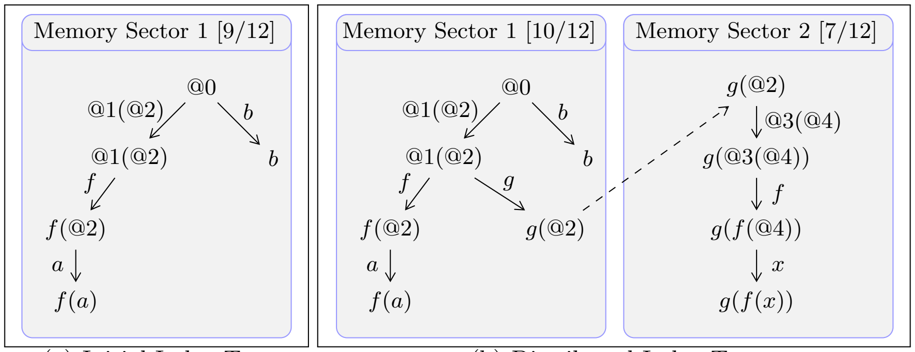
Remote Index Nodes They represent cross-sector links. Their data consists of a pair of memory sector ID and node ID, which uniquely determine the corresponding index node.

As harvests are fed into the system, the index is built. Notice that, instead of building it on the heap (with no control over individual node's memory locations), the system places it inside specific memory sectors.

Memory sectors are continuous areas of memory of fixed sizes[9], usually represented as memory mapped files. They are the smallest units of replication,

[9] The exact size depends on the RAM sizes of the nodes where the system will run, typical values ranging between 128Mb and 2048Mb.

migration, and load balancing. When the system starts, an initial memory sector is created and the tree is built in this area. As terms get inserted, the sector fills. When it reaches a given threshold, a new sector is created and the contents of the original sector get split between the two.



(a) Initial Index Tree (b) Distributed Index Tree

Fig. 7. Tree Distribution across Memory Sectors

In Figure 7, we present an example of term insertion which causes a tree split. Consider the initial tree containing the expressions $f(a)$ and b and Memory Sectors of size 12 units, as depicted in Figure 7(a). We consider a memory model where each link and each node costs 1 unit. Let's insert $g(f(x))$. As the resulting tree would overflow the current memory sector, a new one is created and parts of the tree are migrated to the new memory sector (See Figure 7(b)). To split the tree, the system performs a DFS traversal, as long as the size needed to store the explored and queued nodes does not exceed a given threshold (in our case, 10). Once a node expansion would go over it, all the internal nodes in the queue are transformed into remote nodes and the rest is moved to the other memory sector.

5.2 Distribution of Indexing and Querying

The advantage of the new data structures is that indexes can be distributed over multiple computers: when the memory usage on one machine goes above a specific threshold, a percentage of the memory sectors are migrated to other machines. As the tree uses only relative pointers and the sectors are represented as memory mapped files, it is enough to flush the memory sector to the file, send it across the network, and re-map it into the new system's memory (of course, we assume endian-compatible machines). All operations on memory sectors (splits, migrations) are coordinated by a master node, which keeps track of their locations, as well as cross-sector links.

EdNote:3

[3] This process will result in a memory organization as pictured in Figure 8 on the right. Note that this scheme tries to keep the "spine" of the index on the master node. As queries will normally only request an initial segment of results, this will minimize cross-sector and cross-machine query continuations.

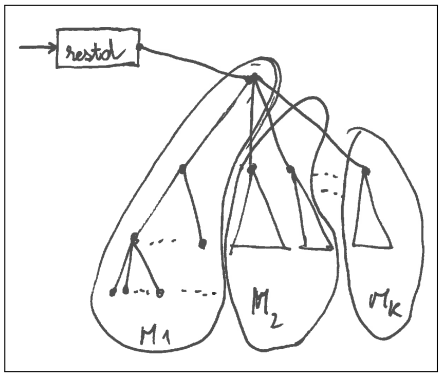


Fig. 8. Structure of a Distributed Index

For a distributed instance of MathWebSearch we have the following setup: All nodes in the network of MathWebSearch machines run mwsd (with the index and the URI database). MathWebSearch has only one restd, which resides on the master node. Upon receiving $\mathcal{Q}$ restd passes it to mwsd on the master node, which start DFS search on its substitution tree index fragment. Whenever mwsd hits a remote index node whose target sector is on a different machine, it forwards the respective subquery $\mathcal{Q}'$ to the mwsd on the remote machine, while continuing processing on the current machine.

In result-unlimited queries, the master mwsd will just wait for the slave mwsd to return the results and aggregate that with its own result set. In result-limited cases, some of the slave's results may be irrelevant, since the result limit has already been reached. The tradeoffs and efficiency issues involved in such effects will still need to be investigated. Similarly, the top of the index tree (which resides on the master node) will receive much higher processing loads, possibly becoming a bottleneck to the overall system; we will investigate strategies for replication of that.

6 Conclusion and Future Work

We have presented a scalable extension of a search engine for mathematical formulae. In contrast to other approaches, MathWebSearch uses the full content structure of formulae and is easily extensible to other content-oriented formats. Our first evaluation shows that query times are low and essentially constant in index/harvest size, so that a search engine can scale up to web proportions. Contrary to our expectations, index size is linear in harvest size for the arXiv corpus, which transcends the main memory limits of standard servers. Therefore, developing parallelization/distribution strategies is a priority. This paper reports the the establishment of the core distribution algorithms and functionality; exploring the distribution, management, and load balancing is beyond the scope of this paper. We conclude the paper with a tabulation of open research areas for information retrieval in mathematical/technical documents.

[3] EdNote: New figure

6.1 Additional Corpora

The arXiv corpus we are currently using for benchmarking is paradigmatic for the "informal but rigorous mathematics" that dominates mathematical communication today. Here, the Content MathML has to be heuristically reconstructed from the presentation in the sources. This is unnecessary for corpora of formalized mathematics, e.g. the Mizar Mathematical Library [Miz] with over 50 000 formal theorems. The problems of obtaining the content MathML are different here: Even though the representations are formal in principle, the surface languages are human-oriented, and fully explicit representations need reconstruction processes (e.g. for reconstructing elided types and arguments, resolution of operator overloading, etc.). We are currently working on Content MathML exporters for the Mizar Library and the TPTP (Thousands of Problems for Theorem Proving) library [SS]. Other future targets could be the input files of mathematical software systems e.g. computer algebra systems like Mathematica, numerical computation systems like MatLab, or statistics programs like the R system [Tee11].

6.2 Extending the Indexing

A current weakness of the system is the fact that it can only search for formulae that match the query terms up to α-equivalence. Many applications would for instance benefit from stronger equalities. Our search in the running example might be used to find a useful identity for $\int_0^\infty f(x) \cdot g(x)dx$, if we know that $s(x) \cdot s(x) = s^2(x)$. MathWebSearch can be extended to an *E-Retrieval* engine (i.e., search modulo 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: It would be very useful, if we could restrict searches to formulae that are consistent with current (mathematical) assumptions.

6.3 Result Ranking

Advances in ranking have made word-based search engines scalable from a user point of view. For formula search engines ranking is an open research question, there is only one paper that covers this in a more presentation-search oriented setting [You06a]. To solve the problem, we have to consider the following aspects:

- What is a good measure for relevance in theory (pagerank only applies to pages)?
- How can be compute this efficiently.
- Can we organize the index, so that it finds the most relevant hits (as estimated by this measure) first?
- Is a single measure enough?

We plan to look at the following simple measures as starting points:
- pagerank by citations over the papers
- the size (whatever that means) of the substitutions (small might be beautiful)
- similarly, the size of the formulae
- popularity of the papers (by download)

Finally, we would like to allow specification of content queries using more widely known formats, like LaTeX: strings like `\frac{1}{x^2}` or `1/x^2` could be processed as well. This can be reached by applying an extension (by query variables) of the LaTeX to XML transformation used on the arXiv to process queries. The new LaTeXML daemon [GSK11] allows integrating this efficiently.

6.4 Advanced Search Services

Another important application of the unification search in MathWebSearch is applicable theorem search (like our example with Hölder's inequality in Section 2). The MathWebSearch system already supports the necessary queries (unification), but the arXiv corpus we are currently using does not have the necessary degree of formalization (explicitly marked up universal quantifications). We plan to utilize (possibly shallow) linguistic technologies to reliably analyze phrases like "*Let f and g be functions from $\mathbb{R}$ to $\mathbb{R}$...*" that mark the identifiers f and g as universal and to retrieve the associated sortal restrictions. Note that the linguistic capabilities 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 is c universal) or to detect that universals in a negative scope are indeed existential.

References

[arXMLiv] *arXMLiv Build System.* URL: `http://arxmliv.kwarc.info` (visited on 05/15/2010).

[Ber] *Berkeley DB.* URL: `http://www.oracle.com/technology/products/berkeley-db/index.html` (visited on 03/03/2010).

[BF06] Jon Borwein and William M. Farmer, eds. *Mathematical Knowledge Management (MKM).* LNAI 4108. Springer Verlag, 2006.

[Dav+11] James Davenport et al., eds. *Intelligent Computer Mathematics.* LNAI 6824. Springer Verlag, 2011. ISBN: 978-3-642-22672-4.

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

[GSK11] Deyan Ginev, Heinrich Stamerjohanns, and Michael Kohlhase. "The LaTeXML Daemon: Editable Math on the Collaborative Web". In: *Intelligent Computer Mathematics.* Ed. by James Davenport et al. LNAI 6824. Springer Verlag, 2011, pp. 292–294. ISBN: 978-3-642-22672-4. URL: `https://svn.kwarc.info/repos/arXMLiv/doc/cicm-systems11/paper.pdf`.

[Kau+07] Manuel Kauers et al., eds. *MKM/Calculemus.* LNAI 4573. Springer Verlag, 2007. ISBN: 978-3-540-73083-5.

[KK07] Andrea Kohlhase and Michael Kohlhase. “*Re*examining the MKM Value Proposition: From Math Web Search to Math Web *Re*Search”. In: *Towards Mechanized Mathematical Assistants. MKM/Calculemus*. Ed. by Manuel Kauers et al. LNAI 4573. Springer Verlag, 2007, pp. 266–279. ISBN: 978-3-540-73083-5. URL: `http://mathweb.org/projects/mws/pubs/mkm07.pdf`.

[Koh+11] Michael Kohlhase et al. “The Planetary System: Web 3.0 & Active Documents for STEM”. In: *Procedia Computer Science* 4 (2011): *Special issue: Proceedings of the International Conference on Computational Science (ICCS)*. Ed. by Mitsuhisa Sato et al. Finalist at the Executable Papers Challenge, pp. 598–607. DOI: `10.1016/j.procs.2011.04.063`. URL: `https://svn.mathweb.org/repos/planetary/doc/epc11/paper.pdf`.

[KŞ06] Michael Kohlhase and Ioan Şucan. “A Search Engine for Mathematical Formulae”. In: *Proceedings of Artificial Intelligence and Symbolic Computation, AISC’2006*. Ed. by Tetsuo Ida, Jacques Calmet, and Dongming Wang. LNAI 4120. Springer Verlag, 2006, pp. 241–253. URL: `http://kwarc.info/kohlhase/papers/aisc06.pdf`.

[LM06] Paul Libbrecht and Erica Melis. “Methods for Access and Retrieval of Mathematical Content in ActiveMath”. In: *Proceedings of ICMS-2006*. Ed. by N. Takayama and A. Iglesias. LNAI 4151. Springer Verlag, 2006, pp. 331–342. URL: `http://www.activemath.org/publications/Libbrecht-Melis-Access-and-Retrieval-ActiveMath-ICMS-2006.pdf`.

[Mic] *GNU MicroHTTPd Library*. seen Jul 2011. URL: `http://www.gnu.org/software/libmicrohttpd/` (visited on 07/11/2011).

[Miz] *Mizar Mathematical Library*. URL: `http://www.mizar.org/library` (visited on 12/02/2009).

[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*. Seattle, Washington, USA: ACM Press, 2006, pp. 735–735. ISBN: 1-59593-369-7. DOI: `http://doi.acm.org/10.1145/1148170.1148348`.

[MWS] *Math Web Search*. `https://trac.mathweb.org/MWS/`. seen Jan. 2011. URL: `https://trac.mathweb.org/MWS/`.

[NK07] Immanuel Normann and Michael Kohlhase. “Extended Formula Normalization for ϵ-Retrieval and Sharing of Mathematical Knowledge”. In: *Towards Mechanized Mathematical Assistants. MKM/-Calculemus*. Ed. by Manuel Kauers et al. LNAI 4573. Springer Verlag, 2007, pp. 266–279. ISBN: 978-3-540-73083-5.

[Pos] *IEEE POSIX*. ISO/IEC 9945. 1988.

[SK08] Heinrich Stamerjohanns and Michael Kohlhase. “Transforming the arXiv to XML”. In: *Intelligent Computer Mathematics*. 9th International Conference, AISC, 15th Symposium, Calculemus, 7th International Conference MKM (Birmingham, UK, July 28–Aug. 1, 2008).

Ed. by Serge Autexier et al. LNAI 5144. Springer Verlag, 2008, pp. 574–582. URL: http://kwarc.info/kohlhase/papers/mkm08-arXMLiv.pdf.

[SL11] Petr Sojka and Martin Líška. “Indexing and Searching Mathematics in Digital Libraries – Architecture, Design and Scalability Issues.” In: *Intelligent Computer Mathematics*. Ed. by James Davenport et al. LNAI 6824. Springer Verlag, 2011, pp. 228–243. ISBN: 978-3-642-22672-4.

[SS] Geoff Sutcliffe and Christian Sutner. *The TPTP Problem Library for Automated Theorem Proving*. URL: http://www.tptp.org (visited on 12/20/2011).

[Tee11] Paul Teetor. *R Cookbook*. second. ISBN: 978-3486705171. O’Reilly, 2011. URL: http://oreilly.com/catalog/9780596809157.

[Vei] Daniel Veillard. *The XML C parser and toolkit of Gnome; libxml*. URL: http://xmlsoft.org (visited on 07/11/2011).

[You06a] Abdou Youssef. “Methods of Relevance Ranking and Hit-content Generation in Math Search”. In: *Mathematical Knowledge Management (MKM)*. Ed. by Jon Borwein and William M. Farmer. LNAI 4108. Springer Verlag, 2006, pp. 393–406.

[You06b] Abdou Youssef. “Roles of Math Search in Mathematics”. In: *Mathematical Knowledge Management (MKM)*. Ed. by Jon Borwein and William M. Farmer. LNAI 4108. Springer Verlag, 2006, pp. 2–16.