Guided Research: Final Report

MaTeSearch

A combined math and text search engine

Ştefan Anca
<s.anca@jacobs-university.de>
Prof. Michael Kohlhase
<m.kohlhase@jacobs-university.de>
Jacobs University

Spring Semester 2007

Abstract

This project presents a new type of search engine, which can handle a mathematical query and a text query at the same time and output documents which contain both items. As the name suggests, MaTeSearch uses math and text together in order to search for scientifical content on the internet. MaTeSearch is built on top of two already existing search engines, MathWebSearch and Nutch. The first is a semantic search engine that uses an XML query language to search for mathematical terms and the second is a text-based search engine built on the open-source Lucene architecture. MaTeSearch connects to these components in order to obtain two different sets of results which are merged by intersection and then presented as output. A demo version of the MaTeSearch system is already running online, having indexed the Connexions repository in both text and math. This project also provides an in-depth analysis of the ranking strategy that should be used on the results of such combined search engines.

Contents

1 Introduction

The modern world society is becoming more and more information-centered as its sources of data increase at a steady, if not exponential rate. Academic activities are becoming exceedingly dependent on existing public information resources in the same way as quotidian life relies more and more on Google searches for local event information or Wikipedia topic queries. Being able to search for semantic mathematical information online is now no longer impossible since the appearance of content-based search engines like MathWebSearch. This project analyzes ways to enrich this existing math-search engine by adding parallel text-search capabilities alongside it. Through MaTeSearch, existing search capabilities become more powerful by allowing a query of a string and a formula at the same time, thus leading to more precise results. The current project looks at a working implementation of the simple intersection of the results from the two composite search engines and discusses heuristics to properly weigh the results.

2 State of the art in Search for Mathematics

Searching for regular text is a service that has been widely available to computer users in many forms. From searching for file names on a local computer to searching for articles, forums, or any webpages that contain a certain phrase on the internet, text-based search holds no secret for the basic computer user. Indeed, many of the internet-related activities that are available to us at the moment are based on text-search. For mathematical purposes, however, text search is limited to finding names of authors, articles or books and theory keywords (such as "Fermat's little theorem"). Most of the engines that we come across in a search for mathematical search engines on the internet are text-based. Search engines like these have math search capabilities solely because they index huge libraries of mathematical texts. Examples would be the search engines at `arXiv.org` from Cornell University [arX07], `MathSearch` at University of Sydney [Mat99] and the `Zentralblatt-Math`

at FIZ Karlsruhe [Zen07].

These engines provide search capabilities which are enough to guide us if we know the universally-agreed name of the theory that we are looking for. However, they are not useful if we are looking, for example, for articles that contain a certain formula, in the case we want to find out whether someone else has been working in a different direction from the same starting point. In other words, text search can not help the user who has no more than a formula to work with and wants to find out more information about it. That's why articles containing mathematical formulae are not indexed solely on their string contents. In order to index the formulae, they have to be encoded in some sort of representation which can be queried later at search time. The most popular and, by now traditional, ways of encoding mathematical formulae is in LaTeXor XML format. Projects already implementing these ideas are the `Digital Libray of Mathematical Functions` [DLM05], which makes use of tools converting LaTeX formulae into text for indexing and uses a combined text + LaTeX query for searching and the `MBase` [MBa06] collection of formalised mathematics content, which uses `OMDoc` and `Content MathML` XML representations for indexing and searching. But these two search engines do not provide semantic capabilities, where the query can contain repeated items, such as $f(x_3)-g(x_3)+1$, with x_3 being a wildcard, matching any term. `MathWeb Search` [KŞ06], on the other hand, provides exactly this semantic search ability, being able to match subterms within formulae, and also keep the syntax clear and easy to understand. At the moment, `MWS` indexes a wide range of mathematical documents, from famous repositories like the one at `Wolfram Research`, but is continuously expanding.

3 Nutch - A Lucene based text search engine

Finding a text-based search engine to use is not such a difficult task. Apart from Google, whose search algorithms are kept secret, many smaller websites have inbuilt search engines based on the open-source Java-based Lucene en-

gine [Luc07]. The latter is a Jakarta project which provides very good already existing indexing and search capabilities in the form of a JAVA API which can be easily integrated into any type of program. Lucene is the best open-source text-search service available and it does not fall behind in quality [Goe00]. It uses multiple indexes that it merges once in a while to store data more efficiently. Lucene is an unusually flexible text search engine: it indexes incrementally, with small indexes (30% of original); runs in very little RAM (independent of corpus size); supports arbitrary boolean queries across multiple user-defined document fields; returns results ordered by relevance; and supports user-defined lexical analysis and stemming algorithms [RKR04]. Both searching and indexing are highly customizable, allowing the user to build a very need-specific database and also retrieve results in a personalized manner. It is also very scalable to large databases, has small memory requirements and has been ported to a number of different programming languages like `C++, Perl, Python, Ruby`, etc. [HG04].

Nutch [Nut07a] is an open source web-search software. It builds on Lucene, adding web-specifics, such as a crawler, a link-graph database, parsers for HTML and other document formats. Nutch has indexing and searching capabilities for the whole internet but can scale down to intranets and even personal computers [ser]. The Nutch developers pride themselves in the transparency of their ranking algorithms, and extensibility of their product [www]. Nutch has a highly modular architecture that uses plug-in APIs for media-type parsing, HTML analysis, data retrieval, protocols, and queries. The core has four major components [RKR04]:

- **Searcher**: Given a query, it must quickly find a small relevant subset of a corpus of documents, then present them.
- **Indexer**: Creates the inverted index from which the searcher extracts results. It uses Lucene storing indexes.
- **Database**: Stores the document contents for indexing and later summarization by the searcher, along with information such as the link structure of the document space and the time each document was last

fetched.

- **Fetcher**: Requests web pages, parses them, and extracts links from them.

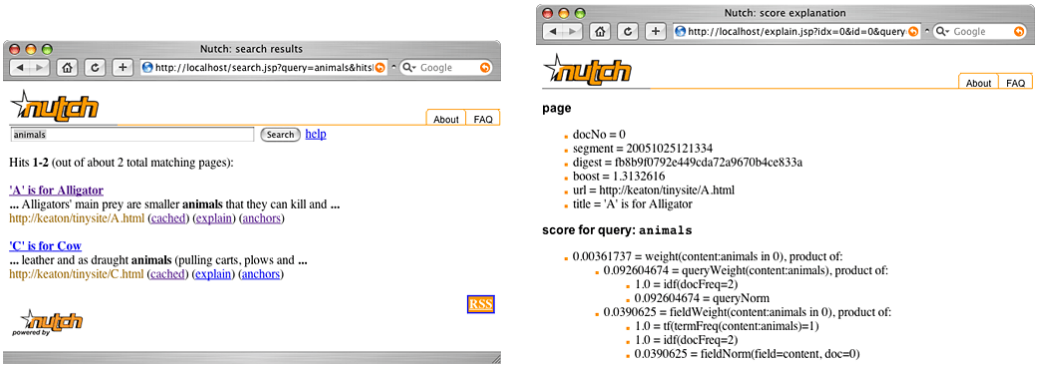

Figure 1: A Nutch webapp search (left) and explanation of the score for a result (right)

The Nutch crawler is highly customizable, starting from a list of user-defined links or a top domain and then crawling to other links found on the pages reached up to a user-specified depth. The user can also specify file type restrictions on indexed items and regular expression-driven filters to eliminate certain websites from the candidates considered for indexing [Nut07b]. The Nutch crawler employs the Fetcher and the Indexer in order to create a Lucene database. For HTML pages, Nutch uses a full-text indexing strategy provided by Lucene, after the documents have been stripped down to their text content. Nutch's search user interface runs as a Java Server Page(JSP) that parses the user's textual query and invokes the search method of a NutchBean [Bat03]. Usually, the GUI of Nutch is a webapp contained in the Apache Tomcat servlet container (see Figure 1). The search capabilities can also be accessed at class interface level, if Java code accessing the NutchBean class is used [Whi06]. Nutch provides a ranking explanation for each hit, sorted using the **org.apache.lucene.search.Similarity** Class. The score of query `q` for document `d` correlates to the cosine-distance or dot-product between document and query vectors in a *Vector Space Model (VSM) of Information Retrieval*. A document whose vector is closer to the query vector

in that model is scored higher [RKR04]. The tests performed at the Open Source Lab at Oregon State University, in mid-2004, found Nutch to be equal in quality to Google, which had been running at the lab since 2002 [Ben04]. For convenience reasons, Nutch was chosen over Lucene to implement the textual part of MaTeSearch, as it provides enough functionality to start the search immediately.

4 MathWeb Search

4.1 A Semantic Math Search Engine

For the mathematical part of the search, we have used the existing MathWebSearch engine, developed by Ioan Şucan and Michael Kohlhase at International University Bremen in 2006 [KŞ06] and released under the General GNU Public License [FSF91]. This search engine uses *substitution-tree indexing* techniques where the index is stored as a tree of terms with every node maintaining an entire subterm. Children of a node are obtained by substituting some of the *generic terms* with actual values but the substitutions themselves are not stored at the nodes. The leaf nodes contain no more *generic terms*, but fully substituted terms. The key to storing mathematical formulae in such an index is to represent them in prefix notation, which is obtained by transformation from `MathML` [ABC+03]. Thus, MathWebSearch indexes only documents containing `Content MathML` formulae or documents which store math in a representation which is easily convertible to `MathML`. MathWebSearch has its own crawlers that fetch documents containing Content MathML from special repositories on the internet, convert the math terms into string representation and store it into the corresponding databases using MySQL. Then, an index is computed from these databases for the search server to run on. The search clients connect to a webserver, which handles all queries. In turn, the webserver connects to the search server to get the results from the index [KŞ06]. At search time, the webserver expects XML or string queries and returns results specifying the link, title and description

of each of the hits, plus Xpointer links to all the locations where the term was matched inside the document. MathWebSearch provides both a GUI and a comprehensive API for easy integration into further systems [Mat07]. The GUI is a webapp which can be accessed at search.mathweb.org and provides a very easy-to-use searching environment. Apart from the XML and string input types for the query, the MathWebSearch website also provides the **WiRiS** formula editor, which converts visual representations of formulae into OpenMath, as you can see in Figure 2. The website uses the API to send special commands to the webserver, which returns the results through a socket. The results returned by the webserver are ordered by the number of matches of the query term [KŞ06]. Hence, a document will rank higher, the more occurences of the query term it contains. At the moment, MathWebSearch indexes over $1,600,000$ terms from the repositories at `http://cnx.org` [CNX07] and `http://functions.wolfram.com` [WOL07] and is continuously expanding.

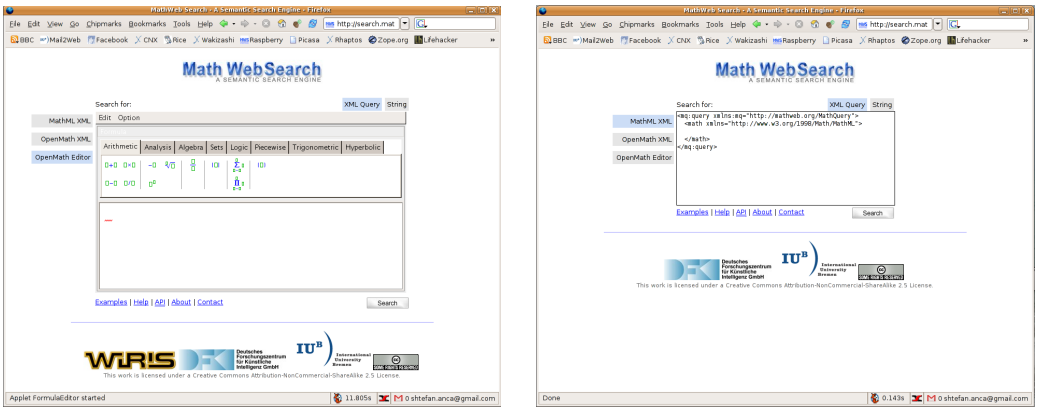

Figure 2: A MathWebSearch at search.mathweb.org. WiRiS formula editor (left) and regular XML query input box (right)

4.2 The Math Query Language

Since MaTeSearch uses MathWebSearch for all the math queries, the query language is also important in the context of this project. MathWebSearch accepts two XML query languages and one string query language [KŞ06].

MathML$^{\mathcal{Q}}$ extends the MathML XML language by adding new attributes like `mq:generic` and `mq:anyorder`, where the prex `mq:` abbreviates the namespace URI `http://mathweb.org/MathQuery`. The second XML-type query language is OpenMath, and the shorthand string representation is similar to the internal representation of the indexed terms. The most important attribute introduced by MathML$^{\mathcal{Q}}$ is `mq:generic`. The `mq:generic` attribute takes a string value and can be specied for any tag in the query. When such an attribute is encountered, the contents of the tag that contains it is ignored and it matches any term encountered in the search process. For, example, the MathML$^{\mathcal{Q}}$ and string representations of $f(x, g(x, y))$ are shown in Table 4.2. This search term matches documents containing both formulae like $f(a, g(a, 3))$ but also $f((m+3)^n, g((m+3)^n), \sqrt[n]{m}))$. The query language for Nutch is simple string text with classical search engine rules, like the use of quotation marks "" for specifying exact string queries [Nut07b].

MathML$^{\mathcal{Q}}$ representation

`<apply>`
`<ci>f</ci>`
`<ci mq:generic=x/>`
`<apply>`
`<ci>g</ci>`
`<ci mq:generic=x/>`
`<ci mq:generic=y/>`
`</apply>`
`</apply>`

String representation
`f(@x,g(@x,@y))`

Table 1: Query Representations

5 Bringing math and text together in one search

5.1 Main Research Question

The main question posed by this research project deals with the integration of two different search engines in order to produce a unitary result. What this comes down to, in the end, is the question of how to take a math query and a string query and intersect the result set. This problem could have been solved trivially if both queries were of the same type: then a single search engine combining both searches would have been used. Lucene offers the option to search for two different strings (or two different fields of type string) at the same time by concatenating them with an `AND` keyword [HG04]. Similarly, MathWebSearch concatenates two different queries with the `<mq:and>` extension tag [KŞ06]. It is at the moment very difficult to design and implement a search engine that can do both math and text search at the same time by keeping a single combined index. First, a new indexing method would have to be found that would combine both representations of information and second, a query language that would allow mixed text + math search queries would have to be conceived. Taking the two existing search engines and combining the way queries and results are intersected to output a unitary result is much easier in terms of time and computation. For the purposes of testing the MaTeSearch engine, all the documents in the Connexions [CNX07] database were indexed by both MathWebSearch and Nutch.

5.2 Examples

The goal of this project is to improve the already existing math search engine with a regular text-search addition. This way, the users of MathWebSearch will be able to filter out the unwanted results caused by searching a very popular formula. Assume, (**Example 1**), that *Bayes' theorem*, $P(A|B) = \frac{P(B|A)\cdot P(A)}{P(B)}$ is the search query in MathWebSearch [Mat07]. This formula

is used in different fields of science (from Statistical Physics to Artificial Intelligence) so the result set is potentially very big. By also searching for the name of the theory of interest to the user, the result set will not contain unwanted hits from across all disciplines where this formula is used. Hence, if we search for the Bayes' Theorem formula and **at the same time** search for the string `"Bayesian Networks"`, the set of results will be smaller, therefore narrowing down the search to the useful documents only. This way, the search results will be a closer fit to the intended query. As you can see in Example 1, both A and B are entered as generic identifiers, since authors don't usually stick to one name for the variables they use in such general formulae.

Example 1: MaTeSearch query with Content MathML representing Bayes' Theorem

MathML query

```
<apply>
  <eq/>
  <apply>
   <ci>P</ci>
   <condition>
    <ci mq:generic = "B" />
   </condition>
   <ci mq:generic = "A" />
  </apply>
  <apply>
   <divide/>
   <apply>
    <ci>P</ci>
    <apply>
     <intersect/>
     <ci mq:generic = "A" />
     <ci mq:generic = "B" />
    </apply>
   </apply>
   <apply>
    <ci>P</ci>document
     <ci mq:generic = "B" />
   </apply>
  </apply>
</apply>
```

String query

```
Bayesian Networks
```

For **Example 2**, let's consider the use case where an engineer who has graduated a few years back from college, needs the formula for the probability den-

sity function 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 the engineer does not know this. He/She 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, he/she enters the search query $f(x, y)$, having some memory of the actual formula. If this search query is entered into MathWebSearch alone, with wildcards for the arguments, as shown in Example 2, the search engine will return 3 pages of matched documents, with the article on *Sums of Random Variables* listed at the bottom of the second page. If the engineer is unsure exactly which probability document to open, he/she will spend some time browsing through the first page of results to figure out exactly what formula he/she is looking for. Using MaTeSearch, the engineer adds the string query `"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 at the bottom of the first page of results. Thus, the time to search for the right result is cut in half and it is therefore, much easier for the engineer to quickly access the desired information.

As we can observe in Example 2, both the XML and String queries can be very vague and can point us in very many different directions if used by themselves. But, when matched together, 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.

6 Development Decisions

Since the text-search functionality was provided by Nutch through a class interface that gives simple access to indexing and searching functions [Goe00],

Example 2: MaTeSearch query with Content MathML representing $f(x, y)$

MathML^Q query

```
<apply>
  <fn><ci>f</ci></fn>
  <ci mq:generic="x" />
  <ci mq:generic="y" />
</apply>
```

String query

```
variable
```

all the work that had to be done on the text-search side of the project was to index the same repositories that MathWebSearch indexes. First, Nutch version 0.8.1 [Nut07a] was installed on the same server as MathWebSearch, at search.mathweb.org. Since MathWebSearch keeps a database of all indexed links, a simple MySQL query pulled out the URLs of all documents from the Connexions database into a file, which was then fed as a list of websites to crawl over to Nutch. A simple default nutch crawl was performed on these links alone (with the crawl depth set to 1, so that the crawler would not try to open other links found on the specified webpages) [Nut07b]. Thus, a Lucene index of the same Connexions documents that are already indexed by MathWebSearch was obtained. This was the first step, as both search engines had indexes of the same documents.

Next, having the two search engines operational and queryable on the same machine, a method to connect them had to be devised. Basically, the MaTeSearch code has to take two input queries from the user, one string and the other one XML, feed them into the corresponding search engines, retrieve the set of results from each of them and then perform an intersection to output a single result set back to the user. The architecture that was used is shown in Figure 3. Communication with MathWebSearch is done through the API described above. A python script opens a socket that connects to the webserver `webappD` on a special port and sends the XML or string shorthand representation of the query. The script intercepts the result of the webserver, parses it to obtain the document links out of the Xpointers

and passes a modified output further down the execution chain. The python script can take the math query input as a command line argument or read it from a file and can return it the same way. The script also has the option to output the raw, unprocessed result string, therefore only acting as a communication tool to the webserver. Python was chosen as the programming language for the script because the socket communication to the webserver is easy to set up and manage in this programming language. Communication to the Nutch search engine is done by connecting directly to the core Nutch search class org.apache.nutch.searcher.NutchBean, following the example of the JavaScript that runs behind the Nutch webapp [Whi06]. This connection is achieved through a SearchApp Java program which is the front end API for MaTeSearch. SearchApp also connects to the Python script to retrieve the MathWebSearch results, as specified above.

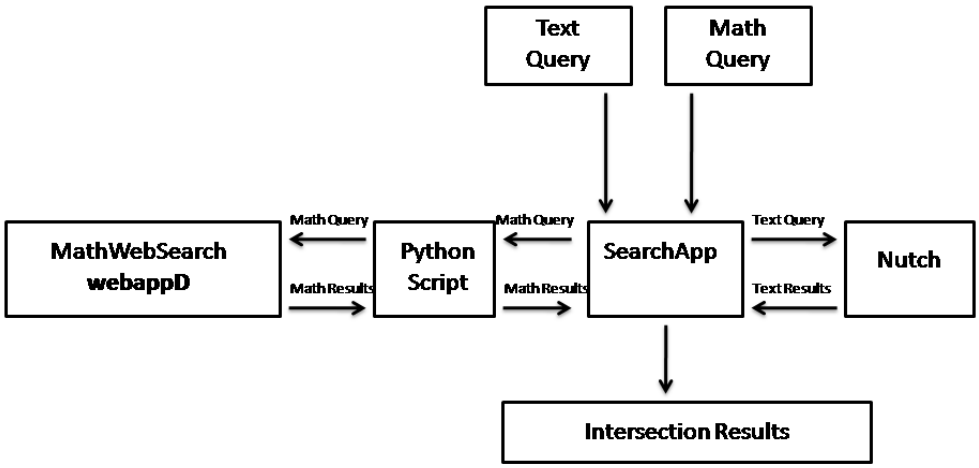

Figure 3: The MaTeSearch program architecture

In conclusion, the user inputs a text query and an XML/string-shorthand math query to SearchApp (through input files or as commandline arguments), SearchApp passes the math query to the python script and retrieves a simpler, modified output and also performs a simple text search on Nutch with the inputted text query. The documents returned by Nutch are then intersected (by link) with the ones returned by MathWebSearch in the same

SearchApp. The results are then outputted to the user but can also be reformatted in MathWebSearch style, in order to provide an API for the website at mathweb.org.

Even though the python script is able to provide subsets of the MathWebSearch results (as a response to the search command `XMLQ` [Mat07]) and NutchBean gives access to the first n results of the Lucene search, SearchApp takes the complete set of results from both sources in order to be able to give a good intersection output. That's why it is recommended that, at least for this stage of development, the math queries inputted directly to SearchApp are `XMLQ.ALL` type of queries [Mat07]. Basically, the way that the search results from these repositories are intersected is very simple. The text query Q^N is inputted to Nutch while the math query Q^M is fed into MathWebSearch in parallel. The complete sets of results R^N and R^M are fed into a component performing the $\cap$, which is SearchApp. Then this component intersects the results and outputs the *documents* that match both queries. We call this result $Res = R^N \cap R^M$.

MaTeSearch is available as a webservice in a very similar manner to MathWebSearch and can be found at search.mathweb.org/MaTeSearch. For the moment, the interface is very similar to that of MathWebSearch, as can be seen in Figure 4, the only difference being the extra TextSearch box and the underlying PHP code.

7 Result ranking analysis

7.1 Problems

As we have it now, the results returned by MaTeSearch are ranked by MathWebSearch, meaning that the document with most matches of the math query term ranks first, indifferent of its Nutch ranking. This is meant to be a temporary ranking, as it does not correctly return the most desirable document from the point of view of the string query. For example, if the user

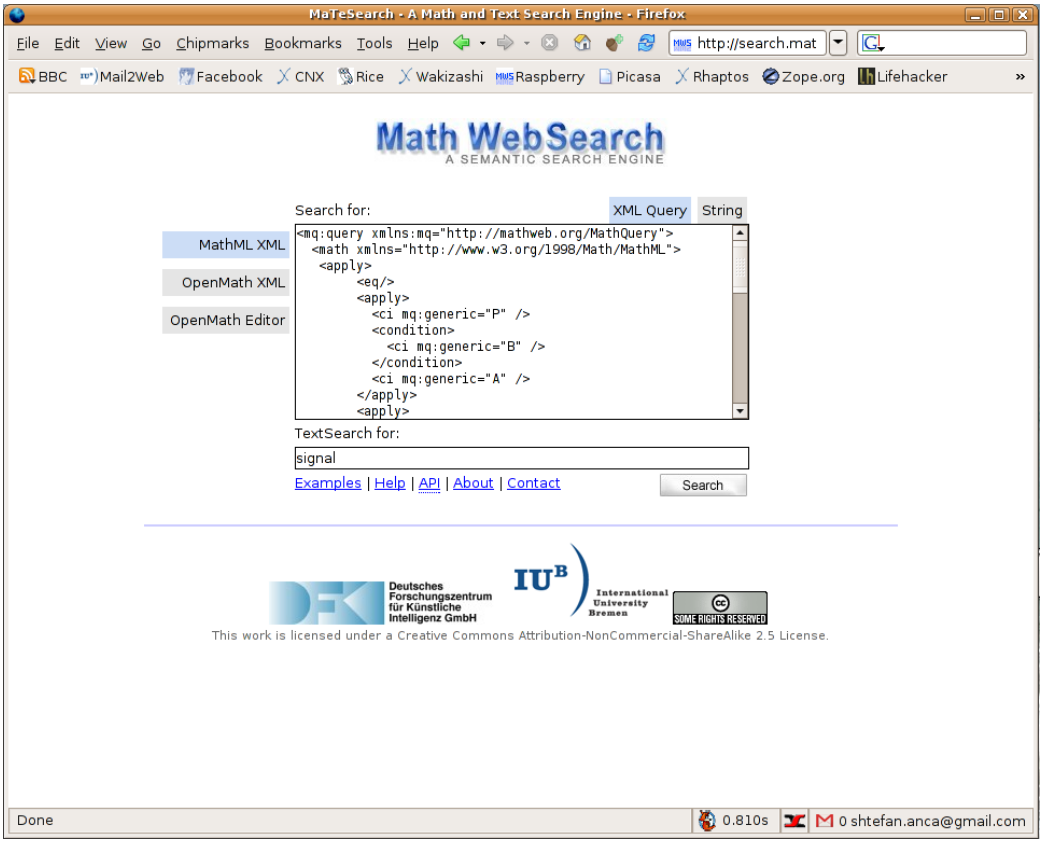

Figure 4: A MaTeSearch at search.mathweb.org/MaTeSearch

knows quite well what he/she is searching for and inputs the string query `Signal Circular Convolution`, which is defined as

$$(f(t) \circledast g(t)) = \int_0^T \int_0^T f(\tau)g(t-\tau)d\tau dt$$

but also inputs the XML math query representation of

$$\int_{@t1}^{@t2} f(@x)e^{-(2\pi i f @x)}d@x$$

the results will not point the right way. The XML math query represents the Fourier Transform, which, although related, might not appear in a document on Circular Convolution. Indeed, running a MathWebSearch on the XML query representing the above term did not output the desired document

on Circular Convolution (it just happens that the Connexions document entitled *Circular Convolution Property of Fourier Series* uses ω in its Fourier transform formulas instead of $2\pi t$). Hence, the intersection result will not yield the document mentioned above, even if its subtitle is called exactly *"Signal Circular Convolution"*. Similarly, a user that remembers a formula very well and searches for a complicated term without any wildcards, like Cauchy's Integral Formula with the following choice of variables

$$f(a) = \frac{1}{2\pi i} \int_C \frac{f(z)}{z-a} dz$$

but also for a random string `Johnson`, for the reason that he/she is mistaking one of the authors names for example, will not get back the document where the formula matches exactly (the single result returned by MathWebSearch). The result set will be empty, as the math query is too precise to match more than one document.

As we can see, these are examples where the current ranking scheme of MaTeSearch is undesirable and the user would have to change his/her math or text query in order to get the right result. However, a good search engine should be able to provide the user with those results that match some parts of the user query perfectly. We are now going to analyze what we need to look at in order to create a good result ranking heuristic for MaTeSearch. The outcome of this analysis is not implemented in the current version of MaTeSearch but should provide a good start for a future project attempting to change the ranking of MaTeSearch.

7.2 A different ranking Heuristic

First, looking at the examples above, we realize that a result ranking strategy that throws away all non-intersecting documents is non-optimal. The two different result sets have to be taken separately with their independent ranking scores given by the two search engines and then combined with a proper heuristic. Taking a closer look at how MathWebSearch works, we

see that the resulting documents do not have a percentile ranking score. As mentioned above, they are just ranked by the decreasing number of matches of search terms. We would like to create a ranking for MathWebSearch documents in order to be able to compare desirability on the same scale with the Nutch ranking, which goes from 0 up to 1, where a score of 1 denotes the perfectly matching document. MathWebSearch only provides us with results specifying the number of matches, so we have to use this infomation to create a score from 0 to 1. First, let us set an average upper limit for the number of matches in a document l_a. Looking at a few reasonable searches at search.mathweb.org, we see that an average search does not score documents with more than 6 or 7 matches per document. To stretch the interval a little, we assume $l_a = 7$. Now that we have the discrete interval $[1, 7]$, we have to extrapolate it to the real interval $[0, 1)$. We do this with the help of function

$$f_{MWS}(x) = 1 - e^{-kx}$$

where $0 < k < 1$ has to be chosen such that $f_{MWS}(l_m) \approx 1$. Our choice is $k = \frac{3}{4}$, such that $f_{MWS}(l_m) = 0.82$ is considered a near-perfect result. Documents with a higher number of matches will score slightly higher on this scale, but not by much. Also, the "worst" documents will score $f_{MWS}(1) = 0.22$, see Figure 5.

Now, we have an approximate ranking on a scale of 0 to 1 for MathWebSearch document results and another one for Nutch results and we are ready to combine them to create a global ranking score. Basically, the global ranking score $R(d_n)$ of document d_n will be calculated like this:

$$R(d_n) = w_M(R^M) \cdot f_{MWS}(d_n) \cdot R_M + w_N(R^N) \cdot f_N(d_n) \cdot R_N$$

where $w_M()$ and $w_N()$ are weighting functions that weigh MathWebSearch and Nutch result documents, respectively and $f_N()$ is the ranking score that Nutch provides for its results. R_M and R_N are indicators, which take the value of 1 or 0, depending on whether the document appears in the Math-

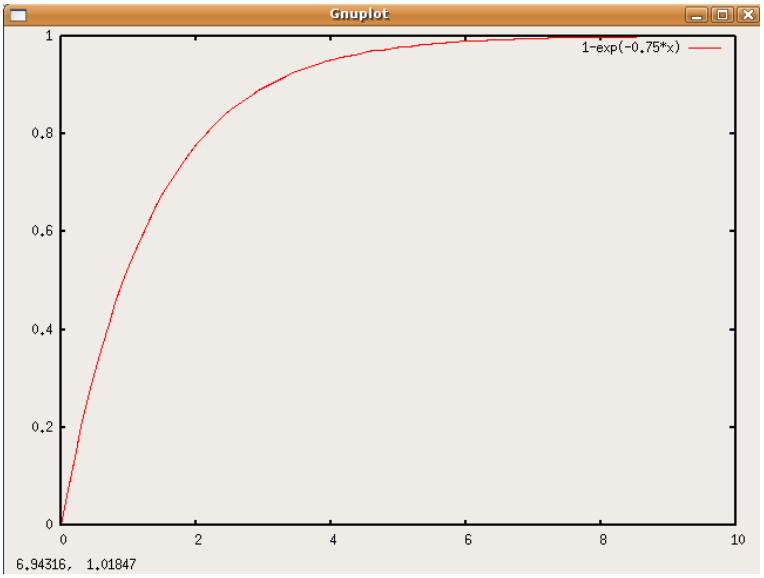

Figure 5: The MathWebSearch result ranking extrapolation function f_{MWS}

WebSearch or Nutch result set or not, respectively.

$$R_M = (int)(d_n \in R^M)$$

$$R_N = (int)(d_n \in R^N)$$

The functions that we have to determine are the two weighting functions w_M and w_N. To do this, we propose that we take a step that regular search engines do not take while ranking and look at the sizes of the result sets. These are useful because they show us the relevance of the individual results and determine the aforementioned weights.

7.3 Approximating the weights

In the case of the Nutch result set, R^N, a small number of results can be interpreted to mean that the search was very "to the point" and only the relevant documents were returned. A really small number of results or no results at all is usually interpreted as the outcome of a defective query, like a spelling mistake or an unreasonable high number of search words. On the

other hand, a very high number of results might indicate that the search was too vague and the results are too broad or that the search term is a very frequent word. Finding a good function that will match all these requests is not an easy job, and there are many possibilities to choose from, like a Binomial or Gaussian distribution. To try to imitate this behaviour as closely as possible, we propose the tweaked Gaussian:

$$w_N(R_N) = w_N(x) = a \cdot \frac{1}{\sigma\sqrt{2\pi}} \cdot \exp\left(-\frac{(\frac{x}{b} - \mu)^2}{2 \cdot \sigma^2}\right)$$

where $x = \#R^N$ is the size of the Nutch result set, $\sigma = 4$ and $\mu = 6$. The extra parameters $a = 10$ and $b = 4$ are there to stretch the input range and bring the max of the function to 1, see Figure 6. This function has a peak close to 1 between 24 and 25 (with the help of the tweaking of a and b to the specified values), this being an approximation of the best result set size for an average good text query. Any result set with less than 3 elements scores a weight below 0.3 on this scale. This approximation greatly depends on the size of the repositories and can easily be changed by modifying the parameters a, b, σ, μ. With the help of this weight, we can now calculate the right-hand component of our global ranking function $R()$, by retrieving important ranking information from the number of results returned by Nutch.

The weight for the MathWebSearch results can be calculated in a similar fashion, applying the same argumentation as before regarding the size of the result set R^M. In this case, the Gaussian will be shifted to the left, as MathWebSearch queries which are faulty will output no results at all, while MathWebSearch queries which are too vague (i.e. many wildcards or very simple terms) will provide many results. Considering that the average result size for a MathWebSearch query is smaller than for a Nutch text query, we consider a Gaussian with the following parameters:$\sigma = 5, \mu = 0, a = 12, b = 4$. We call this function $w_M()$. This Gaussian has a peak $\cong$1 at 0 (unreachable), but any query which returns 10 or less results will score a weight of at least a 0.8, see Figure 7.

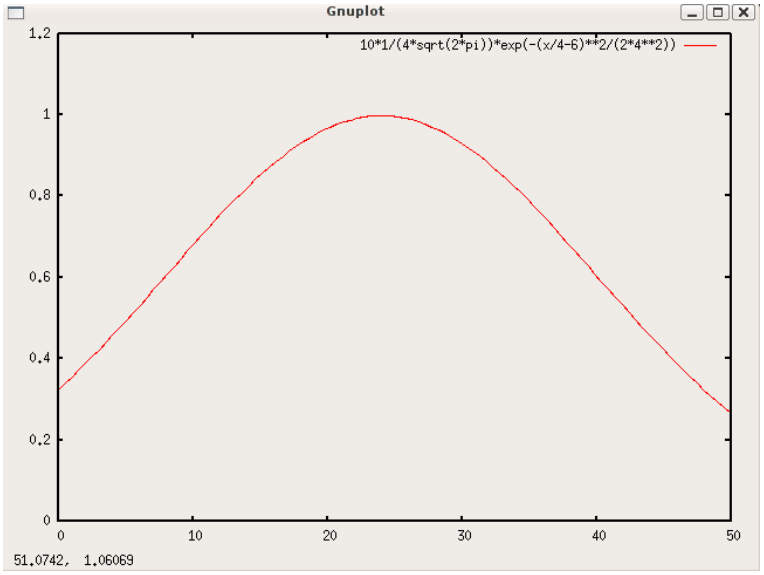

Figure 6: The Nutch weighting function $w_N()$ as a function of the result set size

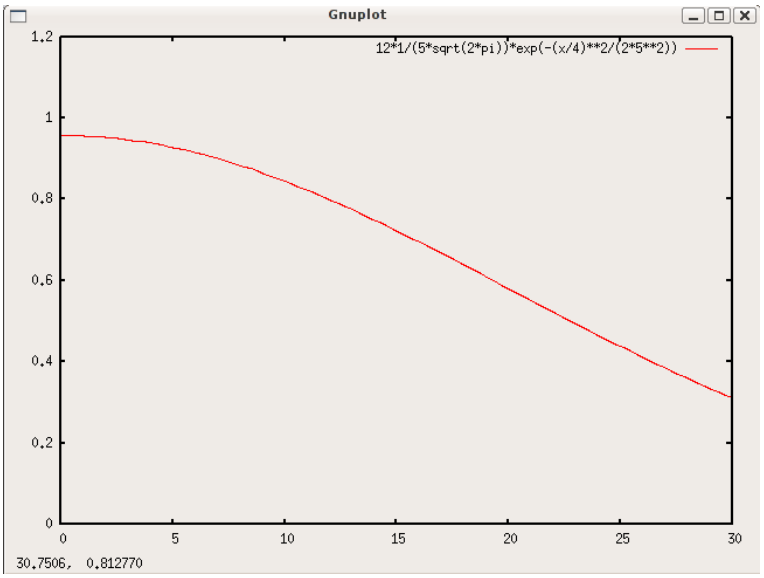

Figure 7: The Nutch weighting function $w_N()$ as a function of the result set size

We now have all the formulae needed to compute the ranking score of a document, with the help of the formula for $R(d_n)$ listed above. This score will be a real value between 0 and 2 (unreachable), but can be scaled down

to the interval $[0, 1)$ by calculating $\frac{R(d_n)}{2}$. As mentioned before, this ranking formula would take care of unwanted events like missing out on key results as shown in the examples at the beginning of this section. Documents which fit the queries perfectly are ranked higher by this heuristic, with the help of the weights and the documents which intersect in the two indeces are given extra points by the addition of both scores. Most importantly, this heuristic does not remove documents from the final result set solely on the reason that they do not appear in both intermediary result sets. This heuristic is going to be implemented in the next version of MaTeSearch and can serve as the basis for ulterior MaTeSearch extension projects.

8 Conclusion and Future work

This project has presented a combined Math and Text search engine called MaTeSearch. It builds on the already existing search capabilities of a Semantic Math Search engine called MathWebSearch and an open-source Text Search engine called Nutch, based on Lucene Java architecture. We have presented combined math+text query capabilities and presented an analysis of a good ranking heuristic.

Comparing MathWebSearch, Nutch and MaTeSearch, we can see the differences between text-based search, math-based search and, of course, the combination of the two. Both MathWebSearch and Nutch have databases indexing the Connexions repository, containing almost 4000 documents. MathWebSearch stores 59000 term rows in its Connexions database and over 86000 term rows from the Wolfram Mathematica repositories. An average MathWebSearch takes a few miliseconds, averaging at around 0.050 sec. A Nutch search on the Connexions repository alone takes a little over a second. A number of searches performed on the same database averaged the search and retrieval time at 1.020 sec. Correspondingly, the MaTeSearch average response time to a search for both math and text is between 1.2 and 1.3 seconds. Obviously, the C server running the MathWebSearch is much faster

than the Java class architecture employed by Nutch and that can be seen in the response times. At the same time, the text search engine has a much bigger database of terms, given the nature of full-text indexing. The MaTeSearch engine could be optimized to respond faster to user queries by removing some of the interprocess communication, by connecting to the webserver direclty from Java, for example.

In the near future, MaTeSearch will use the Nutch crawler to index all of the websites that MathWebSearch indexes at the moment, so that the two search engines can use the same repositories for searching and increase the probability of intersection results. At the same time, MaTeSearch can use the Nutch crawler to index other open-source math repositories, in order to have increased text-search capabilities. Also, the ranking strategy presented in the previous section will be further developed, tested and included in the next version of the system.

References

[ABC+03] Ron Ausbrooks, Stephen Buswell, David Carlisle, Stphane 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, October 2003. Available at http://www.w3.org/TR/MathML2.

[arX07] arXiv.org. Cornell University Library at `http://arxiv.org/`, seen May 2007.

[Bat03] John Battelle. An open source search engine. SearchEngineWatch.com article, seen May 2003. http://searchenginewatch.com/showPage.html?page=3071971.

[Ben04] Lyle Benedict. Comparision of nutch and google search engine implementations on the oregon state university website. CommerceNet Labs Technical Report 04, November 2004.

[CNX07] Connexions. Project home page at `http://cnx.rice.edu/`, seen February 2007.

[DLM05] Digital Library of Mathematical Functions. Project home page at `http://dlmf.nist.gov/`, April 2005.

[FSF91] Free Software Foundation FSF. Gnu general public license. Software License available at `http://www.gnu.org/copyleft/gpl.html`, 1991.

[Goe00] Brian Goetz. The lucene search engine: Powerful, flexible and free. JavaWorld.com article, 2000. http://www.javaworld.com/javaworld/jw-09-2000/jw-0915-lucene.html.

[HG04] Erik Hatcher and Otis Gospodnetic. *Lucene in Action (In Action series)*. Manning Publications Co., Greenwich, CT, USA, 2004.

[KŞ06] Michael Kohlhase and Ioan Şucan. A search engine for mathematical formulae. In Tetsuo Ida, Jacques Calmet, and Dongming Wang, editors, *Proceedings of Artificial Intelligence and Symbolic Computation, AISC'2006*, number 4120 in LNAI, pages 241–253. Springer Verlag, 2006.

[Luc07] LUCENE. Project home page at `http://lucene.apache.org/java/`, seen February 2007.

[Mat99] MATHSEARCH. Project home page at `http://www.maths.usyd.edu.au/MathSearch.html`, seen May 1999.

[Mat07] Math web search. Web page at `http://www.mathweb.org/`, seen March 2007.

[MBa06] MBASE MATHEMATICAL KNOWLEDGE BASE. Project home page at `http://www.mathweb.org/mbase/`, seen May 2006.

[Nut07a] NUTCH. Project home page at `http://lucene.apache.org/nutch/`, seen May 2007.

[Nut07b] NUTCH WIKI. Wiki page at `http://wiki.apache.org/nutch/`, seen May 2007.

[RKR04] Kragen Sitaker Rohit Khare, Doug Cutting and Adam Rifkin. Nutch: A flexible and scalable open-source web search engine. Oregon State University, June 2004.

[ser] Intranet search with nutch. TheServerSide Java Symposium, Las Vegas, seen May. http://wiki.apache.org/nutch-data/attachments/Presentations/attachments/serverside1.pdf.

[Whi06] Tom White. Introduction to nutch. Java.net article, seen April 2006. http://today.java.net/pub/a/today/2006/01/10/introduction-to-nutch-1.html.

[WOL07] Wolfram. Project home page at http://functions.wolfram.com, seen February 2007.

[www] Nutch: Open source web search. WWW2004, New York, seen May. http://wiki.apache.org/nutch-data/attachments/Presentations/attachments/www2004.pdf.

[Zen07] Zentralblatt MATH. FIZ Karlsruhe project page at http://www.zblmath.fiz-karlsruhe.de/MATH/home, seen May 2007.