Extending Full Text Search Engine for Mathematical Content

Jozef Mišutka and Leo Galamboš

Charles University in Prague,
Ke Karlovu 3, 121 16 Prague, Czech Republic
E-mail: jmisutka@gmail.com

Abstract. The WWW became the main resource of mathematical knowledge. Currently available full text search engines can be used on these documents but they are deficient in almost all cases. By applying axioms, equal transformations, and by using different notation each formula can be expressed in numerous ways. Most of these documents do not contain semantic information; therefore, precise mathematical interpretation is impossible. On the other hand, semantic information can help to give more precise information. In this work we address these issues and present a new technique how to search for mathematical formulae in real-world mathematical documents, but still offering an extensible level of mathematical awareness. It exploits the advantages of full text search engine and stores each formula not only once but in several generalised representations. Because it is designed as an extension, any full text search engine can adopt it. Based on the proposed theory we developed EgoMath — new mathematical search engine. Experiments with EgoMath over two document sets, containing semantic information, showed that this technique can be used to build a fully-fledged mathematical search engine.

Key words: mathematical discourse, language processing, mathematical searching, full text search engine, indexing

1 Introduction

There are several ways how to create and publish semantically annotated mathematical content. However, these documents are still a minority of the mathematical content on the WWW. Among the commonly used document formats to exchange mathematics (LaTeX, MathML, PDF, PS, Word) only MathML contains support for semantics.

The success of full text search engines has shown that despite missing semantic information satisfactory search results can be produced. Although, currently available full text search engines can be used on documents containing mathematical content too they are clearly deficient in almost all cases.

We present a technique how to index and search for mathematical content on the WWW using full text search engine. Every full text search engine can easily adopt it because it is designed as an extension. It is primarily intended for real-world scientific documents which do not implicitly contain semantic information. It still offers an extensible level of mathematical awareness supporting also similarity search. We developed a new mathematical search engine — EgoMath — based on Egothor v2 full text search engine [1] using the technique described in this paper.

The rest of the paper is organised as follows. Section 2 briefly describes the state-of-art of mathematical searching. Section 3 gives the general overview of the design. In Section 4 and Section 5 the proposed technique is described in detail. Section 6 includes experimental results using EgoMath. It shows how performance is effected when changing properties of the search engines. Conclusion and future directions are discussed in Section 7.

2 Related Work

As described in [2] and [3] there are two main approaches in mathematical searching. MathDex [4,5], LeActiveMath [6] use the first mainly "syntactic" approach and MBase [7], Helm [8] search engine and MathWebSearch [3] use the second "semantic" approach. There are few search engines which use neither the formula syntax nor semantics but still can be considered as mathematical aware [9]. The closest work to our paper is [5] and [6]. However, they both only take use of syntax and can not handle mathematics.

3 Design

We think that simple textual search either in meta-data or in raw text is very important. The proof is the Whelp search engine [9] which relies on meta-data to describe mathematical formulae. The information retrieval techniques used for this type of searching do not need to be connected with mathematics. That is why we did not want to rely on one specific full text search engine and designed our mathematical search engine as an extension to an arbitrary full text search engine. The architecture is shown in Figure 1. Both affected parts — indexing and searching — are described in detail in the following sections.

4 Indexing Mathematical Formulae

Full text indexing can be thought of as a description of an arbitrary input by textual words — tokens. Identity function can be used when the input consists of simple words. However, mathematical formulae are highly structured without a general canonical form because of equal transformations, different notations etc. We use *linearisation*, *transformation rules*, *generalisation rules* and *ordering algorithm* described below to simplify the complex and highly symbolic mathematical structures into linear structures with well defined symbols.

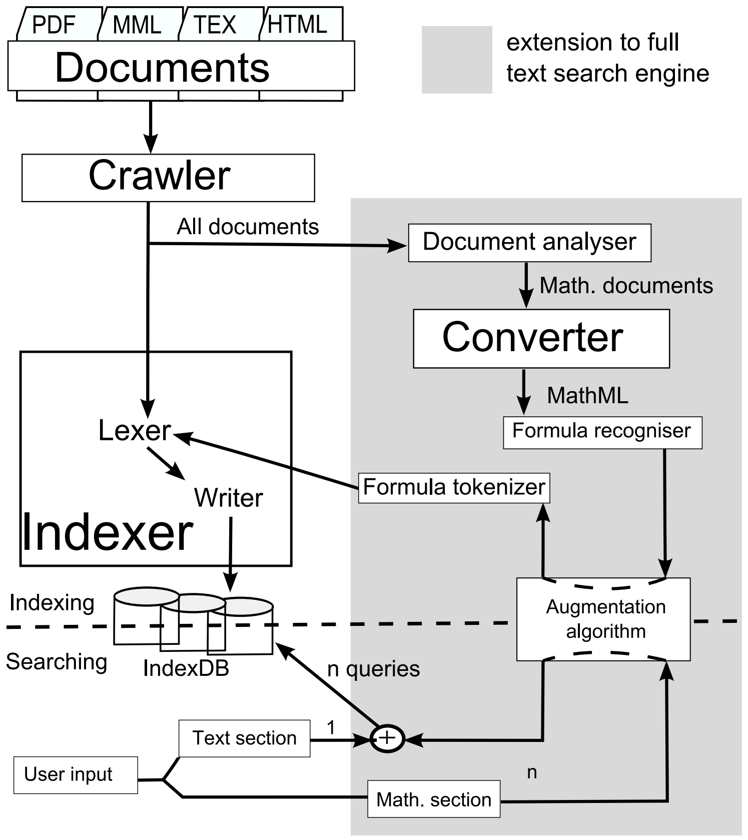

Fig. 1. Architecture of an arbitrary full text search engine

4.1 Parsing Mathematical Formulae

We analysed several formats suitable for mathematical indexing (MathML, LaTeX, TeX, XML, PDF, PS, HTML, Word, OpenMath, OMDoc) capable either of describing the visual presentation of mathematics or describing the semantic meaning. Because the search engine is designed for WWW, one of the basic requirements is to index PDF document format which does not directly support description of formula semantics. Even when the source code of the documents (e.g. TeX, LaTeX) is available not all symbols can be parsed unambiguously [10]. Since the majority of input documents does not contain enough semantic information they must be expressed in one of the presentation formats. MathML was chosen as the primary supported format because it can encode both mathematical visualisation (Presentation MathML) and semantics (Content

MathML). In the following text we mainly focus on parsing Presentation MathML.

Mathematical text is highly structured and symbolic, hence can be easily recognised from common text. Identification of mathematical formulae is minimised to identification of mathematical markup language. This is the task of document analyser. Mathematical documents are sent to the converter. The conversion to supported form must be tolerant because the meaning of symbols is context dependent and there is usually little semantic information to use. Mathematical formulae $a\,b$ resp. $a(b+c)$ are very likely to be the shorter form of $a*b$ resp. $a*(b+c)$ but on the other hand Π can be either the constant or a function representing permutation. A multiple characters to one character mapping has been introduced because many characters look similar or have the same meaning. This phase has shown that is very prone to incorrect symbol recognition which led to incorrect formulae.

Full text search engine must use a recognition technique which analyses input and parses it to words, sentences etc. Mathematical search engine must use an analogous technique and is called formula recogniser. Every document containing mathematical notation is converted to Presentation or Content MathML. Then, the MathML document is delegated to the mathematical extension. Afterwards, it is parsed into a tree-like structure supporting mathematical operations.

There is an important difference between parsing Content and Presentation MathML. Content MathML contains information whether an element is a number, a constant, a variable or any other type but the Presentation MathML does not. To remedy this important deficiency several simple heuristics are applied together with a paradigm described below. *When the correct meaning can not be deduced the solution is to choose one solely meaning and operate with the symbol identically in both the indexing and searching phase*. We can improve this technique by indexing formulae in Content MathML also as they would not contain any semantic information. Each ambiguous symbol is converted to its normal form with predefined semantic meaning, for example π, Π, Pi, pi to function π.

This technique can notably increase recall but decrease precision. The user can refine his search by using simple textual query or by applying similar techniques used by full text search engines e.g. ranking algorithm.

4.2 Storing Mathematical Formulae

Full text indexer works only with single linear words whereas mathematical formulae can be structured into more levels. To adapt the structured notation for sequential indexer linearisation must be performed [11]. Storing mathematical formulae using postfix notation has two main advantages: 1) no need to use parentheses, 2) it enables one special type of similarity searching except similarity searching provided by the generalisation rules. Consider the following example: formula $(a+b)-(c+d)$ is converted to $ab+cd+-$, let's assume that formula tokens are $ab+$ and $cd+$. The resulting index database contains three words in this order: $ab+$, $cd+$, $-$. This representation allows to search for the subformulae ($ab+$, $cd+$) without knowing the mathematical operation between them.

Augmentation algorithm

Mathematical formulae can be expressed in numerous equivalent ways but full text search engines can search only for documents containing specified words. The most important problems include: 1) no commonly used mathematical format nor unitary notation ($1/x = \frac{1}{x} = (x)^{-1}$, $\pi = \Pi = \mathrm{Pi}$), 2) symbol meaning dependent on context, 3) no canonical form ($1+1+a = a+2$, $sin^2x = 1-cos^2x$), 4) structured text ($e^{\frac{x+1}{x-1}}$), 5) many mathematical structures with different axioms.

To fully exploit the full text search engine and reduce the main disadvantages, an indexed formula is not represented only by one word (or ordered sequences of words) but by several words (or ordered sequences of words). Generally speaking, *the input is not stored only once but is augmented and stored in various different synonyms*—it is the opposite of stemming[1]. We call this technique *augmentation*. The first representation is the ordered input formula. Next representation is created by applying transformation and generalisation rules together with an ordering algorithm on the last representation.

Some assumptions are made on the underlying mathematical model which is simplified in each step of the algorithm. A representation from later iteration would match more formulae because it is generalised. Augmentation does not solve the unique canonical form problem completely, but it can reduce the probability that two equivalent formulae do not match. Storing all of the possible representations is clearly impossible because unique canonical form of mathematical formulae does not exist.

Many scientific fields use formulae (physics, mathematics, computer science, medicine, chemistry, etc.) to describe various processes. Many formulae are sound and valid only in specific mathematical structures. In the simplest design, instead of distinguishing between them, all structures are generalised into single one in which these basic and most common axioms hold: 1) *commutativity*, 2) *associativity*, and 3) *distributivity*.

From the mathematical perspective, the indexing stage uses a function Q to create different representations. The domain is the space of all mathematical formulae (F) and the range is $F^N := F_1 \times F_2 \times, \dots, \times F_N$. The function Q produces N formulae $f_1, f_2, \dots, f_N$ for one input formula. The number N is a predefined constant dependent on the generalisation and transformation rules. The function Q is defined as $Q : F \to F^N$, $Q(f) = [f_1, \dots, f_N]$ and must satisfy one requirement about its domain F^N: $\forall i$, f_{i+1} is a generalisation (or identity) of f_i. This algorithm is called generalisation algorithm. There can be more than one function Q with different specialisations because the number of formulae in one document is usually negligible comparing to the number

[1] Stemming is the process where inflected or derived words are reduced to their root form.

of textual words. This list is an example of transformation and generalisation rules:

1. *Partial evaluation:* $7 + a + 5 \xrightarrow{\text{(converted to)}} 12 + a$
2. *Approximate numerical constants:* $5.82 \doteq 6$
3. *Remove brackets using distributivity:* $a * (b + c) \rightarrow a * b + a * c$
4. *Multiply tokens:* $\frac{a+b}{2} * \Pi \rightarrow \frac{\Pi a + \Pi b}{2}$
5. *Assign each numerator its own denominator:* $\frac{\Pi a + \Pi b}{2} \rightarrow \frac{\Pi a}{2} + \frac{\Pi b}{2}$
6. *Replace constants with* const *symbol:*
   $74 + a^2 + b^2 \rightarrow \mathtt{const} + a^{\mathtt{const}} + b^{\mathtt{const}}$
7. *Replace unknown constants, variables with* id *symbol:*
   $a^2 - b^2 + 2bc \rightarrow \mathtt{id}_1^2 - \mathtt{id}_1^2 + 2\mathtt{id}_1\mathtt{id}_2$
   or $\rightarrow \mathtt{id}_1^2 - \mathtt{id}_2^2 + 2\mathtt{id}_1\mathtt{id}_2 \ldots$

Another problem which must be addressed is that mathematically equivalent formulae with the same but permuted operators or operands would be considered as different when compared letter by letter. Ordering algorithmi guarantees that two mathematically equal formulae with the same but permuted operands have the same canonical representation and that two similar (but not equal) formulae have a similar (but not equal) unique representations. This can be guaranteed because of the simplifications and assumptions made on the underlying mathematical apparatus. The indexer usually recognises several document sections e.g. title, body, meta-data. To prevent the ambiguous searching, resulting from collisions between mathematical tokens and simple textual tokens mathematical section is introduced. A search for a proof of a formula could result in searching for word "proof" in the text section and the formula in the mathematical section. When the search engine supports the proximity operator it can be even specified that the word "proof" and the text representation of formula must be at a distance of maximum N tokens.

One of the most important but less obvious problems is the question of what exactly the atomic information (*grain*) in a mathematical search engine is. In a simple full text search engine the smallest information we can search for is a word. The grain of a formula should be its reasonably big fragment—subformula. Formula tokenizer is the part of the system which decides what the atomic information is. When tokens are small the probability of two being equal is higher and as a consequence the index database is smaller. Generalisation rules, like substituting variables for one $\mathtt{id}$ or more $\mathtt{id}_1, \mathtt{id}_2, \ldots, \mathtt{id}_n$ symbols, can be applied either on the whole formula at once or subsequently on all formula grains. All variables must be substituted for one $\mathtt{id}$ symbol when applying the rule on the whole formula. Otherwise, it could break searching for subformulae. Let's assume that formula $a + b$ has two grains a and b. After applying the generalisation rule on the whole formula, we get $\mathtt{id}_1 + \mathtt{id}_2$. The same algorithm used on the indexed formula is applied on the search formula. Thus, when searching for b we will end up with searching for $\mathtt{id}_1$, but we should be searching for $\mathtt{id}_2$. If we apply the rule on each grain separately, we get $\mathtt{id}_1 + \mathtt{id}_1$ and search for b will be successful. This is also demonstrated in the list of example rules above. The tokenizer which was used to create the first representation in 7) divided the input into three grains a^2, b^2, $2bc$; therefore, it marked both first ids with index 1 and the result is $id_1^2 - id_1^2 + 2id_1id_2$. In the second representation the grain is the whole expression and ids can be indexed incrementally.

Using different formula tokenizer has a great impact on the performance. The evaluation of different formula tokenizers can be found in Section 6.

4.3 Ranking Function

Each word in a document has a weight which indicates the relevance to the document. It is a common practise that words in titles are ranked higher than words in body of a document because they are considered more important. If two formulae in different documents match a query ($\doteq$) but both match with a different representation of the formula ($f_i \doteq f_j$ but $i \neq j$, let $i < j$) then document containing f_i should rank higher. Let R be a ranking function which computes word rankings then requirement for the ranking of the formula words can be written: $R(f_1) \geqslant R(f_2) \geqslant ... \geqslant R(f_N)$. The ranking algorithm of mathematical formulae is based on the similarity search which uses formula distance to rank each formula. It is clear that the first representation should be ranked highest and that later representation is less similar than the previous one. Currently, the formula distance is hard coded based on the number of the representation.

5 Searching

Searching phase is the only user interactive phase of a search engine. User enters a query which is executed and the results are displayed. This includes several steps: 1) query parsing, 2) mapping query operators to supported internal constructs, 3) finding all words/phrases from the query, 4) evaluating the logic of the query and collecting suitable documents, 5) sorting them according to their rank, 6) displaying the result list. The mathematical extension is part of 1), 2) and 6).

User input is separated into simple textual query and mathematical query. Afterwards, the mathematical query is processed by the same algorithm used in the indexing phase. The algorithm produces N representations which are appended to the simple textual query, using the AND boolean operator. The result are N sequentially executed search queries. Later query have higher probability of a hit because the mathematical representation is more generalised than the previous one.

The search page and the displaying of results in commonly used full text search engines are similar (Google, Live Search, Yahoo). We extend this interface by adding one or more additional input fields for mathematical formulae. The text query must be present in the text section of the document and the formula in the mathematical section. If there is a match in one step of the algorithm it is more relevant than that the results obtained by using representations from following steps. The queries are performed till the first match of K different documents are found. Different similarities of different representations can be used to limit a search and achieve finer precision.

5.1 Mathematical Query Language

The most important goal is to have the query language as simple and user-friendly as possible but with no limits to the expressivity. Many users searching for mathematics have already made contact with science papers. We can assume that more users are familiar with LaTeX than with any other mathematical document format. Therefore, we propose using LaTeX language extended with tags supporting semantic information. According to a simple survey in [2] the preferred way of inputing mathematical queries is LaTeX too. The query language can be supported by a graphical user interface.

5.2 Displaying Results

There are many ways how to display results. Displaying parts of the text where word/phrase was found is a common practice which helps users to decide which document is relevant without the need to open it. There is no effective technique which extracts interesting parts from found documents without big storage overhead or without undergoing the same process as in the indexing phase which is very time consuming. Another problem is that the found formula representation can be different from the original formula and there is no connection between the original formula and the representation except the position in the document. The searching phase of a mathematical search engine must display at least a small abstract of the text extracted from the document together with the original form of found formulae.

6 Experimental Evaluation

EgoMath is the mathematical extension of Egothor v2 full text search engine. It contains the indexing and searching techniques described in this paper.

Statistics regarding precision and recall are not included since there is no accepted evaluation metric designed specifically for mathematical searching. We think that these results are very little informative. Several recall and precision statistics of EgoMath can be found in [11].

EgoMath uses several simplifications: 1) equations are considered as two separate formulae with equal operator between them, 2) constraints and variables of known operators are not considered ($\int_0^\infty x^2 dx \rightarrow \int x^2$), 3) matrix is converted to a set of formulae. These simplifications are not based on any known limitations.

Every formula is stored in five representations. It uses a superset of rules described in Section 4.2. First two representations have relatively high rank because they are very similar to the original formula. Only basic mathematical operations are applied together with the ordering algorithm. The remaining three representations are created by applying more complex transformations rules. The main intention was to reduce the number of possible representations mapping very common variations to a single one e.g. constants are not important in many parts of mathematics so they are substituted by one symbol. The precise definitions can be found in [11].

6.1 Document Sets

Two different document sets downloaded in July 2007 were used for our experiments: 1) Connections[2] (referred to as *CNX* in the remainder) with 421 scientific documents (32,306 indexed formulae) totalling 99 MB, 2) part of the arXiv[3] (referred to as *ARXIV* in the remainder) with 1915 mathematical documents (852,388 indexed formulae) totalling 252 MB. These document sets were chosen carefully because documents in *CNX* contain both Presentation and Content MathML and because the document set is currently indexed by MathWebSearch and MathDex. *ARXIV* contains both MathML elements but as was already mentioned above, the Content MathML can be ambiguous as it was created automatically by LaTeXML [12,13].

Mathematical Formula Granularity

One of the important observations made is that granularity of formulae has an considerable impact on index database size, speed and mainly on applicability. This section provides comparison of different formula tokenizers responsible for different formula granularity.

We have included 6 different tokenizers. The common used, *com*, accepts subformula with small depth difference and entity count, *coms* accepts only really simple ones, *all* accepts all formulae, *alls* accepts all nodes with neither index nor exponent, *num* accepts only numbers, and *const* accepts only constants.

The number of all representations is the same for all tokenizers because the number does not depend on the algorithm of producing tokens: 1) *ARXIV* — 4,261,940, 2) *CNX* — 161,530. The difference between the maximum and minimum number of different representations is very small: 1) *ARXIV* — max 2,121,729 with *all*, min 2,120,319 with *num* and *const*, 2) *CNX* — max 78,630 with *all*, min 78,518 with *num* and *const*.

Average characteristics are shown in Figure 2. It is interesting that the two different document sets have similar characteristics. The first graph shows that in both document sets the biggest word count is approximately 8 times the size of the smallest one. Second graph shows the average number of words per representation. Tokenizers producing many words have two disadvantages: 1) it

[2] `http://cnx.org/`
[3] `http://arxiv.org/`

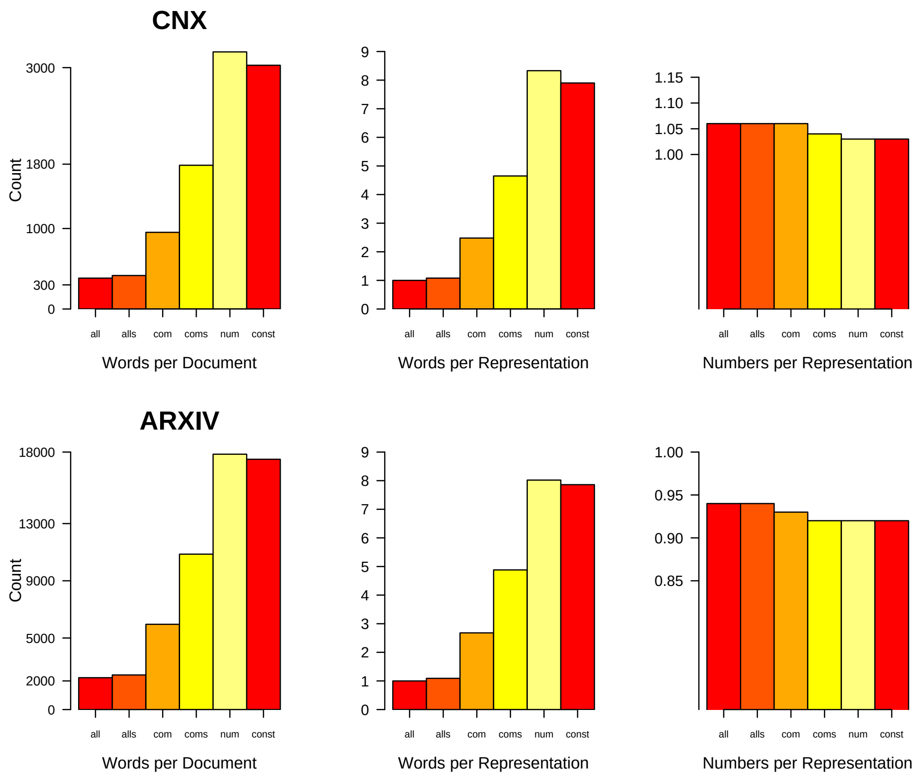

Fig. 2. Average characteristics of document sets

is difficult to reasonably define similar subformulae, and 2) higher word count in one formula can theoretically cause performance problems. The last graph shows the number of numerical constants per one formula representation. It can be seen that there are little formulae with more than one numerical constant.

Index database size

The index database size in this experiment includes the whole index directory including inverted index, indexed meta-data, term occurrences, indexed normal text, index-sequential file for improving performance etc.

Figure 3 shows the comparison of index database sizes when different formula tokenizers used. *all* and *alls* produced the largest databases because they accept all resp. almost all formulae making the words representing a formula very long. Longer words have lower probability that the database already contains them. Tokenizers *num* and *const* accept only formulae with one entity. The probability that two identical words are produced by these tokenizers is higher. As expected, the index database size of *com* is bigger than

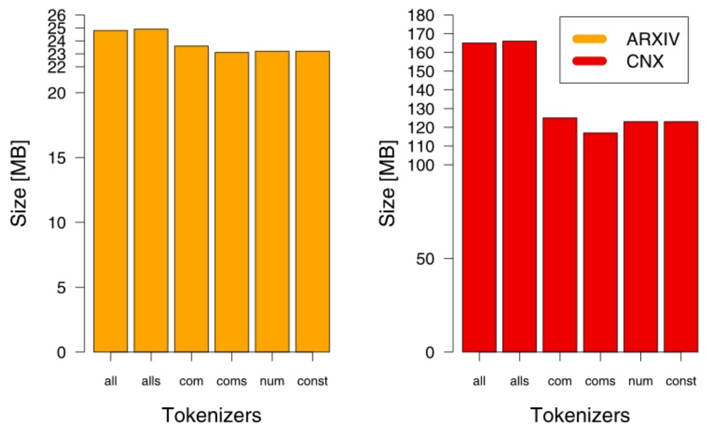

Fig. 3. Index database size using different formula tokenizers

num and *const*. Tokenizer *coms* produced unexpectedly smallest database. The reason for this behaviour is that the index database includes inverted index and word occurrences. There are also other files but either they are very small or have similar size for all tokenizers. The size of the inverted index is the smallest using *num* tokenizer and the size of word occurrences is the smallest using *all*. The medium size of all files is produced exactly by *coms* making the index database the smallest.

PDF support

Applicability has been one of the most important goals of our work. Great emphasis was put on the ability to index PDF format as it is the most used scientific document format. The application [14] can produce MathML from PDF documents. The evaluation showed two small issues: 1) speed, 2) accuracy. A conversion takes tens of seconds and would be impossible to use on a larger dynamic collection of documents. However, we think that this disadvantage is not significant because mathematical documents are changed very seldom. It is assumed that the speed of indexing surpasses the number of new documents. Another problem was accuracy. It is interesting that the newer version outperformed the older one in the number of recognised characters but on the other hand the newer version sometimes converts single character to a set of characters (e.g. M was converted to IVI).

7 Conclusion and Future Works

The key contribution of this paper is a description of how to extend an arbitrary full text search engine to a fully-fledged mathematical search engine. Based on the principles described in this paper we created EgoMath. On one hand, from the full text search engine it inherits the advantages which has already proven as very important in searching, but it also inherits the static index database which does not directly support dynamic indexing and searching for mathematical formulae. By exploiting the current state-of-art of full text searching together with the new described paradigm, searching in real-world scientific documents is possible with an extensible level of mathematical awareness supporting also similarity searching. It is different from all other "semantic" techniques because it does not try to find a user query formula by concretising it. On the contrary, at first it tries to find an exact match. If not successful, the user query formula is generalised and the search is repeated.

Evaluation showed that fine granularity does not only influence the usability from the user point of view but also the speed and size of the index database. The differences can be significant and must be taken into consideration. There are few details which are still missing in EgoMath. One of the most important parts for a user — result displaying — has to be improved according to this paper. There are several features worth of further research which can greatly increase the applicability of EgoMath e.g. searching in meta-data, searching in references, displaying authors. The main focus is now put on the user interface for making EgoMath publicly available. Next step in indexing is to include Wikipedia[4] in our index database.

This work shows that there are many possibilities how to address the problem of mathematical searching opening several questions for future research. How useful is this method? Another question is tightly connected with the first one. How can we evaluate existing mathematical search engines? One of the challenges of this research field is how to measure the applicability. We think that at this moment, only an exhaustive cross comparison of available search engine can produce useful information. We are planning to perform such comparison in the future. It would be also interesting to find out whether the proposed technique could be easily used on other structured data (e.g. chemical formulae). And finally, how can be advanced search operators like proximity operator used to improve the similarity searching.

Acknowledgement

The work was supported by the project 1ET100300419 of the Program Information Society (of the Thematic Program II of the National Research Program of the Czech Republic) "Intelligent Models, Algorithms, Methods and Tools for the Semantic Web Realisation".

References

1. Egothor v2 search engine, `http://www.egothor.org`.

[4] `http://www.wikipedia.org/`

2. Zhao, J., Kan, M., Theng, Y. L.: Math Information Retrieval: User Requirements and Prototype Implementation. To appear in JCDL '08, Pennsylvania (2008).
3. Kohlhase, M., Şucan, I. A.: A search engine for mathematical formulae. Proceedings of Artificial Intelligence and Symbolic Computation, AISC '06, LNAI 4120, Springer Verlag, Germany (2006).
4. Miller, B., Youssef, A.: Technical aspects of the digital library of mathematical functions. Annals of Mathematics and Artificial Intelligence, 121–136 (2003).
5. Miner, R., Munavalli, R.: An approach to mathematical search through query formulation and data normalization. In Towards Mechanized Mathematical Assistants, MKM 2007, 342–355 (2007).
6. Libbrecht, P., Melis, E.: Methods for access and retrieval of mathematical content in ActiveMath. Proceedings of ICMS 2006, LNAI 4151, Springer Berlin/Heidelberg, 331–342 (2006).
7. Kohlhase, M., Franke, A.: MBase: Representing knowledge and context for the integration of mathematical software systems. Journal of Symbolic Computation, Special Issue on the Integration of Computer algebra and Deduction Systems, 365–402 (2001).
8. Asperti, A., Selmi, M.: Efficient retrieval of mathematical statements. In Mathematical Knowledge Management, LNCS 3119, Springer Verlag, 1–4 (2004).
9. Asperti, A., Guidi, F., Sacerdoti Coen, C., Tassi, E., Zacchiroli, S.: A content based mathematical search engine: Whelp. Proceedings of the TYPES 2004, LNCS 3839, Springer Verlag, 17–32 (2004).
10. Stuber, J., van den Brand, M.: Extracting Mathematical Semantics from LaTeX Documents. LNCS 2901, Springer, Germany, 160–173 (2003).
11. Mišutka, J.: Mathematical search engine. Master thesis, Faculty of Mathematics and Physics, Charles University in Prague (2007).
12. Miller, B. R.: Authoring mathematical knowledge. In 2nd North American Workshop on Mathematical Knowledge Management, Phoenix (2004). `http://dlmf.nist.gov/LaTeXML/`.
13. Miller, B. R.: DLMF, LaTeXML and some lessons learned. Hot Topic Workshop on The Evolution of Mathematical Communication in the Age of Digital Libraries (2006).
14. Suzuki, M., Tamari, F., Fukuda, R., Uchida, S., Kanahori, T.: INFTY — An integrated OCR system for mathematical documents. Proceedings of DocEng, France (2003).