A New Mathematics Retrieval System

Shahab Kamali
University Of Waterloo
skamali@cs.uwaterloo.ca

Frank Wm. Tompa
University Of Waterloo
fwtompa@cs.uwaterloo.ca

ABSTRACT

The Web contains a large collection of documents, some with mathematical expressions. Because mathematical expressions are objects with complex structures and rather few distinct symbols, conventional text retrieval systems are not very successful in mathematics retrieval. The lack of a definition for similarity between mathematical expressions, and the inadequacy of searching for exact matches only, makes the problem of mathematics retrieval even harder. As a result, the few existing mathematics retrieval systems are not very helpful in addressing users' needs.

We propose a powerful query language for mathematical expressions that augments exact matching with approximate matching, but in a way that is controlled by the user. We also introduce a novel indexing scheme that scales well for large collections of expressions. Based on this indexing scheme, an efficient lookup algorithm is proposed.

Categories and Subject Descriptors

H.3.3 [**INFORMATION STORAGE AND RETRIEVAL**]: Information Search and Retrieval

General Terms

Design, Algorithms, Performance

1. INTRODUCTION

Retrieving the mathematical content of the Web is currently very limited. Conventional text retrieval systems are not tuned for mathematical expressions, which are objects with complex structures and rather few distinct terms. Moreover, there is no clear definition for similarity between mathematical expressions, and merely searching for exact matches often results in missing useful information.

Each mathematical expression has two sides, its mathematical meaning, often expressed in LaTeX [6] or Presentation MathML [4], and its appearance, usually expressed using Content MathML [4] or OpenMath [3]. The majority of mathematical expressions on the Web are represented by their appearance. The lack of content information forces a retrieval system to rely mostly on the presentation of expressions, which makes it hard to judge whether two expressions are mathematically similar or not. The relevance of two expressions depends on the users' needs, and while in one users's opinion two expressions are related, another user might find them totally dissimilar. For example, a user who is interested in $sin^2(x)$ might also be interested in $sin^3(x)$ but not in $sin^2(x+1)$, and another user might find the former irrelevant and the latter useful. We know of no consensus for similarity in general. On the other hand, by limiting the search to exact matches, many relevant expressions will be missed, and the user might need to issue a large number of queries to find a useful answer. For example if the user is looking for $\sum_{i=1}^{10} \frac{1}{(i+1)^2}$, then $\sum_{j=1}^{10} \frac{1}{(j+1)^2}$, $\sum_{i=1}^{n} \frac{1}{(i+1)^2}$, and $\sum_{x=1}^{n} \frac{1}{x^2}$ probably address her needs.

Currently, the few content-based mathematics retrieval systems [5, 9] are limited to resources that encode the semantics of mathematical expressions, which are not very popular on the Web. There are a few mathematics retrieval systems [2, 8, 10, 12, 13] that rely on the presentation of mathematical expressions, but they either can find exact matches only, or they use a "bag of symbols" model that usually returns a large number of irrelevant results.

The rest of this paper is organized as follows. In Section 2, we describe the problems that we address in this paper. Our query language is described in Section 3. In Section 4 the indexing algorithm is explained, and in Section 5 the matching and lookup algorithms are presented. We finally present an evaluation of our algorithms before concluding the paper.

2. PROBLEM STATEMENT

Given a mathematical expression, the problem is to find expressions that match it. In the absence of a consensus definition for similarity, we have no basis on which to judge relevance, which is necessary to build an automatic retrieval system. We therefore choose to augment the query with extra information from the user. In this paper we propose a powerful query language that uses various forms of wild cards. It is specific enough to reduce irrelevant results while it provides the flexibility to capture many similar expressions.

As an information retrieval system, the mathematics retrieval system should scale well to the size of the Web which is a huge repository of mathematical expressions. Collecting these expressions and efficiently indexing them is also an important problem. We propose a novel indexing scheme

CIKM'10, October 26–30, 2010, Toronto, Ontario, Canada.

that is efficient in terms of memory consumption and lookup time, and allows fast matching and lookup of expressions. Based on this indexing scheme, we propose an efficient lookup algorithm.

We assume all expressions are encoded with Presentation MathML. There are translation tools that automatically translate expressions from most encoding schemes to this form [7, 11], so this assumption does not limit our retrieval system. A MathML-encoded expression can be represented as a tree, e.g. Figure 1. Our indexing and lookup algorithms are based on this tree representation.

3. THE QUERY LANGUAGE

As shown in the introduction, searching only for exact matches to a given expression is not adequate as some relevant expressions would not be retrieved. On the other hand, ranking mathematical expressions based on approximate matching of expressions results in the retrieval of many irrelevant expressions. In general only the user can judge whether two expressions that look similar are useful for her or not. To solve this problem, in this section we propose a new query language that allows approximate matching of expressions, but in a controlled way.

We express queries as mathematical expressions that include *wild cards*. The following wild cards are defined:

- $[Ni]$ matches any number.
- $[Vi]$ matches any variable.
- $[Oi]$ matches any operator.
- $[Ei]$ matches any expression.

In the above, i is a natural number denoting the index of a wild card, e.g. $[N1]$, $[O3]$. If in a query wild cards of the same type have the same indices, they must match subtrees that are equal.For example if the query is $[V1]^{[V1]}$, then it matches x^x and y^y but not x^y, but if it is $[V1]^{[V2]}$ then it matches all three. Similarly, $\frac{[E1]}{[E1]+1}$ matches $\frac{x}{x+1}$ and also $\frac{x^2}{x^2+1}$ but not $\frac{\sqrt{x}}{x^2+1}$, while $\frac{[E1]}{[E2]+1}$ matches all of them. Wild cards with no index, such as $[E]$, are also allowed. If such a wild card is repeated, then the matching subtrees do not need to be equal. For example $[N]x^{[N]}$ matches $2x^3$ as well as $2x^2$.

Constraints can be specified for wild cards in a query using a "where" clause. For example the query "$[E]^2[O1]3$ **where** $O1 \in \{+,-\}$" matches x^2+3, and $(x+1)^2-3$ but not $x^2 \times 3$. For a number wild card, a constraint on the range of the number can be specified, e.g. "$x^{[N1]}$ **where** $1 \leq N1 \leq 5$" matches x^2 but not x^9 or x^{-1}. Constraints can be defined for other types of wild cards similarly. The following is a list of constraints that our system currently supports:

- Number wild cards: The number should belong to a specific range or domain. The domain of a number can be natural, real, etc.
- Variable wild cards: The variable should belong to a specific set of possible names, e.g. "$V \in \{x, y\}$".
- Operator wild cards: The operator should belong to a specific set of operators, e.g. "$O \in \{+,-\}$".
- Expression wild cards: The expression should contain a specific subexpression: "{E *contains* Q'}" (i.e. Q' matches a subtree of E). Q' can be any query, including one with wild cards and constraints.

The constraint on expressions allows us to perform subexpression matching. For example "$[E1]+1$ **where** $E1$ *contains* x^2" matches x^2+1 and $\sqrt{x^2+y}+1$ but not $x+1$, or $\sqrt{x}+1$. "$[E1]$ **where** $E1$ *contains* $[E2]^2$" matches all expressions that contain a quadratic.

We also support optional patterns, which are represented in braces, such as $\{E\}$. An optional part might not appear in some matching expressions. For example, $x^2+[N]$ matches x^2+1 but not x^2, whereas $x^2\{+[N]\}$ matches both.

In our implementation we use LaTeX [6] to express our queries. Although we could have chosen to use any other markup language, LaTeX is widely used for publishing mathematical information and is fairly easy to learn. In addition there are automatic tools that translate expressions from LaTeX to Presentation MathML, e.g. Tralics [7].

4. INDEXING

Assume we are given a set of mathematical expressions in the form of Presentation MathML trees. First, we assign a data structure called *exp-info* to each expression. It stores information such as the Web address of the pages from which the expression is taken, and a list of the labels of all leaves in the same left-to-right order that they appear in the tree. Then, we perform a simple syntactic normalization as follows. We remove nodes that represent numbers and variables. These nodes can be detected by the labels of their parents, "<mn>" and "<mi>", see Figure 1-b. We also remove the nodes representing brackets and add new nodes with a special label, e.g. "()", as the parent of the surrounded expression (see Figure 1-c) (Note that we are *not* performing any mathematical normalization.). In the rest of this paper we assume all trees are normalized and we use terms "tree" and "normalized tree" interchangeably.

Some stored trees represent mathematical expressions taken directly from web pages, and some trees instead represent their subtrees. We call the former a *web expression*, and the latter a *proper subtree*. The label of the root of complete trees is always "<math>", so they can be easily distinguished from proper subtrees.

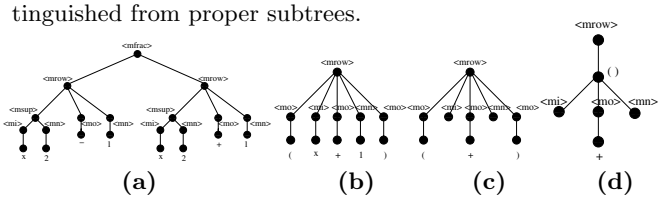
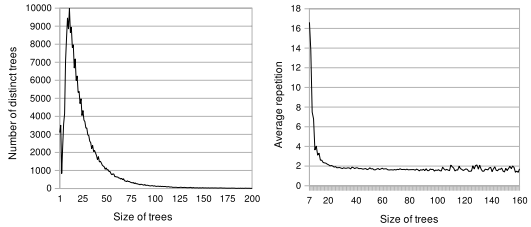


Figure 1: (a) Presentation MathML tree for $\frac{x^2-1}{x^2+1}$ (b) A subtree representing $(x+1)$. (b)After values of numbers and names of variables are removed. (c)After the brackets are transformed.

Our indexing algorithm is based on the observation that many subtrees appear repeatedly in various trees. To investigate this, we ran experiments on a collection of 297,300 expressions taken from articles with various topics. (In Section 6 we explain how web expressions were collected.) The number of subtrees of specific sizes and the average number of their repetitions is presented in Figure 2, which shows that many expression trees share several common subtrees. The basis of our indexing algorithm is to store each subtree once only and to allow subexpressions to point to them. This significantly decreases the size of the index, and as we will explain later, it also speeds up the lookup algorithm. The motivation behind storing the structure of each tree independently of the values of numbers and names of variables (by normalizing them as explained above) is to increase the number of common subtrees further.



(a) (b)

Figure 2: (a) The number of distinct subtrees of specific size. (b) Average number of repetitions.

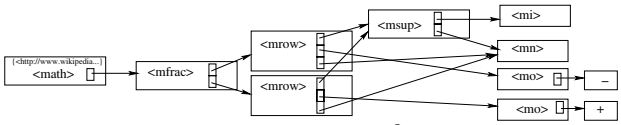
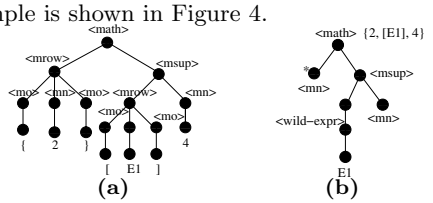


Figure 3: The index after $\frac{x^2-1}{x^2+1}$ is added.

We assign a hash signature to each subtree such that equal subtrees have the same signatures. A table of unique MathML (sub)trees, indexed by their signatures, forms the basis of our index. Each entry of the table contains the label of the root, and a list of pointers to entries corresponding to the list of the children of the root. Given a tree, we find the corresponding entry by computing its signature and looking it up in the table. Initially, the index is empty. We add expression trees one by one to the index. To add a tree, T, we first perform a lookup in the table. If it is found, we return a pointer to the corresponding entry in the table. We also update the *exp-info* if T is a complete tree. If T is not found, we add a new entry for T to the table. Then, we recursively insert subtrees that correspond to the children of the root of T in the index, and insert a list of the pointers to their corresponding entries in the entry for T. This algorithm guarantees that each tree is inserted once only, even if it repeats. In Figure 3 a schematic of the index is shown after $\frac{x^2-1}{x^2+1}$ is added.

5. THE LOOKUP ALGORITHM

Similar to an expression, a query is represented as a MathML tree, with some extra tags that represent wild cards and optional parts. We normalize a query tree as explained in Section 4 and also modify subtrees that represent wild cards. We also mark each optional subtree with a special flag that is stored in its root and remove the surrounding braces ("{}"). An example is shown in Figure 4.



(a) (b)

Figure 4: (a) A query tree representing $\{2\}[E1]^4$ (b) Modified tree.

Consider a tree, T. $T.root$ is the root of T and C_T is the number of children of $T.root$. $T[i]$, $1 \leq i \leq C_T$, is the subtree rooted at the i^{th} child of $T.root$ from the left, and $T[i..j]$ is the sequence of subtrees $T[i]$ to $T[j]$.

Given a query Q and expression E, $match(Q, E)$ returns either true or false. If Q does not contain any wild card or optional parts, $match(Q, E)$ is true if they have the same signatures. If Q contains wild cards, but the root is not itself a wild card, then if Q and E have different roots, the result is false. The result is also false if $C_Q < C_E$. If there are k optional subtrees among the children of $Q.root$, then the result is false if $C_Q - k > C_E$. If none of these cases happen, we recursively match the children of both trees using the boolean function m:

$$m(Q[1..C_Q], E[1..C_E]) = \begin{cases} false & \text{if } match(Q[1], E[1]) \text{ is false and } Q[1] \text{ is not optional} \\ m(Q[2..C_Q], E[2..C_E]) & \text{if } match(Q[1], E[1]) \text{ is true and } Q[1] \text{ is not optional} \\ m(Q[2..C_Q], E[2..C_E])\ OR\ m(Q[2..C_Q], E[1..C_E]) & \text{if } match(Q[1], E[1]) \text{ is true and } Q[1] \text{ is optional} \\ m(Q[2..C_Q], E[1..C_E]) & \text{if } match(Q[1], E[1]) \text{ is false and } Q[1] \text{ is optional} \end{cases}$$

If the root of Q is a wild card, we evaluate the match as follows, and if the match is successful and the wild card has an index, we bind the wild card to E. First if the wild card has an index, we need to determine whether it has already been bound to a subtree because of a previous match having been made when matching another part of the query. If a subtree representing expression E' is already bound to the wildcard represented by Q, then Q matches E only if E and E' are equal, i.e. have the same signatures. If no expression has previously been bound to Q, we need to compare the types of labels at the roots. If Q is a number wild card and E is not a number, i.e. its root's label is not "<mn>", then the result is false. Similarly, variable, operator, and expression wild cards must match variables, operators, and expressions respectively. Otherwise, if there are no constraints on the wild card, we return true.

Assume Q is an operator wild card with the constraint that it should belong to a specific set of operators, S. It matches E only if $E.root$ is "<mo>" and its child's label is in S. If Q is an expression wild card and there is a constraint that E should contain Q', we match all subtrees of E against Q' and return true as soon as a match is found; otherwise, if no match is found, we return false. While matching trees, we keep track of the matched nodes that represent numbers or variables (whose children are removed and stored in the *exp-info* during the normalization). It allows us to check constraints on number or variable wild cards by retrieving the corresponding values from the *exp-info* .

In Section 4, we showed that many subtree structures repeat several times in various expressions. This implies that while performing a search, the same subtrees are repeatedly compared against the query subtrees, so we can cache and reuse the result of these comparisons to optimize our lookup algorithm. In particular, we allocate a local cache for each index entry, and any time we attempt to match the corresponding tree against another tree, we first examine its cache. Only if a cache miss occurs do we run the more extensive match, whereupon we save the result in the cache.

6. EXPERIMENTAL RESULTS

In this section we present the results of the empirical evaluation of our algorithms. We crawled a set of web pages totaling 60GB in size. We extracted 4000 mathematical expressions encoded with MathML (content and presentation), 3000 of them from the web pages of the MathML test suite maintained by W3C. Some main publishers of mathematical information, including Wikipedia and Wolfram, present

Query	Lookup time	Examples of matched expressions
$\frac{x^2-1}{x^2+1}$	250 ms	
$x+y=(x_1+y_1, x_2+y_2)$	275 ms	
$\frac{[E]^2-1}{[E]^2+1}$	410 ms	$\frac{(x+1)^2-1}{(x+1)^2+1}$
$[E]^3$	240 ms	y^3, $(\frac{1}{x})^3$
$[V]^3$	230 ms	x^3, y^3
$[N1]+[N2]$ **where** $N1 \neq N2$	270 ms	$2+4$, $43+70$
$[V]^{[N]}$	220 ms	x^2, y^3
$\frac{k_{[V1]}-k_{-[V1]}}{q-q^{-1}}[O]t_{[E]}$	255 ms	$\frac{k_i-k_{-i}}{q-q^{-1}} \rightarrow t_{i+1}$
$\int_{[V1]=[N]}^{[N]} [E1]d[V1]$ **where** $E1$ **contains** x^2	390 ms	$\int_{x=1}^{10} \frac{1}{x^2}dx$
$\sum_{[V1]=[N]}^{[N]} [E1][E2]d[V1]$ **where** $E1$ **contains** $sin(\frac{\pi}{[V1]})$ **and** $E2$ **contains** $cos(\frac{\pi}{[V1]})$	630 ms	$\sum_{i=2}^{5} \frac{1}{sin(\frac{\pi}{i})} \frac{1}{cos(\frac{\pi}{i})}$
$[E1]^{[E1]}$	270 ms	x^x, $\sqrt{R}^{\sqrt{R}}$
$[E1]$ **where** $E1$ **contains** x^2	460 ms	$2x^2$, x^2, $\frac{1}{x^2}$
$\frac{[V1]}{[V1][O1]1}$ **where** $[O1] \in \{+,-\}$	240 ms	$\frac{i}{i+1}$, $\frac{i}{i-1}$
$[E1]^{[N1]}$ **where** $2 \leq N1 \leq 9$ **and** $E1$ **contains** $sin(x)$	510 ms	$\frac{1}{sin(x)}^5$
$\sum_{[V1]=[N]}^{[N]} \frac{1}{[E1]}$ **where** $E1$ **contains** $[V1]^2$	430 ms	$\sum_{i=1}^{10} \frac{1}{2i^2}$
$x^2\{+1\}$	470 ms	x^2, x^2+1
$\frac{x^2+[N]\{[E]\}}{x+1}$	670 ms	$\frac{x^2+1}{x+1}$, $\frac{x^2+2(x+2)}{x+1}$

Table 1: Sample queries and lookup times.

mathematical expressions as images. They also annotate these with pieces of LaTeX. We could collect a set of 293,300 expressions encoded this way. After refining the expressions to fix their LaTeX errors, we translated the collected expressions into MathML using a TeX to presentation MathML translator [7]. So in total we could collect 297,300 expressions from various resources on the Web [1]. The results presented in this section are based on this set of expressions. We ran our experiments on a PC with a 3GHz AMD Athlon Dual Core Processor, 2GB memory, and Linux version 2.6.24-27 operating system.

We define a simple index to be a set of expression trees that are stored independently from each other. Note that in a simple index common subtrees are stored several times while in our index they are stored only once. Figure 5, in which the x-axis represents the number of indexed expressions, presents the ratio of the size of the simple index to the size of our index as new expressions are indexed. This figure implies that the size of our index grows slower as more expressions are indexed. The reason is that the size of the simple index grows linearly as new expressions are added while the size of our index increases slower because repeating subtrees are stored only once.

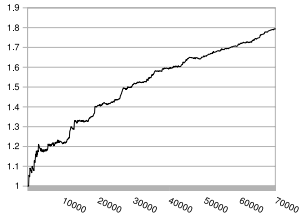


Figure 5: The ratio of the memory usage of the simple index and our index.

Table 1 lists some sample queries and the lookup time (in milliseconds) for each one. Looking up queries that contain optional patterns or require subexpression matching takes relatively longer because in these cases generally a larger number of subtrees are matched. In all cases the lookup times of the listed queries are less than one second. Assuming that these queries represent a wide range of queries that our system supports, we can conclude that our matching and lookup algorithms are quite efficient.

7. CONCLUSIONS

We propose a novel and powerful query language for mathematical expressions that allows users to issue queries that flexibly match various expressions. We also propose an efficient indexing scheme, and based on this indexing scheme, we propose fast matching and lookup algorithms. Our evaluations highlight the performance of our algorithms in terms of memory consumption and lookup time. We believe that our query language will allow users to pose queries that match expressions relevant to their needs, but the effectiveness of the language remains to be tested. We also believe that our approach based on wild cards and our efficient index can be applied to other tree-structured data matching, such as the retrieval of chemical formulae, symbolic logic expressions, or perhaps musical passages.

8. REFERENCES

[1] http://db.uwaterloo.ca/mathretrieval.

[2] G. Bancerek. Information retrieval and rendering with MML Query. pages 266–279, 2006.

[3] O. Caprotti, D. Carlisle, and A. Cohen. The OpenMath standard. *The OpenMath Esprit Consortium*, 2002.

[4] D. Carlisle, P. Ion, and R. Miner. Mathematical Markup Language (MathML) version 3.0. 2009.

[5] T. H. Einwohner and R. J. Fateman. Searching techniques for integral tables. In *ISSAC '95*, pages 133–139, New York, NY, USA, 1995. ACM.

[6] G. Gratzer. *Math into LaTeX: An Introduction to LaTeXand AmS-LaTeX*. 1995.

[7] J. Grimm. *Tralics, A LaTeX to XML translator*. 2008.

[8] F. Guidi and I. Schena. A query language for a metadata framework about mathematical resources. In *MKM*, pages 105–118, 2003.

[9] M. Kohlhase and I. A. Sucan. A search engine for mathematical formulae. In *Proc. of Artificial Intelligence and Symbolic Computation, number 4120 in LNAI*, pages 241–253. Springer, 2006.

[10] R. Munavalli and R. Miner. Mathfind: a math-aware search engine. In *SIGIR '06*, pages 735–735, New York, NY, USA, 2006. ACM.

[11] L. A. Sobreviela. A reduce-based OpenMath-MathML translator. *SIGSAM Bull.*, 34(2):31–32, 2000.

[12] A. Youssef. Search of mathematical contents: Issues and methods. *IASSE*, 2005.

[13] A. Youssef. Methods of relevance ranking and hit-content generation in math search. *Calculemus/MKM*, 2007.