An Approach to Mathematical Search Through Query Formulation and Data Normalization

Robert Miner and Rajesh Munavalli

Design Science, Inc., St. Paul, MN 55101, USA
robertm@dessci.com, rajvm19@gmail.com
http://www.dessci.com

Abstract. This article describes an approach to searching for mathematical notation. The approach aims at a search system that can be effectively and economically deployed, and that produces good results with a large portion of the mathematical content freely available on the World Wide Web today. The basic concept is to linearize mathematical notation as a sequence of text tokens, which are then indexed by a traditional text search engine. However, naive generalization of the "phrase query" of text search to mathematical expressions performs poorly. For adequate precision and recall in the mathematical context, more complex combinations of atomic queries are required. Our approach is to query for a weighted collection of significant subexpressions, where weights depend on expression complexity, nesting depth, expression length, and special boosting of well-known expressions.

To make this approach perform well with the technical content that is readily obtainable on the World Wide Web, either directly or through conversion, it is necessary to extensively normalize mathematical expression data to eliminate accidently or irrelevant encoding differences. To do this, a multi-pass normalization process is applied. In successive stages, MathML and XML errors are corrected, character data is canonicalized, white space and other insignificant data is removed, and heuristics are applied to disambiguated expressions. Following these preliminary stages, the MathML tree structure is canonicalized via an augmented precedence parsing step. Finally, mathematical synonyms and some variable names are canonicalized.

1 Introduction

This article describes an approach to searching for mathematical notation. The approach aims at a search system that can be effectively and economically deployed, and that produces good results with a large portion of the mathematical content freely available on the World Wide Web today. We have implemented this approach in the Mathdex [10] search service and web site.

Our approach follows the general model for mathematical search developed by Yousef [12]. The basic concept is to linearize mathematical notation as a sequence of text tokens, which are then indexed by a traditional text search engine. The text search engine performs atomic queries for terms in the usual way, computing

M. Kauers et al. (Eds.): MKM/Calculemus 2007, LNAI 4573, pp. 342–355, 2007.

rankings for documents using a standard vector space model based on term frequencies and inverse document frequencies. Conceptually, a query for a more complex mathematical expression then becomes roughly analogous to a phrase query, where the text notion of phrase has somehow been suitably adapted to mathematics.

Apart from the algorithmic problem of devising query types and ranking methods suitable for mathematics from the building blocks of atomic term queries, there is the practical challenge of applying the approach to the technical content actually available on the Web. Text is relatively straightforward to identify and extract in most document formats, and to normalize for searching purposes (via stemming, etc.) By contrast, mathematics is often hard to identify and extract, and is encoded in many different ways, both at the level of markup and notation.

To address this problem, the strategy we have employed is to convert all content to a common format, XHTML+MathML, again leveraging existing third-party tools whenever possible. We then employ a multi-pass normalization algorithm that attempts to produce a canonical MathML representation for equivalent mathematical notations. By notational equivalence of expressions, we mean that a typical user looking at them would judge them to be the same mathematical notation. For example, expressions that differ trivially in spacing or that have markup differences that make no visual difference to the typeset appearance should share the same canonical representation.

There are several advantages of this general approach. A key practical advantage is that it leverages the very considerable amount of effort that has gone into developing effective, highly optimized text search systems and conversion tools. In our case, we have chosen to use the Apache Lucene [1] text search engine and Apache Nutch [2] web crawler, together with a variety of existing tools for conversion to MathML, particularly blahtex [11] and LaTeXML [13].

At a more theoretical level, by favoring notational similarity over mathematical similarity, we believe this approach offers users simpler, more familiar query formulation. This also works well with a much larger class of documents, since in most cases, sufficiently detailed semantics for mathematical manipulation are not adequately specified, and cannot today be inferred programmatically with sufficient reliability. At the same time, in many contexts where mathematical semantics are available, the appeal and promise of semantic search is great. In particular, see [3], [4], [5], [6], and [7] for current work in this area. The search system for Wolfram Research's Functions [17] web site is also an interesting approach to semantic search.

This work is supported in part by the National Science Foundation through the National Science Digital Library program under grant number 0333645.

2 Query Formulation

2.1 Mathematical N-Grams

Users want to formulate short queries, and obtain complete answers. Unfortunately, short queries are generally ambiguous in capturing the user's information

need. This fundamental dynamic of information retrieval is magnified in the context of mathematical search. When a user forms a text query by giving one or two keywords, it is likely that the keywords will appear literally in relevant documents. By contrast, when a mathematical expression is given as a query, it is likely that most relevant documents will not literally contain the query expression, but will instead contain expressions that merely share one or more common sub-expressions with query expression. For example, a user querying for $x + y$ may want to match $y + x$ and $x + 2 + y$ with a certain degree of relevance. Thus, the paramount challenge of mathematical search is to identify relevant results by finding expressions that are similar to a query expression while differing in variable names, order and structure in potentially non-trivial ways.

In text retrieval systems, character-based n-grams are effectively used to overcome difficulties such as misspellings and multiple tenses. Instead of indexing individual words, n-grams, (sequences of n consecutive characters appearing in a word) are indexed. At a higher structural level, indexing word-based n-grams (sequences of n consecutive words in a document) is more efficient and effective than searching for literal phrase matches. At both levels, focusing on n-grams instead of literal matches gives a natural and effective way of identifying words and phrases that are similar to a query based on the degree of overlap in the constituent n-grams.

Working by analogy, an obvious approach to quantifying similarity to a mathematical query expression is to build an index of "mathematical n-grams." A mathematical n-gram will consist of a sequence of n consecutive building blocks of some sort, but these building blocks could range from entire expressions, at one end of the spectrum, to individual symbols and characters at the other end of the spectrum, depending on the granularity of the information desired.

We chose to use atomic mathematical notations as the basic unit for mathematical n-grams. These mathematical n-grams range from a single variable to short sub-expressions. For us atomic notations have a one-to-one correspondence with a node or set of adjacent nodes belonging to the same parent in the presentation MathML representation of an expression. This approach constructs mathematical n-grams with retrieval characteristics similar to the text n-grams while still preserving meaningful mathematical structure [14].

The length of an n-gram is defined recursively. All MathML token nodes are considered to be 1-grams. The length of a non-token MathML node is defined as the sum of the lengths of its child nodes, omitting a short list of `<mo>` nodes containing common operators, such as $+$, $-$ and invisible times. This is analogous to ignoring stop words like "he" "is", "and," etc., in text search retrieval. Also, n-grams starting or ending with these stop word operators, as in $+z$ or $2y+$, are not indexed or searched. These operators appear only inside higher n-grams, to give importance to exact matches. Currently, we only consider n-grams of length up to 5.

N-grams are categorized by notational role, which is typically identified by the parent element of the nodes included in the n-gram. Each category of n-gram is indexed separately. Following Lucene terminology, these categories are called

fields in the index. Consider some examples. In the MathML expression, `<msup> <mi>x</mi> <mn>2</mn> </msup>`, the `<mn>` node is placed in the "superscript" field. In the expression `<mfrac><mi>x</mi><mi>y</mi><mfrac>`, both the x and y nodes belong to parent element `<mfrac>`, but in this case, finer control over the notation role is desirable. So here, the first child x is indexed in the "numerator" field whereas y is indexed in the "denominator" filed. Similarly in `<mroot> <mi>x</mi> <mi>y</mi> </mroot>`, x is indexed as "base" and y as "root".

For an example of mathematical n-gram construction, consider the expression

$$\frac{x^2}{2y+z}$$

encoded in presentation MathML as:

```
<math>
   <mfrac>
      <msup> <mi>x</mi> <mn>2</mn> </msup>
      <mrow>
        <mrow> <mn>2</mn> <mo> &it; </mo> <mi>y</mi> </mrow>
        <mo>+</mo>
        <mi>z</mi>
      </mrow>
   </mfrac>
</math>
```

Here, the n-grams structure is:

1 grams: x, 2, 2, Invisible Times, y, $+$, z
2 grams: x^2 , $2y$
3 grams: $2y + z$
4 grams: none
5 grams: $\frac{x^2}{2y+z}$

The search space is divided into fields to increase the precision of query matching. During query time, the query expression is broken down into n-grams. The index is then searched for each n-gram, both in its own primary field, as well as other extended fields, which are mathematically meaningful in that query context. For example, matches in the "nth root" field of the index for a query sub-expression in the "square root" field have a high likelihood of being relevant, where as matches in the "denominator" field of the index are less likely to be relevant. Note that n-grams are only indexed in their own primary fields to accurately represent the information in the document. Limiting indexing to the primary field helps retrieve documents with high precision whereas querying on both primary and extended fields helps retrieve documents similar to the query thus increasing the recall.

The weighting for a match in an extended field is derived from the weight of the primary field. This weight is proportional to an informal notion of structural

similarity between the fields. We have precompiled a matrix of similarity values between different fields that are used to assign weights for the extended fields. As will be discussed below, query terms must exceed a threshold query weight to be considered, and hence extended fields with very low similarity and significance to the entire query sub-expression will be automatically dropped.

2.2 Term Level Query Formulation

Naively searching for all constituent n-grams in a query expression would not only slow down the overall search speed but also retrieve a huge number of non-relevant documents. Merely because a query expression contains x as a 1-gram does not mean that any document containing an x is relevant. To address this problem, it is desirable to choose a small subset of the potential query terms. During the query tree construction, each node in the tree is assigned a weight depending on its structural complexity, n-gram length and depth in the tree. Once the tree is constructed, we select the query terms that satisfy a minimum weight threshold condition, beginning at the root of the tree and traversing the branches all the way to the leaf nodes. Empirically, a good minimum weight threshold is about 25% of weight of the root term.

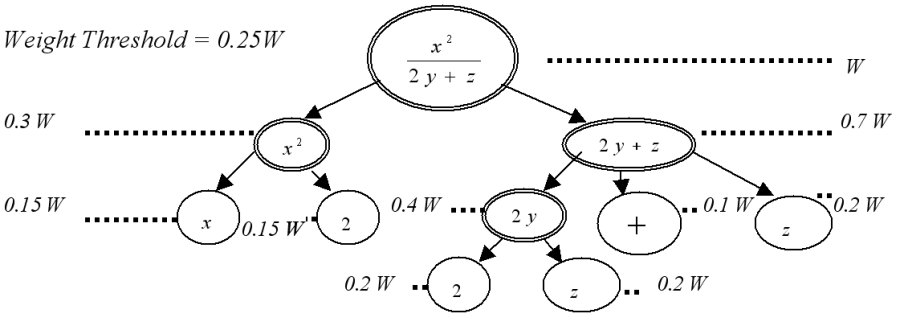

Fig. 1. Weighted Query Tree with a weight of W and minimum threshold $0.25W$

The query term nodes which are selected for inclusion in the final query object are shown in double circled with weight $>= 0.25W$.

The weight of each query term is computed as follows. It is directly proportional to the complexity and length and inversely proportional to the depth of the term node.

```
Weight(term) = (Complexity * Length) / Depth + Special Boost
```

Complexity: Complexity of a term node is computed as the sum of complexities of its child nodes and a predetermined complexity value of the current node. A leaf node typically has complexity of 1. Complex structures like fractions, square roots, matrices etc., get a higher weight thus pruning away the less complex structures which might result in noisy matches.

Length: Length of a term node is computed as the sum of lengths of its child nodes, with the exception of stop word operators as explained above. Longer n-grams have better structural overlap with the query yielding more relevant documents.

Depth: Depth of a term node is equal to the number of branch traversals from the root node to term node under consideration. In general, the higher the depth, the less relevant it is to the query expression.

Special Boost: Certain operators and operands have special meaning in a mathematical expression. For example, common elementary functions such as sin, log, etc., often play a special role in capturing the intention of the user. We have pre-compiled a list of these special operators and operands that are used to boost the weight of the term at query time.

Wild card queries for mathematical expressions are also supported analogously to text wild cards. The scope of the wild card matching is limited to the current mathematical structure. For example, $\frac{x}{*}$ matches any fraction with x in the numerator, but not $x + y + z$. Wild card query terms have less weight compared to their regular counterparts. This is to avoid matching longer ambiguous expressions.

2.3 Expression Level Query Formulation

Query formulation with appropriate weights is vital for effective information retrieval. The challenges of formulating an effective query is emphasized in equation retrieval due to complexity of the mathematical structure and the vagueness in defining mathematical similarity to a given query expression. Meeting the information retrieval need of a diverse user population, from high school math students to research scientists, poses an added challenge.

An effective query ideally should be built around the search task, together with an understanding of the indexing and relevance sorting mechanisms involved. For example, a user with sufficient knowledge to boost the weight of individual query terms and specify specific fields in which to look for them can pose a more intelligent query. Unfortunately, domain expertise, awareness of underlying conceptual search model and its effect on retrieval performance are all the qualities of an expert user. The general user population is far from being willing or able to formulate an intelligent query. Consequently, an expression-level query formulation algorithm, incorporating such expertise, is necessary to attempt to heuristically transform a simple query expression into an advanced query.

We have chosen to optimize our query formulation algorithm to maximize recall to as far as possible without considerably degrading the precision. To this end, our algorithm uses three different logical operations, depending on context, to combine the individual query terms, specifically Boolean, Spanning and Disjunction max operations. The following sections explain how these different logical operations are used and their effect on retrieval performance, using the query expression $\frac{2y+z}{x^2}$ as an example.

Boolean Logic. In general, Boolean IR systems yield high precision but low recall. Terms are either present or absent from the document. By contrast, vector space models result in high recall due to partial matches, and vector space models generally tend to outperform Boolean models due to the fact that precise queries are difficult to construct and Boolean model output has no sense of relevance ranking. For this reason, within the Lucene atomic term queries (in our case queries for individual n-grams) are performed using a vector space model. However, it is still useful to combine the result sets from term queries using the standard Boolean operators AND, OR and NOT[16].

In our algorithm, Boolean combinations of sub-queries are used in situations when the sub-queries have an inherent Boolean logical relationship from the notational structure of the parent query. For example, a query for a fraction naturally decomposes as a query for the numerator and the denominator. For the query example above, the Boolean query "numerator: $2y + z$ AND denominator: x^2 most precisely retrieves documents containing a $2y + z$ in the numerator field and x^2 in the denominator field. Note that precision nonetheless suffers, since there is no guarantee both sub-expressions occur in the same fraction within a document.

Span Logic. Because of commutativity and associativity of common arithmetic operators, some flexibility in the order of terms is mandatory for any mathematical search system. Our algorithm uses span logic to do this. Span logic restricts the search space to only those documents where sub-query objects appear within a given proximity of one another defined by a span scope. In our algorithm, the span scope is computed based on the length of the sub-query objects as well as their weights. To provide mathematically meaningful flexible structure matching, span scopes are limited to consecutive terms belonging to the same index field. For the query example above, a span query "numerator: (2y, z){4}" would match those documents where $2y$ and z appear in the numerator field within an n-gram distance of 4. This would match an expression like $2y + x + z$ in the numerator. Use of span logic generally enhances overall recall.

Disjunction Max Logic. Disjunction max logic is used to select the best match possible among set of terms appearing in different search fields. Unlike the Boolean OR operation, disjunction max operation does not add the search spaces. It rather picks out those search fields that have the maximum possibility of finding relevant documents. This helps to rank the documents in the order of structural preference rather than based on multiple occurrences of a term in the extended fields. Without disjunction max logic, a document with multiple occurrences of x^2, in non-denominator fields would rank higher than a document with only one occurrence x^2 in the denominator field.

Query weights for span and disjunction max logics are significantly higher than the Boolean logic counterparts to achieve higher precision. Empirically we have observed that judicious application of span and disjunction max logic increases precision without undue loss of recall. There is, however, a fair amount of art involved in tuning query weights and logic types. Our algorithm currently uses

a precompiled table of weights and types, based on extensive experimentation, for building up expression queries, taking into account the types of n-grams and index fields involved.

The overall weight of the final query object for a mathematical expression must be normalized for the purpose of comparison of different query objects. Normalization is particularly important when the user further restricts the search scope by providing text search terms.

Text queries are formulated in similar, if simpler, fashion. At index time, a document is analyzed to extract title, heading and body content, which is then indexed in separate search fields. During query time, user text query terms are combined and searched in all three fields with different weight levels. The title is given the highest weight followed by heading and body. Text query weights are normalized with respect to math expression query weights. Both math expression query and text query objects are combined with Boolean logic OR in our current implementation.

3 Data Normalization

The approach to query formulation for mathematical search described in section 2 relies heavily on the assumption that notationally equivalent expressions will be indexed the same way. Unfortunately, mathematical expressions encountered in real documents are very different from mathematical expressions in the abstract; they are represented in diverse markup languages, created by specific authors and tools, all of which introduce their own quirks, conventions and errors. In practice, mathematical expressions that, in the abstract, should be identical can appear very differently to a search system.

The search algorithm described above was originally developed and tested against a limited set of carefully controlled test documents. In order to explore the effects of various strategies for addressing the goals of section 2.1 on precision and recall, it was necessary to use test data where it was possible to determine with confidence what the expected results of a given query should be. We expected the algorithm would perform less well with real-world data, but when we began testing it, we were somewhat surprised to discover that artifacts of encoding, conversion, authoring tools and author coding choices completely dominated, rendering the algorithm virtually useless. Consequently, it was clear that an effective mathematical search system would need to combine a rigorous data normalization component as well as a good search algorithm.

By analyzing documents from a variety of sources, we identified seven areas in which data normalization was required. Since our algorithm operates on mathematical expressions encoded in MathML, the starting point of our analysis included both documents containing MathML directly from the publisher as well as document where we converted the mathematics to MathML using 3rd party conversion tools. We attempted to include MathML from most major authoring and/or conversion systems, including Mathematica, MathType, MathFlow, LaTeXML, Hermes, TeX4ht, itexmml and TtM.

3.1 XML and MathML Error Correction

The first and most fundamental data normalization that the documents we examined required was the correction of XML and MathML errors. Documents containing MathML from publishers or produced with MathML tools largely contained valid MathML. However, essentially all converted documents contained XML errors, MathML errors or both, often systematic and in large numbers. Typically XML errors were unclosed elements, but also run-away attribute values, malformed tags, and other serious errors. MathML errors were usually incorrect child counts, e.g. an `<msup>` element with more or less than 2 child elements.

To address these issues, our indexing workflow begins by passing documents through an error correcting parser we developed that inserts missing end elements, quotation marks, etc. in order to produce well-formed XML. It then checks for MathML element counts, either wrapping extra arguments in merror tags or inserting empty merror elements as necessary. It also strips unrecognized elements and attributes, as well as MathML attributes whose values are not legal.

In addition, during this normalization pass, we implemented a number of ad hoc rules via regular expressions to correct particular systematic problems with the output of specific tools.

3.2 Character Normalization

Perhaps the most significant factor contributing to poor recall for our algorithm turned out to be differences in character data that made little or no visual difference. Unicode contains a number of characters that look like a minus sign, an absolute value bar, and so on. Consequently, a second critical normalization phase chose canonical representatives for each equivalence class of such characters.

3.3 Removal of White Space and Other Non-significant Data

Trivial differences in white space and other non-significant data gives rise to problems similar to the characters discrepancies of the preceding section. While white space can carry meaning in mathematics through alignment, etc., we considered it unlikely that a user would be successful in formulating effective queries to find such cases. On the other hand, the vast majority of white space differences we observed caused serious problems in basic searches, e.g. the presence or absence of a small amount of space before the differential in an integral. Consequently, our third normalization pass removes suspect white space constructs, such as `<mtext>` elements containing only whitespace, `<mspace>`, `<mphantom>`, and `<mpadded>` elements.

Though not technically white space, we also remove other similarly troublesome constructs in this pass, such as `<maction>` elements and semantic annotations. We also remove redundant `<mrow>`s, as in this example: `<mrow> <mrow> <mi>` x `</mi> </mrow></mrow>`.

3.4 Correction of Poor or Ambiguous MathML

The most technically challenging phase of normalization is the application of heuristic rules to improve poor or ambiguous MathML. This has several aspects. The first is fairly straightforward, and consists in the corrections of common errors in MathML coding that produce visually acceptable results using markup that is at odds with the intended mathematical structure.

A well-known example of such an error is attaching a superscript to a parenthesis, instead of the entire base expression to which it applies. Other common problems in converted material include a superscript encoded as an msubsup¿ with a dummy script, or even a `<mmultiscript>` with a single non-empty script. Another common case is a function where the MathML structure it at odds with the natural grouping of the function with its argument, eg. $f(z) = w$ where the $(z) = w$ is grouped in an `<mrow>`.

Beyond such relatively clear-cut problems, there are a number of mathematical notations that are commonly encoded in several different ways. The most important issue is the encoding of elementary functions such as sin x. There are at least three common coding conventions:

```
<mo>sin/mo>mi>x/mi>
<mi>sin/mi>mi>x/mi>
<mi>sin/mi>mo>ApplyFunction/mi>mi>x/mi>
```

While one may argue one is preferable to another, all are valid, so in these cases, our normalization algorithm merely picks a canonical representative.

Another important case where it is necessary to pick a canonical representation is decimal numbers. Both American and European conventions for the comma and decimal point are common, and one format or the other must be chosen. A second issue especially common in the output of translators is that each digit is separately tagged as an `<mn>` in numbers such as 123. The comma and decimal are similarly tagged as operators. For negative numbers, the minus sign may be separately tagged as an operator or be included in the CDATA of an `<mn>`.

Finally, it is very common for expressions to be sub-optimally structured from the point of view of the XML nesting structure reflecting the mathematical structure. One commonly encounters expressions such as as an unstructured row of characters. In other cases, the same expression will be encoded hierarchically with nested mrows¿ grouping arguments and operators. Since the nesting structure can have a large impact on subsequent normalization operations as well as n-gram formation, we apply heuristics to enrich sub-optimal MathML with additional structure. This involves several steps.

First, we attempt to pair fence operators, and refine the MathML structure by adding `<mrow>`s to group fenced terms. For parentheses, brackets, braces and other fence operators with a left and a right delimiter, a deterministic algorithm is possible, at least once occurrences of fence characters where they do not function as fences (e.g. in the interval notation $(-1, 0]$ and so on) have been noted. However, for notations such as absolute value bars, where the same character

functions as both a right and a left fence character, genuinely ambiguous notations arise. In these situations, we apply heuristics to group terms. After fences have been disambiguated, we apply heuristics to distinguish between ambiguous multiplications and function applications, e.g. is a function applied to an argument, or a multiplication of terms.

3.5 Tree Refinement

The preceding normalizations pave the way for a critical tree refinement stage, where the MathML structure is refined so that no mrow¿ contains operators with more than one precedence level. In simple cases, the tree refinement algorithm reduces to standard operator precedence parsing. However, in order to be well defined on all MathML expressions, and to actually produce the canonical MathML structure that is our goal, it must be augmented in several ways.

First, it is necessary to assign precedences to the many hundreds of notations that function as an "operator" in terms of MathML coding, which includes many mathematical notations that are not commonly considered in traditional arithmetic precedence parsing. Second, the algorithm must be extended to accommodate the MathML notion of an embellished operator, typically an operator symbol that has been decorated with scripts or accents, etc. The outline of such an augmented precedence parser was first presented by N. Soiffer and B. Smith of Wolfram Research to the W3C Math Working group in 1996-7.

3.6 Synonyms

The normalization steps described in the preceding sections do a fairly good job of producing a normal form for most commonly encountered encodings of the vast majority of MathML expressions. While severely pathological MathML expressions still cause problems, our empirical results suggest this class of expressions is no more than a few tenths of a percentage of all expressions.

Where notational normalization leaves off, however, mathematical normalization begins, and the line is not always clear. For the practical and theoretical reasons enumerated in the introduction, we chose not to go down the path of attempting to identify mathematically similar expressions in the context of this project, and instead we have focused on notationally similarity for purposes of ranking search results. Nonetheless, after experimenting with our algorithm on a collection of test documents normalized as described above, pragmatism obliged us to make a few concessions toward mathematical equivalence. Consequently, in a sixth and final normalization pass, we select a canonical representative from equivalence classes of mathematical synonyms.

A very basic and widespread type of notational synonym consists in varying conventions for parenthesizing arguments of elementary functions, e.g. vs. . Similarly, the choice of language for elementary function names introduces additional sets of synonyms. Beyond that there are many notational synonyms from relatively simple ones, e.g. competing notations for permutations, to complex ones, such as

notational synonyms for differentiation. To date, we have only implemented normalization for a few basic synonym classes, where empirical experiment suggested that recall for searches was perceived as particularly poor to most users without them. This is a very coarse, subjective criterion of course, and more sophisticated handling of synonyms is an obvious area of improvement for the future.

3.7 Variable Names

Another area that empirical experiments indicated as critical to user perception of search recall involves variable names. The question of when a variable name is significant is a subtle one, and involves both psychology and mathematical equivalence. A user looking for is quite likely interested in finding documents containing the expression . At the same time, as user looking for is less likely to want to see results like .

The approach we have taken is to define equivalence classes of variable names that are traditionally used for similar mathematical purposes across many areas of mathematics. Examples would be $\{i, j, k\}$ used as indicies, $\{s, t\}$ used as parameters, $\{f, g\}$ used as function names , and many others. At indexing time, we index an expression first using its original variable names, but then also in a separate index where variable names are abstracted as merely labeled instances of variables from one of these equivalence classes. That is, would be indexed as something like

```
function_var_1 ( parameter_var_1 +  parameter_var_2)
```

This enables use at query time to query for both exact and abstract variable names, with exact names naturally given much greater weight.

While this technique improves perceived search recall, it is definitely a blunt instrument. More sophisticated handling of the variable name problem is an area for future work. In particular, it is probably necessary to give users at least some control over variable name handling.

4 Our Evaluation

An ideal IR model should retrieve all relevant documents and only relevant documents, in the order of their relevance. Performance evaluation of an information retrieval model targeting a diverse user group with, different information needs is challenging. Evaluation methods should credit IR models that can retrieve highly relevant documents in the order of their relevance with appropriate rankings. Almost all the evaluation methods suffer from the problem of defining appropriate relevance levels. Binary relevance does not reflect the way humans judge the relevance of a document. Instead each document has a degree of relevance.

We adopted the method of Tetsuya Sakai's Average Gain Ratio[15] using multiple relevance levels. For all the evaluations, our n-gram model with equal weighting and zero threshold cut-off was used as the base model. Subsequent iterations of improvement in algorithms were informally evaluated against this base model to assess the performance. Queries with varying degree of mathematical

complexity and length were selected for this purpose. For each of these queries, our team members assigned multiple relevance levels to the all the documents in the index.

In our informal trials, disjunction max scoring greatly increased the number of relevant documents in the top few as compared to using Boolean scoring. Selecting only those query terms with query weight greater than the minimum threshold filtered spurious documents with no relevance to the query. Also boosting certain query terms over others helped better capture the higher-level mathematical intent of queries. We also found that normalization substantially improved recall. Without normalization, recall was quite poor, due to the variety of ways in which equivalent expressions were encoded. Each of the normalization steps described in section 3 was found to improved recall.

While a careful, quantitative evaluation of precision and recall remains to be done, the following examples are suggestive of the current performance. In a collection of around 40,000 documents (see the next section) a query for the expression $s_1, \ldots, s_n \in \{-1, 1\}$ that is known to be in a document in the collections does not find that page in the top 20 results. No match rates higher than 2 stars on a scale of 0 to 5, with the top hits being lengthy formulas containing $\in \{-1, 1\}$ several times in one case, and $s_1, \ldots, s_n$ in the other. At the same time, a search for $T_{f(x)}L_{f(x)}$ returns the document in which the expression is known to reside as the top hit (it's 2.5 star ranking belying a normalization problem with the star scors). The second document rates half a star, and contains $df : T_x X \to T_{f(x)} Y$. This kind of variation in search performance from query to query is typical, and suggests further research and more careful evaluation is still needed.

5 The Mathdex Search Engine

We have implemented the search algorithm and normalization workflow described above as the Mathdex web application. Users enter mathematics query expressions via a graphical equation editor applet. Additional text query terms are entered via a standard HTML text box. Query results are presented to the user in a list, with a document synopsis and a 'best match' equation for each result document. The synopsis is prepared at indexing time by extracting significant phrases, based on term frequencies and document location, e.g. titles and headings are more likely to appear in the synopsis, as are sentences with using rare words or mathematical expressions that appear repeatedly in the document. The best match equation is selected similarly at indexing time, and displayed to the user via a mouseover area. The best match equation is displayed separately, because it may not appear in the synopsis, since in general the best match equation and surrounding text will not be a good choice for conveying what the document is about to the user. If there are multiple equations with the same 'best match' rank, only the first one is displayed currently.

Subject to copyright restrictions on the original document, users may also be able to view a cached copy of a document in the result set. In cache documents, all matching mathematical expressions are highlighted. This is accomplished by

adding JavaScript code to the cache version of the document that searches the document content for expressions matching the query terms. Cache content is prepared in multiple versions, using images and MathML for the mathematics. Content negotiation is used to send highest functionality version supported by the user's browser.

At the time of this writing, Mathdex currently indexes around 25,000 documents from the arXiv[9], 12,000 pages from Wikipedia containing mathematics, approximately 1300 pages from Connexions[8], and around 1000 pages of Wolfram MathWorld[18]. We plan to expand the volume of content indexed significantly in the future.

References

1. Apache Foundation: Lucene Project, `http://lucene.apache.org`
2. Apache Foundation: Nutch Project, `http://lucene.apache.org/nutch`
3. Asperti, A., Guidi, F., Coen, C.S., Tassi, E., Zacchiroli, S.: A Content Based Mathematical Search Engine. In: Filliâtre, J.-C., Paulin-Mohring, C., Werner, B. (eds.) TYPES 2004. LNCS, vol. 3839, pp. 17–32. Springer, Heidelberg (2006)
4. Asperti, A., Selmi, M.: Efficient Retrieval of Mathematical Statements. In: Asperti, A., Bancerek, G., Trybulec, A. (eds.) MKM 2004. LNCS, vol. 3119, pp. 17–31. Springer, Heidelberg (2004)
5. Grzegorz, B.: Information Retrieval and Rendering with MML Query. In: Borwein, J.M., Farmer, W.M. (eds.) MKM 2006. LNCS (LNAI), vol. 4108, pp. 266–279. Springer, Heidelberg (2006)
6. Bancerek, G., Rudniki, P.: Information Retrieval in MML. In: Asperti, A., Buchberger, B., Davenport, J.H. (eds.) MKM 2003. LNCS, vol. 2594, pp. 119–132. Springer, Heidelberg (2003)
7. Cairns, P.: Informalising Formal Mathematics: searching the mizar library with Latent Semantics. In: Asperti, A., Bancerek, G., Trybulec, A. (eds.) MKM 2004. LNCS, vol. 3119, pp. 17–31. Springer, Heidelberg (2004)
8. Braniuk, R. et al.: Connexions, `http://cnx.org`
9. Cornell University Library: The arXiv, `http://arxiv.org`
10. Design Science, Mathdex, `http://www.mathdex.com`
11. Harvey, D.: blahtex, `http://www.blahtex.org/`
12. Miller, B.R., Youssef, A.: Technical Aspects of the Digital Library of Mathematical Functions. In: Annals of Mathematics and Artificial Intelligence, vol. 38(1-3), pp. 121–136. Springer, Netherlands (2003)
13. Miller, B.: DLMF, LaTeXML and some lessons learned. In: The Evolution of Mathematical Communication in the Age of Digital Libraries, IMA "Hot Topic" Workshop (2006) `http://www.ima.umn.edu/2006-2007/SW12.8-9.06/abstracts.html#Miller-Bruce`
14. Ogilvie, P., Callan, J.: Using Language models for flat text queries in XML retrieval. In: Proceedings of INEX 2003, pp. 12–18 (2003)
15. Tetsuya, S.: Average Gain Ratio: A Simple Retrieval Performance Measure for Evaluation with Multiple Relevance Levels, ACM SIGIR (2003)
16. Salton, G., Fox, E., Wu, H.: Extended Boolean Information Retrieval. Communication of the ACM 26(11), 1022–1036 (1983)
17. Trott, M.: Trott's Corner Mathematical Searching of The Wolfram Functions Site. The Mathematica Journal 9(4), 713–726 (2005)
18. Weisstein, E.: Wolfram MathWorld, `http://mathworld.wolfram.com`