Charles University in Prague

Faculty of Mathematics and Physics

DOCTORAL THESIS

Jozef Mišutka

Mathematical Search Engine

Department of Software Engineering

Supervisor of the doctoral thesis:	RNDr. Leo Galamboš, Ph.D.
Study programme:	Computer Science
Specialization:	Software Systems

Prague 2013

Acknowledgments

I am forever beholden to my wife Jana for her infinite patience and support. I am also grateful to my daughter Johanka Zoe who gave me the strength to finish this thesis whenever she smiled at me.

I am also indebted to my mother, father and my sister who have helped me throughout my academic years. All of them had to endure my absence in our family life.

I am also grateful to my supervisor Leo Galamboš for his sober support.

I declare that I carried out this doctoral thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In Prague on the 21st May 2013 Author's signature

Title	Mathematical Search Engine
Author	RNDr. Jozef Mišutka `misutka@ksi.mff.cuni.cz`
Department	Department of Software Engineering Faculty of Mathematics and Physics Charles University in Prague
Supervisor	RNDr. Leo Galamboš, Ph.D. `lg@hq.egothor.org`
Mailing address	Department of Distributed and Dependable Systems Charles University in Prague Malostranské náměstí 25 118 00 Prague, Czech Republic

Abstract

Mathematics has been used to describe phenomena and problems in many research fields for centuries. The basic elements used in the description are formulae which express information symbolically. However, searching for mathematical knowledge in digital form using available tools is still cumbersome. We address this issue by presenting the mathematical search engine EgoMath, based on a full text searching, which can search for mathematical formulae and text. We perform an evaluation over a large collection of documents showing that our solution is usable. Our approach can be used with huge document collections by applying one specialised technique. In order to provide a valuable evaluation of the quality, we built an alternative mathematical search engine using the feature extraction technique proposed by Ma et al. We propose important improvements to this solution achieving interesting results. We perform the first ever cross-evaluation of mathematical search engines based on different algorithms. A comprehensive survey of existing techniques available, presented in this thesis, completes the picture of mathematical searching.

Keywords

Mathematical search engine, Searching in mathematics, Mathematical formulae, Full text search engine

Název práce	Matematický vyhledávač
Autor	RNDr. Jozef Mišutka misutka@ksi.mff.cuni.cz
Katedra	Katedra softwarového inženýrství Matematicko-fyzikální fakulta Univerzita Karlova v Praze
Vedoucí disertační práce	RNDr. Leo Galamboš, Ph.D. lg@hq.egothor.org
Adresa	Katedra distribuovaných a spolehlivých systémů Univerzita Karlova v Praze Malostranské náměstí 25 118 00 Praha

Abstrakt

Po celá staletí se matematika využívá k popisu jevů a problémů v mnoha oblastech výzkumu. Vzorce jsou základními kameny v jazyce matematiky, ale i přesto je hledání matematických vzorců v digitální podobě stále těžkopádné. Tato dizertační práce navrhuje řešení a představuje matematický vyhledávač EgoMath, založený na fulltextovém vyhledávání. Praktická použitelnost je potvrzena testy na velké sbírce dokumentů. Také představíme techniku, díky které může být náš přístup použit na mnohonásobně větší kolekci dat. Aby bylo možné poskytnout cenné hodnocení kvality, vybudovali jsme alternativní matematický vyhledávač založený na práci Ma et al. Příslušná rozšíření umožňují dosažení zajímavých výsledků. Jako první provedeme srovnání dvou matematických vyhledávačů, které jsou postaveny na různých algoritmech. Vyčerpávající přehled stávajících používaných technik doplní obraz stavu výzkumu v oblasti matematického vyhledávání.

Klíčová slova

Matematický vyhledávač, Vyhledávaní v matematice, Matematické vzorce, Textový vyhledávač

Table of Contents

CHAPTER 1

Introduction

Mathematics has been used to describe phenomena and problems in many research fields for centuries. The basic elements used in the description are formulae which express information symbolically. In the last few decades, many new scientific fields have emerged and many old ones have flourished. The incredible amount of information and knowledge we have produced is often available in digital format on the Internet. To be able to use this information, methods to locate the relevant pieces of information must be available.

Retrieving information or knowledge from a collection of resources is examined by Information Retrieval (IR). However, the search for mathematical formulae and knowledge in real-world documents has only recently became active. Several basic building blocks are still missing or are not accepted in this research field, which is evident for example, in the evaluation process.

Unfortunately, mathematical knowledge is mostly encoded without accurate semantic information. The boom of semantic web and related technologies a few years ago seems to have changed only very little. There are several techniques which rely on the strictly defined semantics in curated datasets, but these are not directly applicable. Furthermore, there are only few technologies which are able to cope with the vast amount of data on the Internet. The well known paradigm of full text search engines being one of them.

1.1 Research Goals

The research goals of this thesis can be divided into three distinct parts.

G1 The first goal is to provide a comprehensive list of proposed techniques and available solutions with a brief description of the theory behind them. This has not been done before and should make it easier to orientate in this research field.

G2 The second goal of this thesis is to propose an approach to mathematical searching which can be used on the Internet and is mature enough for production use. This approach should be well scalable for extensive collections of documents.

G3 We need proper evaluation to be able to claim that our approach is feasible and mature. The third goal of this thesis is to provide such evaluation.

1.2 Overview of Contribution

In this thesis we present the mathematical search engine EgoMath capable of searching for mathematical knowledge and formulae in real-world digital documents.

Because the research field of Mathematical Search Engines is still young, we had to create several principal parts of an otherwise established research field in order to include the required parts of a research paper in our thesis.

A comprehensive survey of the mathematical searching research field was compiled in order to include a proper domain analysis. The work on the survey fulfils the first goal (**G1**) we set up.

Building EgoMath has required the building of several different sub-projects, including a parallel framework for resource import and a search engine for different mathematical format conversions. We reach the second goal (**G2**) by including a detailed description of the algorithms behind EgoMath, describing performance measurements over large document sets. We include a description of issues encountered during indexing large collections of documents and the solution of these issues.

In order to meet the last goal (**G3**), we had to implement another mathematical search engine. Therefore, the contribution is twofold. Firstly, we reimplement a feature based alternative solution to mathematical searching by Ma et al. [Ma+10a] and propose several key improvements, making the solution both suitable for larger collections of data and usable in general. We show that the feature based approach to mathematical searching, with our improvements, is really feasible by evaluating it on a large document collection. Secondly, we compare EgoMath to several versions of the feature based approach, verifying that EgoMath is suitable for mathematical searching.

1.3 Thesis Organisation

A brief overview of the theory on searching applicable to mathematical search engines is presented in Chapter 2. The theory should help to better understand the comprehensive survey on mathematical searching presented in Chapter 3. Chapter 4 contains a description of the mathematical search engine EgoMath and the associative subprojects. This chapter also includes the various evaluations. An alternative approach to mathematical searching developed for cross-evaluation purposes is presented in Chapter 5. Finally, Chapter 6 summarises the thesis, its contribution and gives an outlook to future research based on the ideas and methods presented in this thesis.

CHAPTER 2

Brief Theory on Searching

Searching is an integral part of information retrieval which deals with the representation, storage, organisation and access to information. The idea of searching for information using computers dates back to the 1950s and an immense number of publications has been written on this subject since [Ma+08, BaRi11, CaCe04]. In the following text we focus on models, structures and approaches which are, or can be, applicable to mathematical searching. However, we do not dare claim that this will be a comprehensive listing.

The key role of mathematical searching is mathematical formulae . In contrast to simple words or other objects which are commonly searched for, mathematical formulae can (theoretically) have a well defined set of properties, relations, applications and often also a "result". The concept of Information versus Data Retrieval[1] has a new dimension as there are many (mathematically) equal formulae which are textually significantly different. It is quite fundamental to ask what information does a user want when he looks for "$a + 7 = 15$". Is it the value of a, all indexed objects which contain this particular formula, or all indexed objects containing $15 = b + 7$? There is clearly no correct answer. In the following text, we assume that even if we only search for mathematically equal formulae, we are in fact using similarity searching where the similarity function would be defined (if it was possible) as a mathematical equality.

One of the main goals of the mathematical search engine (MSE) EgoMath present-

[1] "Data retrieval, in the context of an IR system, consists mainly of determining which documents of a collection contain the keywords in the user query which, most frequently, is not enough to satisfy the user information need." More details can be found in the book by Baeza-Yates and Ribeiro-Neto [BaRi11].

ed in this thesis is to allow for search in digital mathematical content. We claim that "there is no large mathematical data set including precise semantic information of mathematical objects publicly available". Even if there were, the contents would not be up-to-date, simply because by far the majority of researchers do not include explicit semantic information for mathematical formulae in their papers. Therefore, the first assumption we make is on the quality of input data. We will not cover approaches relying on curated, mathematically precise data. This subject is addressed by specific research fields including theorem provers and proof checkers. Nevertheless, we will mention approaches already used in mathematical searching taken from this research field.

The second assumption we make is that searching should be an interactive and iterative process where we find and learn. The answer should not necessarily change the question but it should give means to slightly reformulate or refine the question. The answer machines in mathematical search engines are also out of the scope of this thesis.

We will look at the process from the other way around, namely, what needs to be done for a user to be allowed to enter a query. In the first section, we briefly describe some of the applicable indexing structures which can be used in MSEs. Indexing structure allows the returning of relevant objects, often in no particular order. The next section will focus on ranking the results, in other words, measuring (dis)similarity between the query and the relevant objects. In the last section we will very briefly mention the methods used mainly to improve the quality when there are many results by finding new or by exploiting explicit relations in the data set.

2.1 Indexing Structures

Indexing consists of parsing and storing data to facilitate IR. It is also performed to improve the search speed over a simple sequential search. A well-known and probably the most used indexing structure is the inverted index. An inverted index is a word[2]-oriented structure which gained popularity because it provides very fast access to large data sets if the distribution of occurrences is not extreme. The cost of fast access is the restriction put on the properties of the similarity function. The advantage of very fast query processing by an inverted index is possible due to the usage of spe-

[2] A word is simply a group of characters in this context.

cific similarity functions e.g., the cosine similarity. The lists belonging to non-query words can be skipped because their weights in the query are zero, while the for example cosine measure applies multiplication of weights (which would lead to zero for any weight in the list). Inverted index is the most used also in mathematical searching (see the survey in Chapter 3) and can be used after mathematical formulae are linearised or for systems which work over textual features extracted from them. It allows for the retrieval of objects that have specific words, phrases, words within a specified proximity. Specific functionality e.g., wild-cards can be also used but often with performance penalty. More queries can be connected using boolean operators to form one complex query.

Another structure which could be used for indexing mathematical formulae are signature files. A signature (or hash) represents the indexed item with a constant number of bits mostly smaller than the indexed item. More formulae could have the same signatures and the false positives must be discarded in a sequential pass. The hashing function can be based on features extracted from every formula. However, in most applications it is inferior to the inverted index [Zo+98].

Many different indexing techniques used in automated reasoning systems are very nicely described by Graf [Gra96]. A substitution tree indexing structure, introduced by Graf [Gra96], subsumes several other tree-based structures and is also used by a few mathematical search engines. If substitutional trees are used to index mathematical formulae, the result is a tree-like structure with nodes containing substitutions of its parents. The indexed formula is reconstructed from a root node by applying one or more substitutions. This indexing structure allows unifiable, generalisations, instances and variants retrieval. Other types of tree based indexing structures from automated reasoning include path indexing [Sti89], code trees [RiVo00] and context trees [Ga+04].

The standard mathematical formats (e.g., MathML and OpenMath) are XML based and therefore suitable for XML search engines. However, the mathematical semantics with regards to similar (equal) formulae cannot be indexed directly. XML indexing is part of the XML retrieval which is a significant research field and is out of the scope of this thesis.

There are specific indexing structures used when the similarity function is metric. However, these structures do not contribute directly to the indexing features and characteristics, but speed up the process of searching. For more details about similarity search indexing structures, read the book Similarity Search by Zezula et al. [Ze+06].

2.2 Relevance and Ranking

The notion of object relevance uses different premises to define the relevance function. Different sets of premises about object relevance yield different retrieval models. In the following text, we try to cover the basic information retrieval models. The description of the underlying mathematical models will help us understand and reason about the behaviour and theoretical applications.

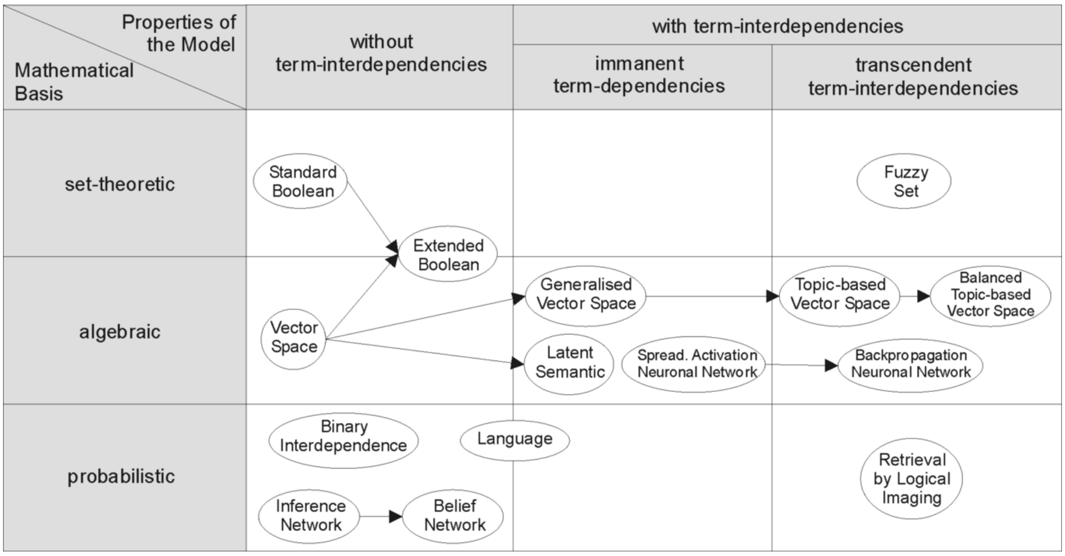


Figure 2.1: Taxonomy of information retrieval models

Objects in information retrieval are often transformed into suitable representations by appropriate models for the effective retrieving of relevant ones. Figure 2.1[3] illustrates the relationship of several common models categorised according to the mathematical basis and the properties of the model. Models where objects are modelled as sets are labelled set theoretic. Objects which are represented as n-dimensional vectors are modelled by algebraic models, and models which use probability theory as its basis are probabilistic. The traditional representatives of the first two are the standard boolean model and the vector space model (VSM). The taxonomy further divides the models to those without term-interdependencies, with immanent term interdependencies which are defined by the model itself and with transcendent term interdependencies which do not set any restriction on the definition.

[3] Translated image from Dominik Kuropka's book Modelle zur Repräsentation natürlichsprachlicher Dokumente [Kur12].

In several taxonomies, another model is mentioned, namely the feature-based model which models documents into vectors of features where a feature can be an arbitrary function. This model does not fit well into the classical categorisation because it can be used to model almost all the other models. These models do not use information or properties other than the input objects themselves.

Another taxonomy from a different point of view distinguishes: 1) boolean, 2) statistical and 3) linguistic and knowledge-based models. The second includes the VSM and the probabilistic retrieval models mentioned above. The first model is often referred to as the "exact match" model; the latter ones as "best match" models.

Let us assume that we are indexing documents and we are searching for more general information. Summarising the contents of documents and queries with only with a set of index words directly from the document can result in the missing of relevant documents which contain synonyms or slightly different words with the same meaning. This problem is often referred to as "the dependence problem" or "the vocabulary problem". Latent Semantic Analysis (LSA) addresses this issue by defining relevance, not by index words, but by automatically derived concepts.

There is no simple representation of mathematical formulae in the models mentioned above without pre-processing. Retaining part of the mathematical semantic information (either extracted or assumed) in textual information retrieval requires indexing more information than the original data. One possible solution is shown by Mišutka and Galamboš [Miš08b].

There are alternative models which could be exploited for mathematical searching. Mathematical formula can be represented as structured text. The structured text retrieval models, like the models based on non-overlapping lists [Bu+92] or models based on proximal nodes [Bae96], give some more power to search for this type of representation; however, the improvement in the quality of mathematical searching using this model is questionable. Another model which is interesting is the rule-based and logic one[4].

Another approach which is often mentioned are models for browsing and/or filtering. Though, not easily applicable to mathematical formulae directly, it can greatly improve the searching experience when browsing is accessible within the result set. This method is used by EgoMath and is described in Chapter 4. The idea is to explore the object space directly by iteratively refining the search.

[4] For more references see Canfore and Cerulo [CaCe04].

Several of these models which are used in similarity searching are described in more detail in the next section. A very good and pragmatic description of the models can be found in an outstanding textbook by Baeza-Yates and Ribeiro-Neto [BaRi11].

Similarity Searching

Even without proof, we can say that similarity searching has become the most important web application today. Similarity is the key principle in (re)cognition, with the cognitive science research field examining similarity in depth. There are many ways to perceive objects e.g., visually, auditory; moreover, perception can be influenced by memory and experience. The meaning of similarity does vary in applications and is strongly dependent on the description of the objects and the context. Because of this, it is very difficult to compare the similarity or dissimilarity functions and to create publicly available testing datasets. More on the general theory behind similarity perception, including an important additional bibliography can be found in an overview paper from 2012 by Zezula [Zez12].

Many users search for a particular object or information and the expected result should be an exact match. In reality, neither the description of an object, nor data stored about the object, nor the user description of the user needs is precise in most cases. This has been significantly emphasised by the growing complexity of objects users want to find. Therefore, the relation of objects is not described by exactness but by (dis)similarity.

In the case of mathematical searching, things get even worse. The perception of equality in mathematics is complex. According to our internal unpublished survey[5], most people thought that $1 + 2$, $2 + 1$ and 3 are the same objects, but on the other hand, most people regarded $cos^2(x) + sin^2(x)$ and 1 as two different objects. Modelling similarity (which is probably the case with most information retrieval systems) strongly depends on the background of the particular user. In the case of mathematical searching, which parts of mathematics (if any) is the user familiar with. Several of the important models are described in more detail below.

The Vector Space Model

The vector space algebraic model is the most successful and well known representation of documents with independent words. Several other important models are

[5] The survey was done with 20 people educated in natural sciences.

derived from the vector space model including the generalised vector space model which can handle word dependencies.

Weight is assigned to each word in an index. The weights are used to compute the degree of similarity between the objects and a notional object which represents the query. One of the advantages of this model is that it takes partially similar documents into account.

The similarity of the documents is used to rank the returned results (not really in fetching the result set itself). The best known similarity function by far is the cosine similarity function defined for a document d and a query q both represented by n dimensional vectors of weights dw and qw respectively as

$$sim(q,d) = \frac{\sum_{i=1}^{n} (dw_i * qw_i)}{\sqrt{\sum_{i=1}^{n} dw_i^2} * \sqrt{\sum_{i=1}^{n} qw_i^2}}. \tag{2.2.1}$$

An important and interesting question is how to define the weights. One example is the well known tf-idf (term frequency - inverse document frequency) weighting scheme [Sa+75] which uses a statistical model of the document set to compute out the importance of words.

The Metric Model

Metric models have gained popularity in the last several years. They allow for a range of new similarity measures. Many data collections meet the requirements needed for metric modelling and therefore, different models can be used and reused. The abstraction of object proximity into metric space is straight-forward and the well explored mathematical theory behind metric spaces gave similarity searching using metric models a big boost. From the metric space point of view, the similarity is defined as dissimilarity and can be imagined as the distance between objects in a coordinate system.

A metric space is defined as (M, d) where M is a set and d is a metric on M. Function d is defined as: $d : M \times M \rightarrow \mathbb{R}$ such that for any $x, y, z \in M$ these properties must hold: $d(x, y) = 0 \Leftrightarrow x = y$ (identity), $d(x, y) = d(y, x)$ (symmetry) and $d(x, z) \leq d(x, y) + d(y, z)$ (triangle inequality). The non-negativity property follows from the other properties. Because of the popularity of the metric models there is substantial effort to map similarity problems where the postulates above do not hold in the metric space.

A typical use case includes searching for multimedia e.g., images, videos and sounds where the vector space mode is insufficient in most cases. There are other use cases where the similarity function is more complicated e.g., biometric identification (comparing face shapes, fingerprints) and similar sequences in time series.

A common problem connected with similarity searching is the well-known dimensionality curse i.e., the solution uses too much space or the speed performance is similar to a simple sequential scan. To predict these situations, the definition of intrinsic dimensionality of metric datasets was introduced by Chávez et al. [Chá+01] (see Section 2.2.1). In many situations, approximate searching can be used to overcome (at least partially) the curse of dimensionality. The most important questions in similarity searching today can be transformed into well known mathematical problems i.e., similarity range and nearest neighbours. The theory and practice behind various data structures used for indexing and searching in metric models is outside the scope of this thesis. A thorough description of metric models was done by Zezula et al. [Ze+06]. We recommend one of the following papers [Chá+01, Ze+06, SkBe+11] for further reading.

The author believes metric similarity functions could be used to model the similarity of mathematical formulae. Most mathematical formulae can be intuitively represented by a tree and the majority of digital mathematics is encoded either in MathML-like (including OMDoc) or TEX-like (including ASCIIMathML) formats which are often parsed into tree-like structures. The metric space offers tree edit distance as one of its main representatives which could be exploited exactly for this purpose. There have been several attempts to do so but these ended up without really exploiting the offered advantages (see Section 3.1.1).

We briefly select depict selected well known metrics used to measure the (dis)similarity of different data structures i.e., vectors with independent components, vectors with interdependent components, trees, sets. These metrics have been either already used in several approaches to mathematical searching or could be used.

Minkowski Distances

Minkowski Distances are a family of metric functions labelled as L_p metrics where $p \geq 1$ defined for vectors $\vec{x}, \vec{y}$ as

$$L_p[\vec{x}, \vec{y}] = \sqrt[p]{\sum_{i=1}^{n} |x_i - y_i|^p}$$

with a few instances well known under different names e.g., L_1 as the Manhattan distance, L_2 the Euclidean distance and L_∞ is called the maximum distance. The time complexity is linear (dependent on the dimensionality n). The similarity function can be used for feature-based MSE.

Quadratic Form Distance

A metric which accounts for simple interdependent components is defined as

$$d_M(\vec{x}, \vec{y}) = \sqrt{(\vec{x} - \vec{y})^T * M * (\vec{x} - \vec{y})}$$

where $\vec{x}, \vec{y}$ are vectors, T is a matrix transposition and M defines how strong the correlation between components is. The definition is equal to the Euclidean distance if M is an identity matrix. A typical use case is similarity in simplified colour histograms, but again, it can be used for feature-based MSEs. The time complexity is quadratic.

Tree Edit Distance

The apparent successor to the edit distance metric is the tree edit distance. It is defined as the minimum cost needed to convert the source tree to the target tree using a predefined set of tree edit operations e.g., insert a node or delete a node. This metric can also be used to measure the (dis)similarity of XML documents. The time complexity of the original algorithm can be up to $O(n^4)$ where n is the number of tree nodes. Because mathematical formulae can be represented as trees, the tree edit distance can be directly used to measure the similarity.

Jaccard Distance

Another common data structure is a set. The Jaccard distance - trivially derived from the Jaccard similarity coefficient - for two sets A, B is defined as

$$d(A,B) = 1 - \frac{|A \cap B|}{|A \cup B|}$$

and is used as the similarity measure. A use case is finding similarity between user behaviour where the behaviour is modelled as a set of actions e.g., visited sites, hobbies or friends. One important modification of this metric, which can be used for vector data, is called the Tanimoto distance [RoTa60]. MSE extracting features could benefit from the advantage that the set of features does not need to be in a particular order.

Non-metric Model

A typical example of a problem with modelling similarity in metric space is the similarity of a man to a centaur, a centaur to a horse and then a man to a horse. While the first two are mostly perceived as similar, the latter is not similar and therefore it is breaking the triangle property of a metric function.

The more human oriented perception of similarity is provided by this model which is not limited by the required properties of the metric space. The loose requirements put on similarity functions cost the model performance. However, other properties hold in non-metric similarity functions and can be exploited to improve the performance. Moreover, domain experts are not burdened with apprehension of non-metric space and they usually define similarity that is not metric. Possible application domain, best practices and further details are described by Skopal and Bustos [SkBe+11].

To be precise, the vector space model with a cosine similarity function should be in this category but because of specific characteristics and history, it is often presented separately.

2.2.1 Indexability of Datasets

One way of looking at an algorithm is to precisely describe the properties and conditions with their consequences. Often, researches face the problem from the other way around. Given a dataset, what is the best way to do something with it? In the existing techniques, nearest neighbour and range queries have an exponential dependency on the dimension of the space which the objects are modelled into. This is called the curse of dimensionality. There have been several attempts to formalise this behaviour and we will describe one of them. A correct general concept of the intrinsic dimension of a dataset is not clear and probably never will be because of specific characteristics of various dimensions and the various purposes a dataset is used for. However, it is common to associate high values of dimension with the curse of dimensionality. The curse of dimensionality characterises a situation where object features are concentrated around particular points e.g., means. Many similarity-based information retrieval algorithm's performance on high-dimensional datasets is comparable to a simple sequential scan. This phenomenon is quite well-known but still there is no mathematical validation of it [Pes10]. We will quantify this meaning using one of several possible approaches called intrinsic dimensionality proposed by Chávez et al. [Chá+01]. We use this approach because it is easy to compute and can be easily understood. The structure of the document set can be disclosed by its distance distribution showing clusters of objects and how "near" they are.

Vector space may suffer from a difference between the representational dimension and their intrinsic dimension i.e., the real number of dimensions in which the points can be embedded while keeping the distances among them. The idea is similar to statistical analysis of data sets in databases for improving performance.

Given a document set D and a metric distance δ the intrinsic dimensionality is defined as

$$\rho(D, \delta) = \frac{\mu^2}{2\sigma^2}$$

where μ is the mean and σ^2 is the variance of distribution in D [Chá+01]. Informally, if the intrinsic dimensionality is high it means that the data set can not be partitioned well and that only few elements can be eliminated from the search process. Intuitive graphical representation can be found in the original paper.

More details of the general concept can be found in several papers [SkBe+11, Pes07, Pes10].

2.3 Clustering

Cluster based search can be utilised as a tool to improve retrieval performance for information retrieval tasks. It is based on the cluster hypothesis which claims that documents in the same cluster behave similarly with respect to the relevance of information needs. To the best of the author's knowledge, there is no evaluation of clustering over a significant number of mathematical documents. However, it can be assumed that scientific documents could be grouped into clusters of documents addressing similar problems. Clustering can be used at various levels during search. The most intuitive is clustering of the search results and/or of the whole data set. Instead of a sorted list, similar documents are grouped. This can be valuable if the query is ambiguous and has several different meanings. In theory, searching for integral could cluster the results based on the type of integral e.g., Riemann, Newton or Lebesgue. The clusters can either be statically or dynamically computed. Another use case can be language modelling, where instead of using the language model, derived from the whole collection, a specific model for the document's cluster is used to estimate the occurrence probabilities more precisely e.g., the tf-idf data set statistics. If the input data can be reasonably clustered then this approach can be used to speed up searching. Instead of computing the similarity to every object, only objects from specific clusters are taken into account.

Clustering can either be flat or hierarchical. Flat clustering is efficient and conceptually simple. Hierarchical clustering outputs a hierarchy, a structure that is more informative than the unstructured set of clusters returned by flat clustering. More information about clustering algorithms can be found in Tan et al. [Th+05] and in Xu and Wunsch [XuWu08]. Clustering is connected with models used for browsing and/or filtering mentioned above.

2.4 Query

The domain of questions which can be asked depends on the indexing structures and the retrieval models being used. On the other hand, the query language has to work with the *retrieval unit* which defines the smallest element that can be retrieved. User information need is expressed by a query.

The simplest query is composed of keywords and the expected outputs are objects containing the keywords. This type of query is intuitive and allows for a very fast retrieval. The crucial problem is finding the correct unambiguous keywords. Another type of keyword query takes context into account by allowing phrase and proximity searches[6]. Boolean logic can be exploited with boolean queries. Boolean and context queries are the basic building blocks of queries in EgoMath and are discussed in more

[6] Proximity search looks for objects where the query word's positions are within a specified distance.

detail in Chapter 4. Pattern matching is usually based on regular expressions[7] which can specify a set of syntactic features that must occur in a text segment in order to mark it relevant.

The basic structural query types like, overlapped lists and proximal nodes have only limited expression power. On the other hand, more complex structural models like the tree matching model involves several NP-Complete problems [KiHe93]. State-of-the-art structural retrieval is mainly XML retrieval. The requirements for structured retrieval query are content constraints, pattern matching constraints and structural constraints. XML query languages can be classified as content-only (user does not know the document structure) and content-and-structure query languages [BaRi11]. Content-and-structure query further categorises into tag-based, path-based and clause-based languages. For mathematical search engines, the clause based languages like XQuery [Ka+04] could be of value.

A more detailed description of single word, context, boolean, natural language queries, pattern matching and structural queries can be found in Baeza-Yates and Ribeiro-Neto [BaRi11].

[7] A regular expression is a specific pattern that provides concise and flexible means to "match" (specify and recognise) strings of text, such as particular characters, words, or patterns of characters.

CHAPTER 3

Domain Analysis

Most scientific papers are published electronically and great efforts have been made to digitise mathematical knowledge in the last few years. The result is that the number of scientific documents is growing rapidly and the necessity to search these documents has increased significantly. This happened only a few years ago and we can still consider the mathematical searching research field as one of the younger ones.

There have been several surveys of the mathematical searching research field published in journals [MuKh11, ZaBl12] but sadly none of them were really comprehensive. One of the additional contributions of this thesis is to fill this gap with a comprehensive survey in the next section. We cover the development in mathematical information retrieval focusing on search engines till 2013.

Mathematical searching and searching in mathematical documents is an interdisciplinary research field. It has been our experience that researchers working in this field are influenced by their research background without explicitly mentioning it. At the end of this chapter a list of elementary but often unanswered questions is compiled. The importance of these rather simple questions has been emphasised while working on the following survey. Several newer approaches would benefit in answering them giving strong reasoning instead of using strong claims.

Let us first create a basic taxonomy of mathematical search engines that will be used in this chapter. Different taxonomies applicable for generic information retrieval systems could be used to categorise mathematical information retrieval systems but we will not explicitly repeat these. The input for a mathematical search engine must be a mathematical formula in one of many possible formats. The important part is how the formula is being processed and queried. We define high level taxonomy as follows:

1. syntactically mathematically aware - search engines which enable search in mathematical documents but do not allow to (at least) approximately describe a mathematical structure where the particular formula lives. By "approximately describe" we mean that a certain level[1] of mathematical operations must be available specifically for this structure.

2. semantically mathematically aware - search engines which enable search for mathematical formulae inside (at least) a partially defined mathematical structure with a certain level of mathematical operations.

3. strict mathematical model - used by automatic theorem provers and precise computational software where the mathematical model is precisely specified and adhered to even if the query language can introduce limitations in its expressiveness.

We must distinguish between mathematically correct solutions which use curated input data sets and solutions which can use semantics but the semantics is guessed or deduced. In the following text, we consider semantically mathematically aware search engines and also those which somehow guess the semantics.

3.1 Searching for Mathematics

The following section describes the development of this research field and existing approaches. We have included papers with new contributions and have eliminated duplicates and simple summaries. We briefly describe the set up used for performance evaluation (if available) but no conclusions have been made. This is because the vast majority of the evaluation is not really a performance evaluation but rather a proof of concept.

We created a graph showing search term popularity connected with mathematical searching in Appendix A.1. There are several interesting points about the graph. If the statistics[2] used to create this graph are correct then the first notable characteristic is a

[1] It is not necessary to precisely define the term "certain" at the moment of writing this thesis because every known mathematical search engine can be easily classified into one of the mentioned categories.

[2] The data is maintained by Google, Inc.

decrease in interest in mathematical searching around July every year. The second notable characteristic is the decline in popularity of "math search" observable from 2010. The interpretation is highly subjective but the authors would interpret it as follows. The decrease every year might be connected with academic summer holidays. The decline from 2010 could be due to the fact that WolframAlpha[3] was launched with a lot of attention. WolframAlpha can handle several types of mathematical queries very well and the need to search for another place which offers mathematical searching was minimal for these types of queries. We can draw basic conclusions about the audience who are interested in mathematical searching.

We do not describe graphical user interfaces (GUIs) of the mathematical search systems in the following survey because there are plentiful other mathematical systems with interesting GUIs available which are outside the scope of this thesis. There are only a few mathematical search engines publicly available so we include several screenshots in Appendix A.

Several key terms have been used differently in different papers; therefore, the meaning is dependent on the context where it occurs e.g., normalisation or normal form means renaming variables [Gra96] in comparison to ordering of parse tree nodes [MiYo03] and semantic normalisation [Miš08b].

Many of the works have been presented at one of the conferences related to the MKM interest group[4]. However, in the last few years several interesting applications have been developed; also outside of this principal research community.

In the next section, we do not explicitly refer to ourselves and the main component of this thesis - EgoMath - to comply with the style of an objective survey.

3.1.1 A Survey

The first non-trivial approach to searching in mathematical structures was published in Searching Techniques for Integral Tables by T. H. Einwohner and Richard J. Fateman in 1995 [EiFa95]. The paper describes textual features extracted from formulae used for searching including a basic sub-formula search. Searching is limited and specialised to integral tables.

A few other papers had briefly mentioned formula search before 2004. These include the Archon Digital library description [Ma+02] which allows for the retrieval of

[3] WolframAlpha Computational Knowledge Engine, available at `http://www.wolframalpha.com/`.

[4] Mathematical Knowledge Management, more details at `http://www.mkm-ig.org`.

equal LaTeX strings from a relational database. The database also contained digital images of formulae which were displayed to the user in the result list. Peter Graf's book [Gra96] and later an updated version by Sekar et al. [Se+01] are related to mathematical searching in general. Both contain an up-to-date overview of indexing techniques and describe building indexes for searching in great detail focusing on specific areas like automated deduction and symbolic computing. Several topics mentioned in the book including path indexing, substitution trees and unification factoring are used in several mathematical search engines.

The first important, coherent and publicly available reference of mathematical searching and searching in mathematical texts was mentioned at the IMA Workshop Enhancing the Searching of Mathematics in 2004 [Ima04] at the University of Minnesota. The conference was part of the NSF award [Nsf02a] Indexing, Searching&Retrieval of Mathematical Contents started in 2002 with Abdou Youssef as the principal investigator. Important points connected with mathematical searching were mentioned in Robert Miner's [Mil04] and Youssef's [You04] talks and papers [MiYo03] e.g., the importance of MathML, TeX in mathematics, important complementary text search to mathematical searching and creating a document collection for usability testing. Most of the discussion was done on the theoretical level. However, several important technical issues were already addressed by Miller and Youssef in DLMF[5] (see Figure B.1 in Appendix) namely textualisation and flattening of mathematical formulae (x^2 into "x begin_superscript 2 end_superscript"), sub-formula searching using the proximity search, thesaurus abbreviations for formula and normalisation with basic textual ordering. The mathematical search was in the ability to parse TeX and MathML and in the description of simple normalisation. Apart from this, a search for mathematically equal formulae or a search for variables could not be performed.

2005

In this year, Youssef continues his work and describes the effort with basic performance evaluation on 2000 equations from 300 mathematical documents [You05]. Youssef lists several major mathematical search issues he identified: 1) defining an intuitive yet expressive mathematical query language, 2) bridging the query language with the language of the content files, 3) making IR systems "understand" mathematical symbols and structures and 4) highlighting matched equations.

[5] NIST Digital Library of Mathematical Functions, available at `http://dlmf.nist.gov`.

The Wolfram Function Site hosted about $90,000$ mathematical formulae at this time. Trott underlines the importance of mathematical searching on this site in the The Mathematica Journal, 2005 [Tro05]. He argues that despite the clear organization of the formulae in their document set (The Wolfram Functions site could be viewed as a large table, with more than 250 functions running horizontally and more than 36 properties running vertically) many formulae can be written in different forms and classified in different ways. Furthermore, some identities might be given in a more general form than needed for a concrete purpose. A search engine should be used for quick and convenient access to the vast amount of knowledge encoded in the identities. The search UI sent a canonical form of the query to the backend starting a Mathematica[6] session. The search process uses a hash table which contains features extracted from every formula. There are four feature types: functions, constants, numbers and operations. This correlates with the UI where a user can select whether he wants a feature or not. It resembles a simple inverted index. There are several very specific operations related to Mathematica. The result list contained images of the found formulae.

Mišutka started to work on a full text math-aware mathematical search engine EgoMath in 2005 [Miš05] which was finalised into his diploma thesis [Miš07]. This work was the foundation of this thesis.

2006

In another paper from Youssef [You06] the motivation and the needs of a math-aware search system in general, including the fields of applicability is summarised. Youssef updates the basic objectives of mathematical search to: math-awareness, a natural math-query language, fine granularity of searchable and retrievable information units, perfect recall, perfect precision, perfect relevance-ranking, useful highlighting and minimum hit-redundancy. He identified these fields of application of mathematical searching: discovery of similarities between fields, computer-aided proving, learning aid and routing (routing was defined in this paper as the process of informing users of the latest information that match a pre-determined query specified by the user, as soon as the information becomes available). This was the first time that the similarity search research field was mentioned in connection with mathematical searching. An attempt to briefly describe a theoretical similarity function was presented.

In another paper from 2006 [YoSh06] multiple rules are used to pre-process every

[6] Mathematica is a computational software program.

formula to get one normalised representation. The description is solely theoretical and does not go into detail. These papers identify several of the objectives and areas of applicability similar to Mišutka [Miš07] but are in strong contrast to several others (compare to the description of [Miš07]).

A new mathematical search engine MathWebSearch (see Figure B.2 in Appendix) was presented by Michael Kohlhase and Ioan A. Şucan [KoŞu06] which does not use a full text search engine but uses the substitution tree indexing structure introduced by Graf [Gra96]. Kohlhase and Şucan built their search engine on these observations: 1) mathematical notation is context-dependent, 2) identical presentations can stand for multiple distinct mathematical objects and 3) certain variations of notations are widely considered irrelevant. To cope with the first two observations, the authors concentrate on the content representations of mathematical formulae. The third one is addressed by the choice of the substitution trees as the indexing structure. An extensible XML-based query language was developed for their search engine. The biggest advantage is that it can find unified formulae using complex substitutions. However, the substitution tree indexing can neither search for sub-formulae nor any other available data (e.g., surrounding text), the whole index should be in a fast memory, for the substitutions to work correctly the indexed formulae must be defined in appropriate mathematical models. The search for sub-formulae is addressed by adding all sub-formulae to the substitution tree together with the parent expression. They claim that this leads to a manageable increase in the index size, because many sub-formulae are shared by the larger expressions. Their evaluation was done on $87,000$ equations resulting in the 770MB index.

Another project with interest in mathematical searching was the learning environment for mathematics called LeActiveMath (see Figure B.3 in Appendix). The search functionality has several similar points to the work of Youssef and Miner. Paul Librecht and Erica Melis describe [LiMe06a, LiMe06b] how to index and search for mathematical formulae with a full text search engine by flattening (similar to textualisation) XML tokens extracted from the formulae. They focus on the mathematical content items and formulae as opposed to the function orientation of Youssef and Miner. Ranking is improved by boosting the score of specific queries. This is possible because LeActiveMath relies on an XML-like markup language for mathematical formulae called OpenMath. LeActiveMath is learner-oriented and relies on provided metadata. Performance testing was done on their private collection set encoded in OpenMath counting $2,761$ documents with $36,389$ formulae. The implementation uses the Apache Lucene search engine[7]. For the first time a survey on user behaviour connected with mathematical searching was published even if only marginal.

2007

Mišutka presents a full text mathematical search engine which can search for mathematical formulae in predefined mathematical structures defined by axiomatic rules. He presents a new technique of searching for mathematical formulae in real-world mathematical documents but still offering an extensible level of mathematical awareness. The work is built on the claim that most of these documents do not contain semantic information; therefore, precise mathematical interpretation is impossible. A proof of concept implementation is available and was evaluated on two different datasets containing $32,306$ and $1,416$ formulae respectively. The problems were summarised into these points: 1) no commonly used mathematical format or unitary notation, 2) symbol meaning dependent on context, 3) structured text, 4) no canonical form, 5) equivalent transformation rules and 6) many mathematical structures with different axioms. The main idea is a chain of transformation rules producing different representations (augmentation) which are indexed as synonyms for the original representations. Each following representation is a generalisation of the previous one. The paper contains a comprehensive summary of technologies which are somehow related to mathematical search i.e., Citeseer[8]-like, web directories, XML search engines, theorem provers and proof checkers, semantic web and Encyclopaedia of Integer Sequences. At the end, a brief description of available search engines was presented[9]. Several other important issues have been touched upon e.g., the ordering algorithm, linearisation, formula tokenisation and sub-formula search.

Altamimi and Youssef present their views [AlYo07] on the role of wildcard searching in mathematical search engines which are based on XML. They developed a new query language that extends the current standard XML query syntax. They claim to present a mathematical query language that enables general users to express their information needs intuitively yet precisely. The presented query language is mapped

[7] Apache Lucene Core, available at `http://lucene.apache.org/core/`.

[8] Citeseer was one of the first academic paper search engines focusing on citations.

[9] These engines were added at the end, after the work was done, because none of them was available during the actual work on the EgoMath mathematical search engine.

into standard XPath[10]/XQuery[11] queries.

According to Youssef, the standard text similarity functions turned out to be inadequate in mathematical search [You07]. One possible similarity (ranking) function taking into account additional information (e.g., type of formula, weight of individual terms) is described in the paper in detail. To be strictly precise and correct, this particular definition of the similarity function can even assign positive values to objects whose keyword intersection is zero and therefore, the full text search engine paradigm cannot be used here. Because of this the similarity function should also be used for retrieving, which means that all elements would have to be traversed. Youssef is also claiming that although the snippets containing the hit title accompanied by a few leading sentences from the target document are simple to produce, these often fail to convey the document's relevant excerpts to the user. He addresses this issue by adding sub-hits for each hit into the result list.

An approach to mathematical searching through query formulation and data normalisation is described by Miner and Munavalli [MiMu07]. The main idea is based on the work of Youssef [MiYo03] but the normalisation process is extended with a multipass algorithm. Encoding errors are corrected in successive stages, character data is canonicalised and heuristics are applied to disambiguated expressions followed by the MathML tree structure canonicalisation and finally canonicalisation of mathematical synonyms and some variable names.

Hijikata et al. describe an approach to mathematical searching also exploiting XPath [Hi+07]. They support formulae in MathML format. The structure of each formula is used to generate all the possible XPath expressions using DOM[12]. Two of these expressions are selected (the first and the deepest path) and concatenated to form a textual word characterising the formula. This word can then be indexed (e.g., by a full text search engine) and searched, returning all the formulae with the same characteristic word. Neither the details of building such a similarity function nor the implications are mentioned. The evaluation was done on $87,000$ formulae but inspecting only two search queries.

Kohlhase continues his work in this research field with Normann by stressing the importance of pre-processing [NoKo07]. Their motivation comes from the field of

[10] XPath, the XML Path Language, is a query language for selecting nodes from an XML document.

[11] XQuery is a query and functional programming language that is designed to query collections of XML data.

[12] Document Object Model (DOM) is a cross-platform and language-independent convention for representing and interacting with objects in XML-like documents.

automated reasoning where they propose a normalisation approach to support limited ϵ-retrieval[13].

Because of the indexing structures used by MathWebSearch it is problematic to directly integrate the textual search. An indirect solution was outlined by Anca [Anc07]. The matched documents returned from MathWebSearch are joined with results returned from a full text search engine based on Apache Lucene. A similarity (ranking) function is proposed which ranks documents from the intersection higher than the other returned documents. Because the similarity functions of the textual and mathematical search are independent they cannot use the additional information from the surrounding of the formulae in the text. The final similarity function consists of the similarity functions connected with specific weights.

In another paper, Andrea and Michael Kohlhase expressed that the community behind formalising mathematics and mathematics in computer science in general does not necessarily focus on the real needs of the user [KoKo07]. Several important questions were raised about the convictions of mathematical search engines authors and the expectations put on the users and data providers of such systems which follow from the convictions. Some of the consequences of the design decisions including the non-trivial ones are summarised.

2008

Mišutka summarises his work in this year focusing on the applicability of the mathematical search engine EgoMath (see Figure B.4 in Appendix) trying to make it production ready [Miš08b]. More experiments shown that this technique can be used to build a fully-fledged mathematical search engine focusing on large amounts of data. The atomic information (grain) showed itself to be very important not only for subformula searching but also for performance. A new document set was created from a subset of the arXiv [Arch] collection for evaluation purposes containing $852,388$ mathematical formulae.

The PhD thesis of Norman [Nor08] gives an up-to-date overview of theorem provers. The renaming problem (α-equivalence[14]) and formula equality problem (ϵ-retrieval) for formalised libraries described in the thesis is strongly connected with mathemat-

[13] ϵ-retrieval is the task of retrieving equal formulae in respect to a theory containing logical equivalences.

[14] Two formulae are α-equivalent when we can transform one of the formulae by renaming (bound) variables into the other without changing semantics.

ical searching. This is proved by the fact that several of the proposed partial solutions are almost equal to the steps in Mišutka's work on the generalisation of formulae [Miš08b].

A regular expression based feature extraction from MathML is described as the basic element of a search engine by Adeel et al. [Ad+08] called Math GO!. A dictionary of regular expressions and mapped keywords is used to extract appropriate keywords from each formula if the regular expressions match. Then these keywords can be searched in a traditional text based information retrieval system. Furthermore, they applied basic clustering to query formulae and the result set. The returned results are only from the most similar cluster. The performance testing was performed on $1,400$ equations.

The technical decisions about a mathematical retrieval system based on interviews with 13 participants were published by Zhao et al. [Zh+08]. The surprising outcome is the suggestion that the math-aware searching is not so much desired and users would rather benefit from linking formulae to keywords. They also promoted resource categorisation and used their Support Vector Machine (SVM)-based resource categorisation prototype implementation as a case study.

The search engine at DLMF with few enhancements was described as a work in progress by Miller and Youssef [MiYo08] including fine-grained highlighting of search terms.

2009

One possible similarity function for mathematical formulae is defined by Kamali and Tompa [KaTo09]. Their algorithm includes normalisation similar to already presented ones. Despite the formal definition, it does not describe the mathematical properties of the similarity function e.g., whether the similarity function is a metric or mathematically equal formulae are always more similar than unequal ones. Moreover, neither performance evaluation nor implementation is available. The key element in the similarity function is finding common subtrees which can be time consuming depending on additional characteristics of the problem. They extracted $4,000$ mathematical formulae from Wikipedia and Wolfram and showing several characteristics[15].

Ştefan Anca continues his involvement in this research field by addressing information mining from mathematical documents [Anc09]. The understanding of the se-

[15] Compare to comprehensive characteristics from this thesis in Figure 4.4 in Section 4.3.

mantics of formulae inside mathematical texts is improved by extracting fixed-structure natural language formulations containing both text and mathematics. The results are integrated with MathWebSearch.

Yokoi and Aizawa define the similarity of mathematical formulae based on set intersections of the formula subpath sets [YoAi09] obtained from their parse trees[16]. The Jaccard's coefficient is used as the basis of the similarity function. The performance was done on a collection of $155,607$ formulae.

Samarasinghe and Hui propose feature keywords, extracted using specific regular expressions e.g, MathML matching "<mi[ˆ>]*>\p ..." which is tagged with the "function" keyword [SaHu09]. Next to the textual keywords, keywords describing the mathematical expression are also extracted using a predefined set of regular expressions. Furthermore, the data is clustered with different clustering techniques i.e., Kohonen's self-organizing maps, k-means and agglomerative hierarchical clustering. The evaluation was performed on 884 formulae.

WolframAlpha (see Figure B.8 in Appendix) is a computation knowledge engine which can return information about particular mathematical equations. It is an answer engine designed specifically for answering questions over curated data focusing on mathematics related ones. The system can return information about mathematical functions but also intermediate steps performed while computing the result. At the moment, it seems to be the best place to look for precise answers regarding mathematical functions. It has been estimated that this answer machine runs on more than $10,000$ CPUs and that the project has more than 15 million lines of code.

2010

Kamali and Tompa extend their work with tree based indexing and a mathematical query language based on MathML; a wildcard-like approach [KaTo10]. The work is based on the observation that many subtrees appear repeatedly in various trees. Therefore, the index contains unique trees built from subtrees with pointers to the fragments they are built from. The index keys are the hash signatures of the (sub)trees. They specify four wild cards i.e., for numbers, variables, operators and expressions. The evaluation dataset contained $297,300$ formulae.

It is important to mention that an initiative of translating (La)TeX articles from

[16] A parse tree is an ordered, rooted tree that represents the syntactic structure of a string according to some formal grammar.

the Cornell ePrint archive [Arch] to MathML [StKo08] using Bruce Miller's LaTeXML [Mil08] was very active in this year. It is connected with Kohlhase's research [St+10, Gi+11]. The activity was backed by the arXMLiv project [Arx12].

A completely different approach to the already existing ones is the content-based image retrieval of formulae using hand-written formulae as the query input described by Yu [Yu10]. Their algorithm uses X-Y cuts which segments the formulae into smaller syntactic blocks along the axis (not necessarily semantically correct) by projecting the bitmap into a space density graph. The (dis)similarity function used for ranking computes the image similarity using dynamic time warping[17]. The evaluation dataset contained 400 pages from CVPR 2008[18] where 200 were used for training and 200 were used for testing.

Ma et al. depict a feature extraction algorithm in their work [Ma+10a] for cluster based retrieval. This work is based on the work by Samarasinghe et al. [SaHu09]. Both semantic and structural features are extracted using more general algorithms than a static set of regular expressions. The cosine similarity function is used to rank the formulae [19]. The evaluation was done on 884 formulae.

The first reference to yet another full text mathematical search engine (see Figure B.7 in Appendix) is by Líška in his bachelor thesis [Líš10] supervised by Sojka. According to Appendix B, the whole indexing phase is an exact rewrite of the algorithm described in detail by Mišutka [Miš07]. This work is the basis for the search engine used at (eu)DML-CZ. The evaluation was done on $384, 183$ formulae.

2011

Mišutka and Galamboš think that another step towards making mathematics in digital form more accessible was to enable mathematical searching in one of the world's largest digital libraries - Wikipedia. A complete rewrite of the frontend and a new version of the Egothor backend was used. The EgoMath v2 search engine has been made easily portable and configurable[20]. The indexing process exploited the latex2mathml[21]

[17] Dynamic time warping (DTW) is an algorithm for measuring similarity between two sequences by finding an optimal match of feature vectors which allows for stretched and compressed sections of the sequence.

[18] Available at `http://www.cvpapers.com/cvpr2008.html` (seen March, 2012).

[19] See Chapter 5 for a in-depth description of the algorithm and its reimplementation with improvements done for this thesis.

[20] Compare with the state-of-the-art EgoMath which had been ported to a new backend without problems described in Chapter 4.

[21] Available at `http://tex2xml.kwarc.info` (seen March, 2012).

service developed by the KWARC group [Kwa12].

The work on the mathematical search engine at (eu)DML-CZ is extended by Sojka et al. [Lí+11, SoLí11]. The main contribution is the creation of Mathematical Retrieval Collection - MREC version 2011.4.439 - built on arXMLiv containing $158,106,118$ formulae. The evaluation was done on the MREC collection. An interesting detail is using 448GB of RAM for their search engine during evaluation. The document set contains almost $440,000$ documents.

Another marginal work, built upon a very interesting dataset, is the LaTeX search which indexes papers form the well-known Springer publisher[22]. It is a simple full text searching over linearised LaTeX fragments. It searches through approx. $8,223,138$[23] LaTeX snippets. Even though it is not a real mathematical search engine, it is one of the very few examples of a commercial company exploiting the demand for searching for mathematical formulae.

The combination of simple textual search with image based retrieval is described by Zanibbi and Yuan [ZaYu11a]. The formula was indexed after transforming it to simple text. The approach was inspired by Miller and Youssef [MiYo03]. Several visual features were extracted from each formula e.g., pixel density. The evaluation was done on a data set consisting of $26,737$ mathematical formulae. The conclusion of whether combining the two different techniques improved the results is unclear.

2012

Kim et al. convert mathematical expression into natural language sentences which are then indexed using a standard full text search engine [Ki+12]. Two types of features are extracted: patterns of identifiers, numbers and operators and structures. There were $1,800$ equations used during evaluation performed by 10 testers who tried to create queries from 200 given equation.

Birialtsev et al. write about semantic searching of formulae (in Russian) [Bi+12]. The search is based on the extraction of semantically annotated variable names from formulae. The query contains variable names e.g., volume or charge and the result includes formulae containing those variable names.

Nguyen et al. describe two additional approaches to mathematical retrieval. In the first paper Nguyen et al. claim that mathematical search systems adopting con-

[22] Available at `http://www.latexsearch.com/` (seen March, 2012).

[23] Seen March, 2012.

ventional text retrieval techniques are ineffective in searching for mathematical expressions. Because of this, they uses the Formal Concept Analysis[24] for similarity [Ng+12b]. The FCA works with features extracted from the MathML representation of a formula. The features are linearised sub-paths, each subpath fragment is concatenated into one single word. Some of the features can contain the real values besides the subpath. Mathematical formulae belonging to one concept are ranked using Jaccard's coefficient on the feature sets. The important part of FCA is the soundness of the underlying attributes which specify the relation, in this case, the extracted features. In this case, the level of understanding mathematics from the semantic point of view is not very clear. The performance testing was done on 489 formulae. In the second approach the search engine searches for keywords and mathematical features. A use case is a question answering system working over e.g., MathOverflow[25]. The interesting part is using Passive-Aggressive algorithm[26] adapted as a similarity function [Ng+12a]. The authors created a TREC-like document set with 31,288 mathematical questions and answers together with 30 manually evaluated queries for their evaluation.

The indexing technique used in MathWebSearch is based on the substitution trees and for acceptable performance must be available in fast memory e.g., RAM. Therefore, Kohlhase et al. examine distributable indexes [Ko+12] in their MathWebSearch. The evaluation was done on $115,000,000$ expressions. Another paper [KoIa+12] presents a method for searching the mathematical knowledge space which allows the answering of questions on the properties of symbols if some properties are known (e.g., if an associative, unital, idempotent magma structure with an operation is also commutative).

Substitution trees were also used by Schellenberg et al. They focus on semantically poor data in contrast to Kohlhase et al. The evaluation was performed on $24,479$ mathematical formulae. They compared their solution to the solution by Zanibbi and Yuan [ZaYu11a]. Different metrics showed different results, keyword based solution outperformed the substitution trees in the evaluation of the first 5 and 20 results. Substitution trees achieved better results in comparison of the first result.

[24] Informally, in contrast to the term independent vector space model the Formal Concept Analysis (FCA) uses term dependencies to find concepts. The interesting observation is that a certain "closure" of the relations is implied which can be exploited for similarity search. An important feature is that the concepts can be easily visualised.

[25] MathOverflow, available at `http://mathoverflow.net/`.

[26] Informally, Passive-Aggressive algorithm is used for online binary classification on learning tasks where datasets are not linearly separable.

It is worth mentioning the work by Kamali et al. who try to recognise and classify mathematical queries from Bing logs [Ka+12]. This can be seen as another step towards the integration of mathematical search engines into traditional search engines despite the fact that the authors claim that mathematical queries should be computed in real time. The reason can be that the focus in the paper is specifically on queries which require simple precise answers.

Marginal Work

There are also the papers [KoFr01, Gui03, As+04, Urb06, Th+06] addressing mathematical formula search in some way, more related to the theorem provers which is outside the scope of this survey. Experiments with a natural language processing technique the LSA[27] on a formalised mathematical library were described by Cairns [Cai04]. He claims that LSA can perform useful retrieval without explicit semantics, find conceptual associations between mathematical notions and also that the users themselves are able to formulate query expressions for which LSA returns useful results. However, he also identified several issues, the most important being the reliability of the queries.

Despite the author's poor knowledge of Japanese, the paper by Kishimoto et al. [Ki+05] is related to the mathematical searching research field. The authors suggest linking the function of similarity-based retrieval for formulae to the function of word-related associative search applied to mathematical terms. They try to find interesting keywords using the LSA method per page and map MathML formulae into the keywords produced by LSA.

There has been one attempt to address mathematical searching by using MathML in MySQL [MiIg08].

String features, extracted from formulae indexed as text with edit distance as a similarity function, form the basics of an algorithm presented by Guan et al. [Gu+12].

Recognising mathematical notation and creating the original mathematical formulae has been an active research field. Performance evaluation is straight-forward and important. The work by Sain et al. proposes an evaluation framework based on tree matching [Sa+11]. This work partly overlaps with the mathematical retrieval research

[27] Latent Semantic Analysis (LSA) is a theory and method for extracting and representing the contextual-usage meaning of words by statistical computations applied to a large corpus of text that uses a mathematical technique called singular value decomposition to identify patterns in the relationships between the terms. Latent semantic indexing (LSI) is an indexing and retrieval method that uses singular value decomposition in the same way as LSA. It is clear that in most cases the term LSI can be replaced with LSA.

field because they need to compare and/or find similar formulae of the recognised result and of the expected one which Sain et al. do using tree edit distances.

There are several references to Mathdex [Miš07, AlYo07, You07, MiMu07, Zh+08] a web-based search engine developed by Design Science which should be able to handle mathematics but no additional up-to-date details could be found by the authors but non-working references[28].

Two citations [Ng+12a, Ng+12b] of another formula search engine available[29] have been found. It seems to do simple textual searching over English Wikipedia.org.

There are at least two commercial projects aimed at mathematical searching. The first one is Scientific equation search[30] (see Figure B.5 in Appendix) which acts like a meta search engine combining their own search framework with data from e.g., WolframAlpha. This project went online in October 2012. The second project is (uni)quation[31] (see Figure B.6 in Appendix) which started around the end of 2009 but has not been updated since 2011[32]. The input is TEX-like and the index contains several well-known mathematical document providers (e.g., MathOverflow, MathExchange[33], dxdy[34], PlanetMath[35] and Wikipedia.org). The underlying technology is not known but experiments show that both work very similar to e.g., the EgoMath mathematical search engine by Mišutka [Miš08b]. They claim to index around $746,000$ mathematical formulae [36].

There are several other search engines which seem to search in mathematics without much information available. Moca$\iint$in [37] [Zh12] takes a different approach in mathematical searching. It allows for searching segments (e.g., theorem, proof, axiom, claim) with operators (e.g., followed by, has part, has text) building a semantic graph. There is an online demo of Equation search[38] which seems to search for mathematics but no further details are available.

An interesting paper by Trott and Weisstein [TrWe12] summarises several of the

[28] MathDex references http://www.ima.umn.edu/2006-2007/SW12.8-9.06/activities/Miner-Robert/index.html, http://www.mathdex.com:8080/mathfind/search (seen March, 2012).

[29] Available at http://shinh.org/wfs/ (seen March, 2012).

[30] Symbolab Scientific Equation Search, available at http://symbolab.com/, formerly known as $\sum$QsQuest.

[31] (uni)quation mathematical expression search engine, available at http://uniquation.com/en/.

[32] Seen March, 2012.

[33] http://math.stackexchange.com/

[34] http://dxdy.ru/

[35] http://planetmath.org/

[36] Seen March, 2012.

[37] Available at http://cll.niimm.ksu.ru/mocassin/ (seen January 2013).

[38] Available at http://www.equationsearch.com/, seen January 2013.

mathematical search engines publicly available today and the possible future for mathematical searching. According to the statistics in this paper, WolframAlpha is the most used mathematical search engine today with 1-2$*10^6$ mathematical searches per day. However, the natural semantic question answer search system in mathematics still has a long way to go. There are several case studies which question what users really want from a mathematical search engine.

3.2 Elementary but Often Unanswered Questions

> **Special Note**: One of the basic motivations of this research field is fairly simple. With the increase in the amount of digital mathematical content the problem of being able to search through it came along. The list of possible applications of the results from this research field grew. The author thinks that some of the solutions are mature enough for production use. However, the author also sees a problem with too heterogeneous attempts to solve some of the problems. The results are too fragmented and for people outside the core community it can be difficult to find useful information about the state-of-the-art of this research field. It is also difficult to find information about the possible solutions with their advantages, disadvantages, performance results and hopefully demo versions. There have been several discussions about a sort of competition of mathematical search engines. The first real attempt to create a dedicated application for creating datasets by the author of this thesis in 2010 has not been adopted yet[39]. The following set of questions should improve the basic information about mathematical search engines, the motivations and not least the actual implicit assumptions researchers work with.

To clarify the preconditions, assumptions and goals which the authors of mathematical search engines work with we suggest a list of questions. We helped ourselves with basic questions which should be asked from the project management point-of-view before project start even though a mathematical search engine is in fact already a product. The list of questions which should clearly characterise a mathematical search

[39] For latest development in this matter see section 3.4

engine and provide more details about it is:

- What problem(s) does your mathematical search engine solve?
- Who are your prospective visitors (and customers)?
- What do you want visitors (and customers) to do with the search engine?
- Who will benefit from your mathematical search engine?
- How would you define "two equal mathematical" formulae and "two similar mathematical formulae " in regards to your search engine?
- How will you evaluate the success of your search engine?

3.3 Case Studies

A more detailed description of mathematical search engines is necessary for a complete picture of the research field and its current state-of-the-art and to understand which features are important for the authors and which claims the particular search engine is built upon. The selected search engines are representatives of different approaches in this research field.

First, we describe the search engine used in DLMF, the oldest one available followed by the description of the EgoMath mathematical search engine is also based on a full text search engine but semantically mathematically aware follows. Then, we describe MathWebSearch which is unique in the chosen indexing structures and the community around it which addresses not only issues strictly connected with mathematical search but mathematical knowledge in general. The final search engine (answer engine) described in this section is WolframAlpha representing a commercial product with millions of users per day.

Despite the fact that all four projects can be labelled as mathematical search engines they are different in the target audience, the purpose and goals. DLMF targets special functions and the user should have good knowledge about what he seeks. EgoMath aims for digital mathematical content mainly on the internet (where most mathematical content has very little semantic information available) and focuses on similarity incorporating mathematical operations together with up-to-date full text search. The user should start with a generic query and refine the search. MathWebSearch uses special indexing structures giving the ability to use more complex and precise mathematical queries relying on the semantics of the input data. MathWebSearch query language(s) offer several ways to specify different properties of the query but this implies that the user should have a very good picture of what he is going to search for. WolframAlpha speciality is the huge curated data set including precise mathematical knowledge offering precise answers including the steps by which the answer was created. The user must formulate a precise query with the help of the advanced linguistic parser.

3.3.1 DLMF

DLMF[40] is the oldest mathematical search engine based on a full text search engine. The project started in 2002 [MiYo03] and the online search functionality was launched in 2008[41]. The original goal was to replace the Handbook of Mathematical Functions by Abramowitz and Stegun [AbSt72] with an online version. The mathematical content was created by a team of experts specifically for DLMF. At the time of preparing the content the MathML format platform had not been mature enough and that is why the LATEX format, even though not easily suitable semantics encoding, was chosen. Moreover, using MathML without proper tools - not available at that time - is very cumbersome. Later, the LATEX format was converted into MathML format [Mil08] which is used to represent the mathematical formulae.

The Apache Lucene search engine is used as the core engine. The mathematical content is converted to text in 3 steps: textualisation, linearisation and normalisation. The first two are trivial converting $\sin^2 x + \cos^2 x$ into e.g., sin begin_superscript 2 end_superscript x + cos begin_superscript 2 end_superscript x. The normalisation was mentioned in several papers [MiYo03, YoSh06] and takes care of reordering tokens (e.g., ^x_y and _y^x) and mapping equivalent forms into a unique format (e.g., x/y to \frac{x}{y}). Textual search is available as it is based on a full text search engine. The curated content indexed by DLMF allows for mappings of keywords to functions e.g., searching for BesselJ returns one of the definitions as the first result [MiYo08].

The summary of this mathematical search engine based on the questions above is as follows. The main goal of DLMF is to represent mathematical formulae in the MathML format in an analogous way to how the user attempts to search for them and

[40] Available at `http://dlmf.nist.gov`, see Figure B.1 in Appendix.

[41] According to `http://web.archive.org/web/20080914065211/http://dlmf.nist.gov/` (seen March, 2012).

which is suitable for a text-based search engine. Another goal is to handle "relaxed" queries which allow for more matches and provide highlighting of the query terms in the results. The audience should be primarily applied scientists (physicists, engineers, mathematicians) interested in properties of special functions and their applications. A user should be able to discover and/or verify properties of special functions.

3.3.2 EgoMath

A comprehensive description of EgoMath can be found in Chapter 4.

3.3.3 MathWebSearch

MathWebSearch[42] is a mathematically aware search engine not based on a full text search engine. The first version was publicly available at the beginning of 2007.

MathWebSearch is an example of a non-textual approach where expressions are parsed into a substitution tree [Gra96] (used in symbolic mathematical systems such as theorem provers). The result is a tree-like structure with nodes containing substitutions of its parents. The formula is constructed from a root node by applying one or more substitutions. To allow searching for sub-formulae , it has to add all its sub-formulae to the index separately. As the only search engine it fully supports α-equivalence. In the latest version of MathWebSearch several new unification-based query types were introduced [Ko+12]. The queries can be categorised into instantiation, generalisation, variation and unification queries. These query types try to match parts of the input formula into formulae which are indexed e.g., by generic substitution or renaming.

The most imminent problem can be seen in the performance as the search for a formula must traverse complex substitution trees which must be in fast memory e.g., RAM. This was addressed in [Ko+12] and resulted in distributed substitution tree index. Because MathWebSearch also aims for indexing documents and not only formulae , textual search should be available. However, the core indexing structure does not support it. This issue was also addressed to some extent by Ştefan Anca [Anc07]. MathWebSearch semantic querying assumes that the indexed content is semantically rich and that the mathematical structures where indexed formula live have specific properties.

[42] Available at `http://search.mathweb.org/`, see Figure B.2 in Appendix.

The summary of this mathematical search engine based on the questions above is as follows. The main goal of MathWebSearch is to make the search for mathematical formulae available using several different techniques. The audience should be mathematical practitioners and automatic theorem provers who should be able to search for theorems.

Lately, the authors of MathWebSearch seem to focus on mathematical knowledge rather than on the indexed mathematical formulae. This has many implications some of them described in a paper by Kohlhase and Iancu [KoIa+12].

3.3.4 WolframAlpha

The WolframAlpha Computational Knowledge Engine[43] is an answer engine designed specifically for answering questions over curated data focusing on mathematics related ones. It went online in 2010. The system can return information about mathematical functions but also about the steps done in computing the answer. At the moment, it seems to be the best place to look for exact results regarding mathematical functions with more than 90% of successfully answered questions[44]. According to the statistics [TrWe12], WolframAlpha is the most used mathematical search engine today with 1-$2{*}10^6$ mathematical searches per day. It has been estimated that this answer machine runs on more than 10.000 CPUs and that the project has more than 15 million lines of code.

The clear disadvantage is that this project works only over selected data with precise answers so the user must know precisely what he is looking for.

The summary of this mathematical search engine based on the questions above is as follows. The main goal of WolframAlpha is to answer any question that has a quantitative answer in different research fields. The audience is practically anybody who has a concrete question.

3.4 Latest Development

During the writing of the thesis, Math IR happening took place at CICM 2012[45] which should be the start point of cross-evaluation of available solutions.

[43] http://www.wolframalpha.com/.

[44] Article by Stephen Wolfram available at http://blog.wolframalpha.com/2012/04/17/overcoming-artificial-stupidity, seen April 2013.

[45] More details at http://www.cicm-conference.org/2012/cicm.php?event=mir&menu=happening.

CHAPTER 4

EgoMath

The theory and implementation behind the mathematical search engine (MSE) EgoMath is described in detail in this chapter. We will cover the latest version (version 3) of EgoMath by extending and updating the paper published by Mišutka and Galamboš [Miš08b] which describes the basic paradigms and algorithms in EgoMath version 1. The original paper is included in Appendix J. It should be read before reading the next sections in order to understand the important algorithms. It is also recommended to understand the basic anatomy and operation of a search engine. There are many papers addressing this issue including the book by Büttcher et al. [Bu+10]. Furthermore, we mention several implementation details of the underlying Solr[1] enterprise search platform which uses the Lucene[2] search library and we recommend a very good in-depth description of Lucene by McCandless et al. [Ma+10b] which applies to most of the full text search engines available today.

We start with a brief description of EgoMath's evolution. Then we describe EgoMath by following the workflow of a common search engine. This includes processing of the input data, indexing, ranking and querying. We also briefly touch on the user interface. Finally, we summarise the contributions and draw conclusions.

[1] Project available at `http://lucene.apache.org/solr/` (seen March, 2012).

[2] Project available at `http://lucene.apache.org/core/` (seen March, 2012).

Evolution of EgoMath

The first version of EgoMath was a proof of concept of the theory based on a prerelease version of the Egothor v2 search engine[3]. A custom user interface based on Struts and Taglib Java web technologies was developed and this version was made publicly available in 2007.

The main goal of EgoMath version 2 was to offer mathematical searching to a wider audience on a well known data set containing common knowledge which allows for simple cross-evaluation when a user knows what he is searching for. The choice was easy and Wikipedia.org was chosen as the main document set. Moreover, there are many other ways to search for knowledge in this digital encyclopaedia which could be used to verify the results from EgoMath. Version 2 was made available in 2010 with a new graphical user interface (see Figure D.1). Despite the fact that later versions of Egothor offered all the needed required components we chose the Solr enterprise search platform 3.x as the underlying technology for newer versions of EgoMath. The upgrade demonstrated that EgoMath's architecture is well designed and extensible. The complete upgrade was finished in a few days. A new mathematical search user interface was developed in PHP as a stand-alone library. Changing implementations, data sets and evaluating the results was the main motivation for creating the Importer framework (see Section 4.3).

We chose to tune the performance for large document sets based on the results and evaluation. In version 3 - the current one - we moved to Solr 4.x which offers more up-to-date features of full text searching including a mature distributed platform. However, instead of horizontal and vertical scaling, we still focus on the performance of non distributed versions as we think that even larger collections of document sets (e.g., arXiv.org[4] document collection) should be processed and made available on mediocre hardware.

Before describing EgoMath in detail, we answer the questions defined in Section 3.2. The answers should further clarify our motivation and goals.

[3] Project available at `http://www.egothor.org/` (seen March, 2012).

[4] Project available at `http://arxiv.org/` (seen March, 2012).

4.1 Answers to Elementary Questions

What problem(s) does your mathematical search engine solve?

The main goal of EgoMath is to enable search for mathematical formulae focusing on real-world documents offering a mathematically aware and deterministic similarity search.

Who are your prospective visitors (and customers)?

Anyone who has knowledge in natural sciences and has a question which can be encoded using formulae.

What do you want visitors (and customers) to do with the search engine?

Users should search for mathematical knowledge by entering mathematical formulae and iteratively refining their search. They should also be able to inspect the result set and receive more information about it than a simple list of the first N results.

Who will benefit from your mathematical search engine?

People from natural sciences who either search for the newest mathematical knowledge, are interested in cross domain searching or want to find similar formulae or results.

How would you define "two equal mathematical formulae" and "two similar mathematical formulae" in regards to your search engine?

The answer depends on the formula context where the formula is defined or is expected to be defined e.g., the mathematical structure with its axioms.

We regard all mathematically equal formulae as similar. However, when two formulae are visually completely different but mathematically equal, although they are similar, the similarity is very low and can be disregarded. We say that two formulae are similar even if they are not mathematically equal but can be easily transformed into the other by applying simple operations like renaming variables (similar to α-renaming in λ-calculus), adding items (sub-formula search) or disregarding subscripts (similarity search).

How will you evaluate the success of your search engine?

Firstly by the number of visitors, then by user feedback, successful queries indicated by logs and finally by the number of returning visitors.

4.2 Architecture

When speaking about EgoMath v3 MSE, we refer to several different components. The first one is the visible GUI stand-alone library which interacts with the search server using REST[5] API. The second part is the search server built on the Solr search enterprise platform with several major modifications and additional libraries. The most important part surviving throughout all versions is the EgoMath library itself. Both the search engine and the EgoMath library are written in Java and therefore, the interaction is straight forward. The library includes parsers for different mathematical formats, the abstract tree representation of formulae, algorithms creating different representations, different mathematical tokenisers, postfix and infix converters, static and dynamic normalised representations of symbols and operators, and also the implementation of the feature based approach (described in Chapter 5).

The main part of EgoMath is the web application deployed as a standard WAR[6]. Parallel crawling and pre-processing of raw input files in different formats (i.e., html, XML Wikipedia.org format, raw MathML, LaTeX) is done by the Importer framework described in the next section. The Importer framework uses the EgoMath library to process the input and convert mathematical notation into different formats i.e., MathML to LaTeX, MathML to EgoMath's index format (see Table 4.2) and LaTeX to EgoMath's index format. For specific conversion from LaTeX into MathML, the Importer collection of mathematical formulae is used which was created using an external web service (see Section 4.3). The technical overview of EgoMath is depicted in Figure 4.1.

Let us first define several important technical terms and paradigms used in the rest of this chapter.

Definition of Important Terms

In the context of full text search engine indexing, a *term* can either be a normal word or a group of characters with specific meanings (under specific conditions, terms can

[5] Representational State Transfer (REST) is a style of software architecture for distributed systems like the World Wide Web.

[6] A WAR file (or Web application ARchive) is an archive file used to distribute a web application.

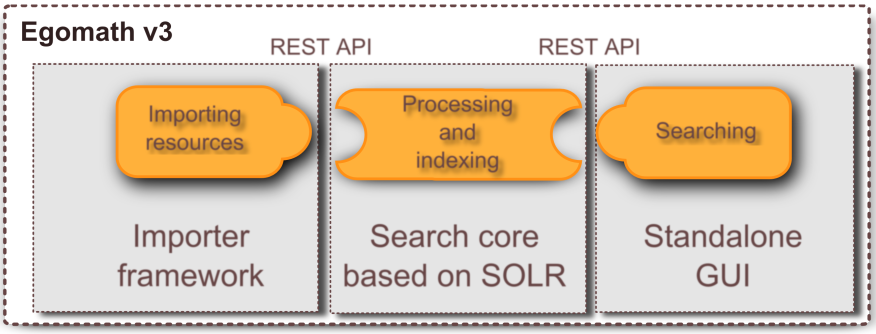


Figure 4.1: System architecture of EgoMath from a search engine point of view

also contain spaces). Terms are the smallest information unit which can be searched for and cannot be further split. A *token* is an occurrence of a term from the text. It consists of a term's text, the start and end offset of the term in the text and often a type.

During the indexing phase a document is converted into a dictionary like structure. Each key-value pair in the dictionary defines a specific *field* which can be searched for. Simple full text search engines use only one field which contains all the text. In the most simple case, a field contains information about the text in a bag-of-words[7] model. More often, the field also contains additional information, like the positions of individual words or information about words in the same position (e.g., useful for including synonyms or stems[8]). When a document is indexed, individual fields are often subject to tokenising and analysing algorithms. A field usually consists either of one element or is *multivalued*, which means it contains an array of elements. The difference between multivalued fields and simple fields from the search point of view is minimal. A naive implementation of a field supporting arrays simply increases the position of the word at the beginning of each element by a very high number, thus keeping the fields separate for most of the queries (an exception can be proximity queries). The only mandatory field in EgoMath is an array of EgoMath's representations of mathematical formulae.

[7] The bag-of-words model is a simple representation of text where the text (such as a sentence or a document) is represented as an unordered collection of words.

[8] In IR, a stem is a base form of a word to which affixes can be attached e.g., one possible the stem of "friendships" is "friend".

4.3 Importer Sub-project

The first phase in a typical search engine workflow includes resource parsing and importing. We designed and implemented a *novel framework* which enables the connecting of different datasets with different search engines. The framework was implemented in the programming language Python. There are several reasons for this decision: 1) Python allows for building cross platform applications with relatively easy programming access to applications written in several other languages e.g., Java, C, C++ (with C front end); 2) datasets are usually files which can be processed in parallel and Python offers an easy-to-use library for concurrent code execution; 3) easy to develop plugin architecture and 4) high quality libraries. The Importer architecture is depicted in Figure 4.2.

Datasets are usually composed of self-contained objects which are independent of each other. These objects can be split into reasonably big files. Therefore, the parallelism is based on parallel access to files. See the Wikipedia.org case study below.

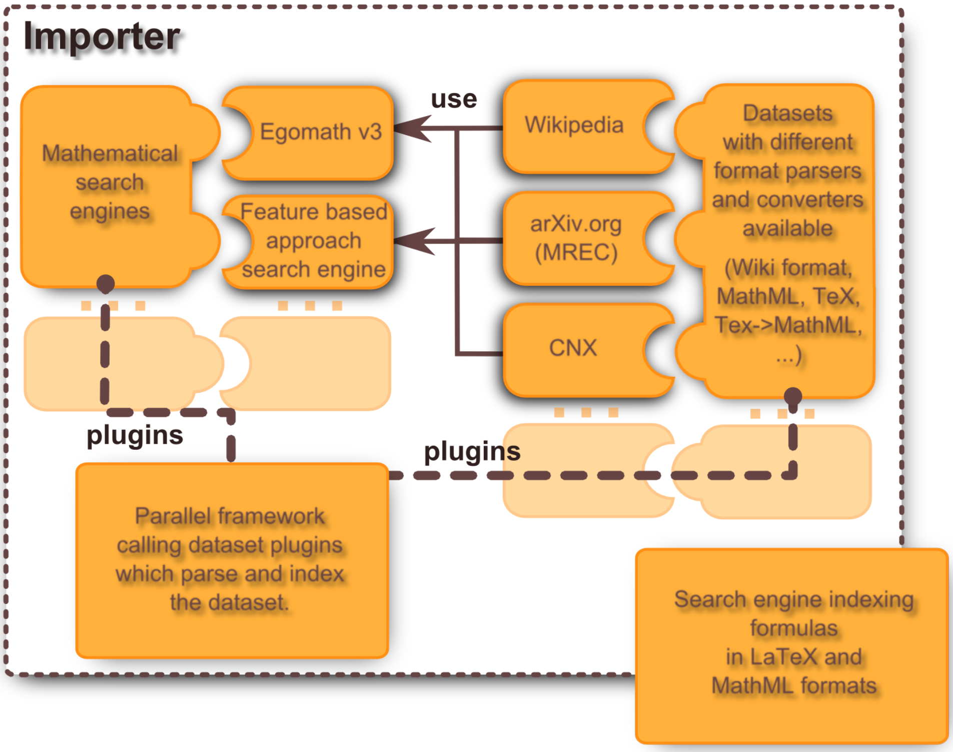

Importer
Mathematical search engines
Egomath v3
Feature based approach search engine
use
Wikipedia
arXiv.org (MREC)
CNX
Datasets with different format parsers and converters available
(Wiki format, MathML, TeX, Tex->MathML, ...)
plugins
plugins
Parallel framework calling dataset plugins which parse and index the dataset.
Search engine indexing formulas in LaTeX and MathML formats

Figure 4.2: Importer framework architecture

Importer Formula Database

A formula database sub-project of the Importer framework evolved into a new formula search engine (note, that the "mathematical" adjective was not used). After parsing two datasets (Wikipedia.org and MREC2011.4.439) in several iterations, it contains approx. 50 million of different mathematical formulae in LaTeX and MML formats including additional information, like the datasets, and an example document from each dataset (if applicable) where it was found. The searchable database is used for multiple purposes. The Wikipedia.org plugin, inside the Importer framework, uses it for fast LaTeX to MML conversion access. The feature based approach search engine (see Chapter 5) uses it for fast querying of formulae from a particular data set. It is also used for statistical information and can be used for several other purposes e.g., verifying and improving the conversion of the external LaTeXML converter or verifying MML content inside a browser (there have been multiple requests for an extensive MML test collection e.g., by the WebKit Open Source community).

Adding Support for New Datasets

New datasets are added simply by creating a new Python module[9] inside the directory of *datasets* and defining the function *exported_process*. The Importer process flow is depicted in Figure 4.3.

At the moment, there are three indexer backends available i.e., EgoMath, the Importer Formula database and the Feature based search engine. There is support for LaTeX and MathML expressions, MathML to LaTeX conversion using EgoMath library and LaTeX to MathML conversion using LaTeXML.

4.3.1 Case Study - Wikipedia.org

The process of indexing Wikipedia.org by EgoMath was automated by the Importer framework. The only requirement is the availability of the Wikipedia XML dump. For instance, the latest dataset i.e., enwiki-20130204-pages-articles.xml from the 4th of February 2013, is almost 43GB and contains $11,569,790$ pages.

The first step in the process is to split the file into several small ones so we can access them in parallel using the Importer framework. We use approx. 4GB files which are split on the article boundaries and are filled with appropriate headers and foot-

[9] The simplest Python module is a new directory with one empty file called *__init__.py*.

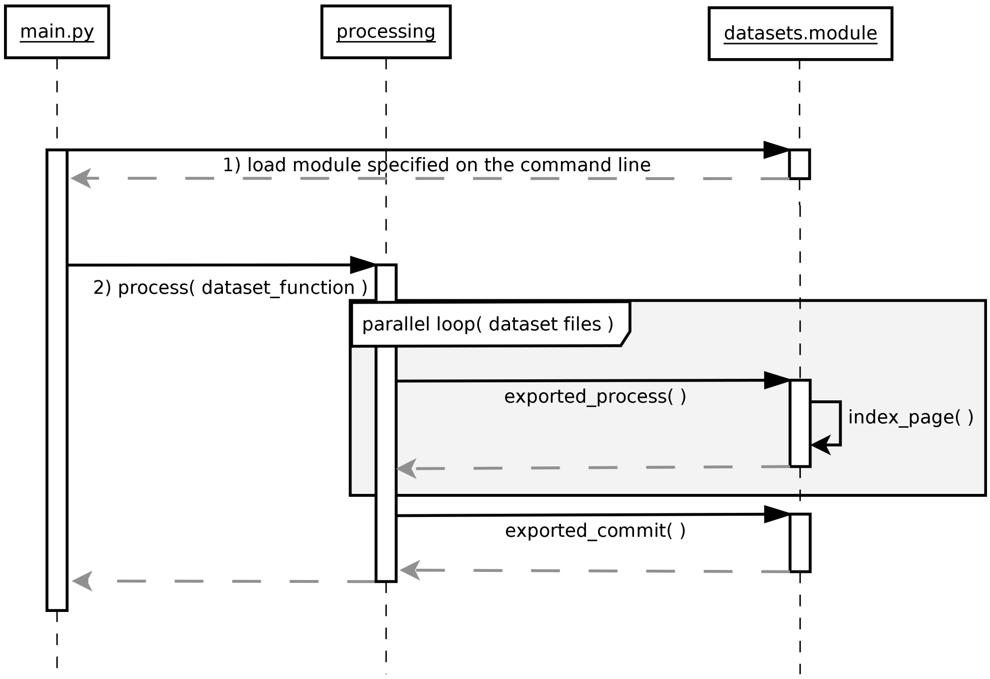

main.py
processing
datasets.module
1) load module specified on the command line
2) process(dataset_function)
parallel loop(dataset files)
exported_process( )
index_page( )
exported_commit( )

Figure 4.3: Importer workflow

ers so that each file is a valid XML file. The next step is to extract only the articles containing mathematics. According to our implementation, an article contains mathematics when it contains <*math*>[10]. We do not consider mathematics encoded using the *texhtml* format (less than 0.01% of total unique formulae). However, the Importer framework also parses Wikipedia mathematical html templates e.g., "{{math|sin &pi; {{=}} 0}}"[11]. The next step is to concatenate the output files into one file containing all the articles with mathematical formulae. This file is almost 550MB containing $32,662$ documents with approx. $490,500$ formulae[12], out of which approx. $291,000$ are unique after normalisation. Out of the $32,662$ articles, around $4,000$ articles use mathematics in unsupported parts of the page e.g., comments, summary tables or there is no formula left after normalisation. The distribution of formula count on a page, and formula length, is visualised in Figure 4.4.

The next step is to create stand-alone files with LaTeX formulae converted into MML. The files are created using a templating system and are exported to html for-

[10] Wikipedia's Manual of Style section, on the Mathematics web page addressing the typesetting of mathematical formulae, describes several ways of typesetting mathematics on Wikipedia.org.

[11] Using parallel execution in the Importer framework and using the Importer Formula SE on a 4 core machine with a max. of 6 concurrent processes, the processing of Wikipedia.org was six times faster than the sequential one (still, the bottleneck was disk I/O).

[12] During the processing we removed non mathematical formulae which were typeset using the mathematical style e.g., "\;".

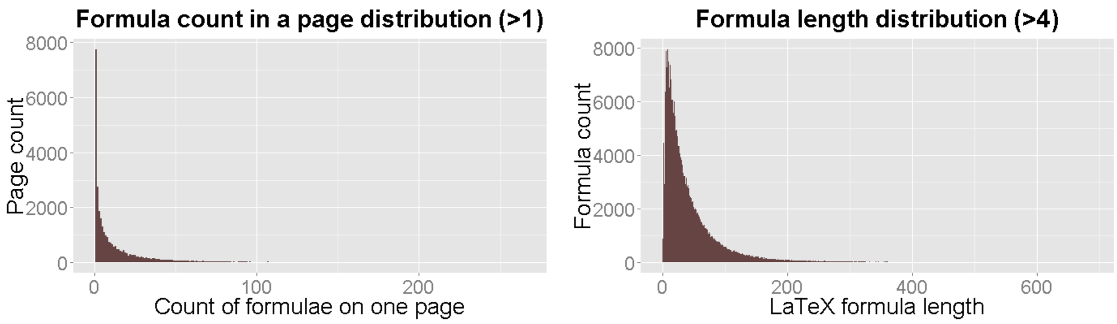


Figure 4.4: Statistics of the English Wikipedia.org dump (enwiki-20130204-pages-articles.xml)

mat, including meta information available from the Wikipedia markup like id, url, categories, languages and references. The Wikipedia parser cleans the text before sending it to the templating system (e.g., comments, noincludes, nowiki and tables are removed). Several types of articles are automatically removed e.g., Wikipedia templates and articles marked for deletion (step 1 to 3 in Figure 4.5).

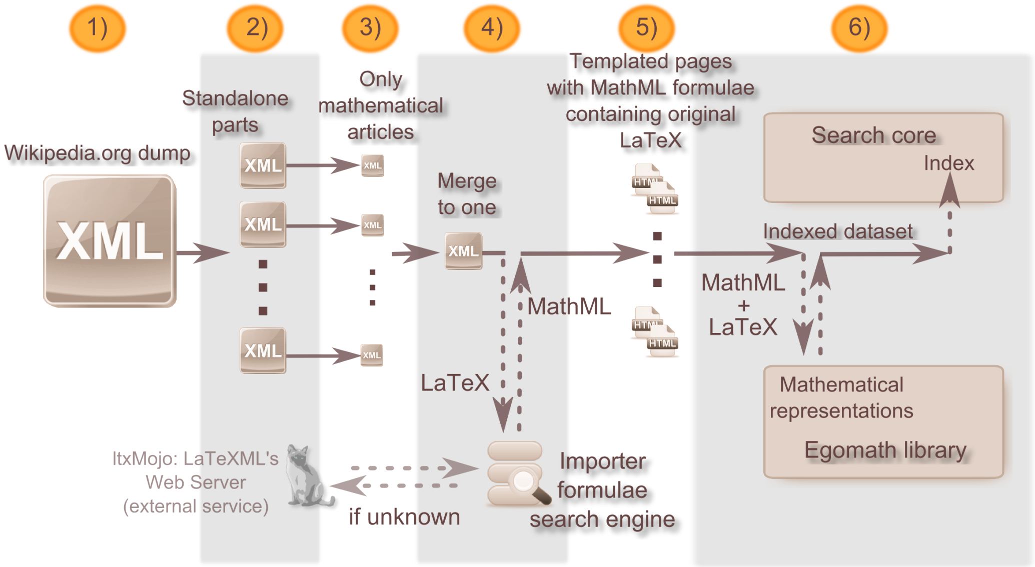

1)
2)
3)
4)
5)
6)
Wikipedia.org dump
XML
Standalone parts
XML
XML
XML
Only mathematical articles
XML
XML
XML
Merge to one
XML
Templated pages with MathML formulae containing original LaTeX
HTML
HTML
MathML
LaTeX
Search core
Index
Indexed dataset
MathML + LaTeX
Mathematical representations
Egomath library
ltxMojo: LaTeXML's Web Server (external service)
if unknown
Importer formulae search engine

Figure 4.5: Wikipedia.org indexing process workflow using the Importer framework

The formulae are converted into MathML representation using the LaTeXML's Web Server (step 4) created by the arXMLiv project[13]. The MathML representation contains the original LaTeX formula in annotation. This is the real bottleneck in the process taking about three days to convert the 290, 000 unique formulae (but using a sparing approach). To overcome this, we use the Importer formula search engine for

[13] Available at http://latexml.mathweb.org/about (seen March, 2012).

the conversions if available.

An important advantage of this approach is that each result file can be used separately and that is why the files can be used to create new smaller datasets. In the Wikipedia.org case study, LATEX formulae are converted to MathML representation (containing the original LATEX) which is inserted back into the text at appropriate positions (step 5). Then, the EgoMath library (written in Java but directly called from Python using the Java Native Interface - JNI) is used to create mathematical representations of each formula (step 6). All the information available including the mathematical representations is then sent to the search core which parses and stores it in its index (step 6). The complete importing workflow for Wikipedia.org is depicted in Figure 4.5[14].

4.3.2 Case Study - arXiv.org

Another interesting dataset for mathematical searching are documents from arXiv.org hosted at the Cornell University Library. Mathematical formulae in the documents had been converted to MML in the arXMLiv project[15]. Furthermore, the converted document set was further processed by Líška et al. [Lí+11] who included only the successfully converted ones and polished the document representation[16].

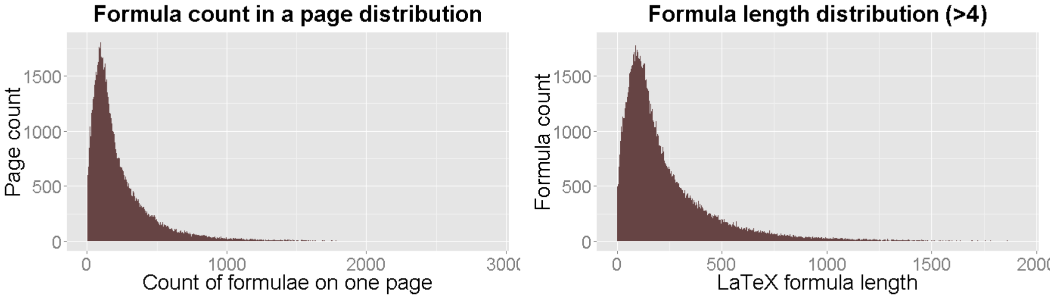


Figure 4.6: MREC2011.4.439 statistics

The overall indexing process is similar to the Wikipedia.org case study but without the need to create individual files. There are $439,423$ ($432,943$ containing at least one mathematical formula) with approx. $158,000,000$ usable formulae. The size of the document set is 121GB. Each formula is firstly parsed, converted into EgoMath's internal tree representation (formula tree) and then converted to LATEX. This process

[14] Indexing $32,662$ files using 6 parallel processes takes approx. 20 minutes.

[15] arXMLiv project uses LaTeXML[Mil08] to convert arXiv.org documents into xhtml. More details are available at `https://trac.kwarc.info/arXMLiv/wiki` (seen March, 2012).

[16] The document set is available at `https://mir.fi.muni.cz/MREC/` (seen March, 2012).

involves four conversions (arXiv.org LaTeX → MML → MREC MML → EgoMath's formula tree → normalised LaTeX) during which the semantics are guessed at various levels. The distribution of formula count on a page and normalised LaTeX formula length is visualised in Figure 4.6[17].

4.3.3 Case Study - Feature Based Approach Search Engine

The last case study is another approach to mathematical searching described in Chapter 5. The Importer Formula search engine is used to perform similarity testing. The process includes defining the important features used for indexing, indexing the features, normalising them and performing the similarity search on test queries. The whole run on the Wikipedia.org document set ($258,404$ formulae) takes approx. 15 minutes.

4.4 Parsing the Input

The first step in a SE workflow is to obtain a data set and to mine the relevant information from it. The ingenious indexing and searching techniques are of no use if the input data are processed incorrectly. Furthermore, pre-processing and normalisation can improve the search experience and the quality (recall and precision in general) of the results which was also observed by Normann [NoKo07] and Miner and Munavalli [MiMu07].

In 2006, the MML format was not widespread and it was clear that without proper tools and development environments, the format will never become widespread[18]. Therefore, we chose to support both the LaTeX and Presentation and Content MML. Both parsers are implemented inside the EgoMath library. The MML format consists of the Content MathML format which can encode semantics of a formula and Presentation MML format which is used to control how a formula is displayed. The fragments for the Content and Presentation MML are shown in Listing 4.1 and Listing 4.2 respectively.

[17] Indexing $439,423$ files using 6 parallel processes takes around 2 days.

[18] Sadly, we think that this is still the case.

Listing 4.1: Content MathML

```
<apply>
 <apply>
  <apply/>
  <plus>
  <ci>a</ci>
  <ci>b</ci>
 </apply>
 <cn>a</cn>
</apply>
```

Listing 4.2: Presentation MathML

```

<msup>
 <mfenced/>
  <mi>a</mi>
  <mo>+</mo>
  <mi>b</mi>
 </mfenced>
 <mn>2</mn>
</msup>
```

When inspecting the example, it is clear that Content MML is not strict with the definition of its identifier elements marked with the *ci* element. If a simple markup of *<ci>a</ci>* is used, no semantic information is given about a at all and a can be e.g., a vector, a variable or a matrix. However, the element *ci* can have a *type* attribute which can define its meaning e.g., *type="vector"*. The usage of type depends on the authoring tool used. Lately, the MML parsers have been revisited in the EgoMath library because the LaTeXML's converter provides both Content and Presentation MML. The formula representation in MML is used in parallel to LaTeX representation. The MML is acquired by the Importer formula search library which internally uses the mentioned converter. Another layer of conversion introduced additional differences between individual input format representations, but on the other hand, it adds robustness to the solution. Let us inspect a simple example of different notations of the π symbol. Several different representations of π are shown in Table 4.1 with their corresponding formula trees. If we choose the semantically poorer representation (e.g., π is a variable) then we loose important information which we had, which could have been used in restraining queries. If we choose the semantically richer representation (e.g., π is a constant) then we would restrain the result list of a loose query (simple querying for π). EgoMath v3 uses all the different notations creating formula representations for each of them and using the representation if it is unique. If the semantic meaning is ambiguous, EgoMath normalises the intermediate format shown in the figure and uses one predefined meaning as described in Section 4.5.

The power of LaTeX and the implicit mathematical typographic consistency (which we found out many people have a problem with) from the Wikipedia.org dataset was a complex stress testing for EgoMath's own LaTeX parser. The formulae are input by dif-

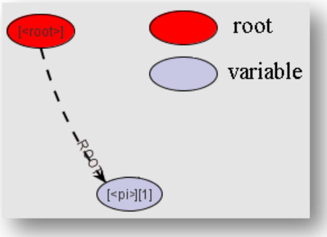
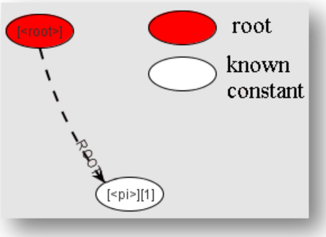
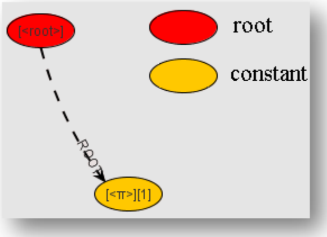
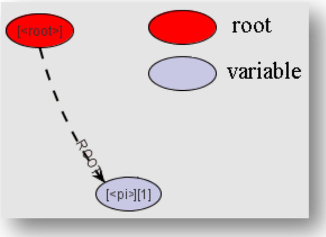
<mtext>&#960;</mtext>	<pi />
	
<cn type="constant">&#960;</cn>	\pi
	

Table 4.1: Table shows the parse trees of three different MML and one LaTeX notation of the same mathematical formula π. The important textual representation is shown above the graphical representation. From top left to bottom right: a parse tree with an unknown variable named π (&#960; is π's unicode html entity decimal code), a parse tree with the well known constant π, a parse tree with a constant named π and a parse tree with an unknown variable named π (equal to the first parse tree).

ferent users clearly following different rules. The problems encountered ranged from unclosed math tags to invalid or very ambiguous inputs e.g., ^{}_{}, ..., . . ., .   .   ., <math><math><math>, various Wikipedia specific TeX tags and spacing issues[19]. The pre-processing and normalisation steps contain around a hundred rewriting rules.

The LaTeX parser first cleans up the formula string by replacing all LaTeX space symbols for real spaces (e.g., "\ ", "\;", "\!" "\quad"), multiple spaces are normalised into one and several unimportant keywords are removed e.g., "\displaystyle". The input is split at word boundaries at specific characters e.g., "^_{}()[]=:;?.," and formatting markup (e.g., "\mathbf", "\mbox", "\textrm", "\cal") is ignored. Afterwards, number elements are parsed from string to numerical values, the fragments which are left at the end of words like ":;." are removed and each word is normalised to its base form (see Appendix J).

A very common semantic abuse in mathematics is to omit the multiplication sign.

[19] All the corrections have been submitted back to Wikipedia.org.

LaTeX notation "a b" (ab) is intuitively understood as $a * b$. Another very common and unfortunate habit is to interchange "1/2" ($1/2$) with "\frac{1}{2}" ($\frac{1}{2}$) or even worse with "1 \over 2" ($\frac{1}{2}$). These issues are addressed just before the infix notation is converted into postfix. The postfix notation removes many (otherwise necessary) brackets in the infix notation (e.g., $a + (b + c)$ becomes "$a\ b\ c\ +\ +$" while $(a + b) + c$ becomes "$a\ b\ +\ c\ +$"). It fails if the operators have varying arity[20]. The performance difference between the version using brackets and the one not using them is discussed in Section 4.9.1.

Then, the input is parsed into a parse tree. A parse tree (internally called a *formula tree*) is a tree structure with one root node allowing for easy breadth first and depth first traversal. Each formula tree can be exported to postfix, infix and LaTeX-based textual representation. There are several types of nodes: leaf, unary, binary and general n-ary. Currently, more than 200 operators, approx. 500 variables and approx. 50 constants are defined. Each node contains information about its depth. This depth does not necessarily correspond to the real depth in the tree, but more to the mathematical operation which can be performed on the same level e.g., leaf nodes in formula $a+b+c$ have the same depth, but leaves in a^b have different depths. The depth of leaves in index is increased by a predefined high number at every level.

Each node has a type e.g., variable, number, constant and can have additional properties (internally called decorations) which can be used to specify indices (bounds) and additional semantics. The arity of operators, the default representation, priority and whether it can be reordered inside the Ordering algorithm are specified in an XML file (symbols.xml) together with the symbol alternatives. A simple example of operator definition is shown in Listing 4.3. A similar format is used for constant and variable definition.

Listing 4.3: Example of an operator definition

```
<operator representation="sine" priority="HIGHEST"
  arity="1" ordered="0">
    <alternative>sinus</alternative>
    <alternative>sin</alternative>
</operator>
```

[20] Arity of a function or operation is the number of arguments or operands the function or operation accepts. Let us assume operator O can have arity 1 or 2. Then, the postfix notation "1 2 O O" is ambiguous and can be correctly represented in infix by $O(O(1, 2))$ or $O(O(1), 2)$.

Formula trees have normalised textual representations (e.g., “aˆb_c” and “a_cˆb” will both be represented in postfix as “a c _ b ˆ” despite the order of the operators).

Every formula can be exported to postfix, infix and LaTeX format.

4.5 Indexing

The indexing phase of EgoMath is a complex process of transforming input objects into structures which can be used for effective searching for mathematical formulae. EgoMath can handle not just simple formulae, but also documents containing more mathematical formulae, processing additional metadata about the document.

We have built our research on several proclamations. The first claim is about the missing semantics form research documents including mathematics. The second claim is that the process of searching can be (should be) iterative unless we know precisely what we are searching for and what the results should be. Even then, unless the user is a machine, the process can be iterative. If the user is a machine (e.g., a theorem prover), the underlying dataset should have specific characteristics which can be exploited to create much more precise queries and tune the search engine for precision e.g., by using the *additional semantics* technique described below. To improve the search experience, we make use of additional information retrieval models namely browsing and filtering. This is possible because research papers and articles follow unwritten conventions and include additional important and interesting information like references, authors, keywords, categories and cited articles.

Browsing and Filtering

We implement browsing and filtering paradigms with one technique called *faceted search* or faceting. Faceting is a technique of accessing information arranged into categories based on indexed terms. The result list returned by the search is extended with information, about the counts, of how many documents would be returned if a specific facet (e.g., name of an author) would be added to the query. Faceting makes it easy for users to explore search results and to filter exactly the results they are looking for.

Faceting can be easily implemented (not effectively, though) by adding new multivalued field specific values and extending each query by a boolean sub-query addressing the particular value. Several other MSEs focused on categorising the formulae using natural language processing techniques like LSA. Faceting is the ideal technique to use this type of categorisation, but on the search object level which is usually a document in EgoMath. If the search paradigm is solely focused on formulae (one searchable object is one formula), additional information about each formula can be directly exploited using the facets.

There is a performance penalty for faceted search depending on the document set. The performance cost of faceting without any caching is discussed and visualised in Section 4.9.7. However, facets are often simple numbers optimised for caching which reduces the performance penalty substantially.

Storing Mathematical Formulae

We omit the details of data mining and filling out the fields of the dictionary mentioned above and we will focus only on the field used to store mathematical formulae. The document is split into parts containing mathematical formulae and their surrounding text - snippets - as depicted on the left side of Table 4.2. The identified formula is sent to the EgoMath library which parses it as described in the previous section. We continue with the description after obtaining the formula tree.

The formula tree is converted into different representations. The general algorithm is described by Mišutka and Galamboš [Miš08b] and is depicted on the right side of Table 4.2. The "egomathh" keyword is used to mark the end of a formula and the keyword "eegomath" is used to mark the beginning of the first line which is used to visualise the content. The reason for this is flexibility and extensibility for different backends and purposes (e.g., the Importer function collection). The first word of each representation is *ego* and a number. The number indicates the level of abstraction it was created with or, more technically, the index of the algorithm in the *generalisation* and *augmentation* process which created the representation. The *ego* prefix is used to avoid polluting the space of words (in this case numbers) with words not belonging to the actual formula.

Let us inspect the technical details from the Solr enterprise search server to finish the indexing process flow. We will revisit formula representation building at the end of this section. We will describe important differences in the description of EgoMath provided by Mišutka and Galamboš and we will justify the mathematical model we chose including the reasoning of the impacts it has.

The EgoMath index has an associated structure defined in the schema.xml file (Solr implementation). Each field must be defined in this file, including the analysis steps

Wikipedia XML article	After processing by EgoMath
`<page>`	
`<title>Pythagorean`	
`trigonometric identity</title>`	
`<id>535827467</id>`	
`...`	`...`
`Mathematically, the Pythagorean`	`Mathematically, the Pythagorean`
`identity states:`	`identity states:`
`:&lt;math&gt;\sin^2 \theta +`	`Tex: eegomath \sin^{2}\theta`
`\cos^2 \theta = 1.\!&lt;/`	`+\cos^{2}\theta=1 egomathh`
`math&gt; This relation between`	`ego0 : 1 theta sine 2`
`sine and cosine is sometimes`	`^ theta cosine 2 ^`
`called the fundamental`	`+ = egomathh`
`Pythagorean trigonometric`	`ego6 : const theta sine`
`identity.`	`const ^ theta cosine const`
`...`	`^ + = egomathh`
`The elementary definitions of`	`ego10 : const id sine const`
`the sine and cosine functions`	`^ id cosine const ^ + =`
`in terms of the sides of a`	`egomathh`
`right triangle are:`	`This relation between`
`:&lt;math&gt;\sin \theta =`	`sine and cosine is sometimes`
`\frac{\mathrm{opposite}}{`	`...`
`\mathrm{hypotenuse}}= \frac{b}`	
`{c}&lt;/math&gt;`	
`...`	`...`

Table 4.2: There are two mathematical formulae in the Wikipedia XML format, with parts omitted by "..." on the left side. One possible snippet produced by the first pass of EgoMath's indexer is shown on the right side.

associated with it for both the indexing and querying phase. The analysis step transforms and normalises data in the fields to improve the searching experience by e.g., removing blank spaces, removing html code, stemming, allowing for wildcard searching or adding synonyms. Note, that the analysis steps defined for indexing can be (should be) configured for the query process too. The configuration schema for EgoMath's mathematical field is shown in Listing 4.4.

Listing 4.4: EgoMath's field configuration for Solr 4.2.1

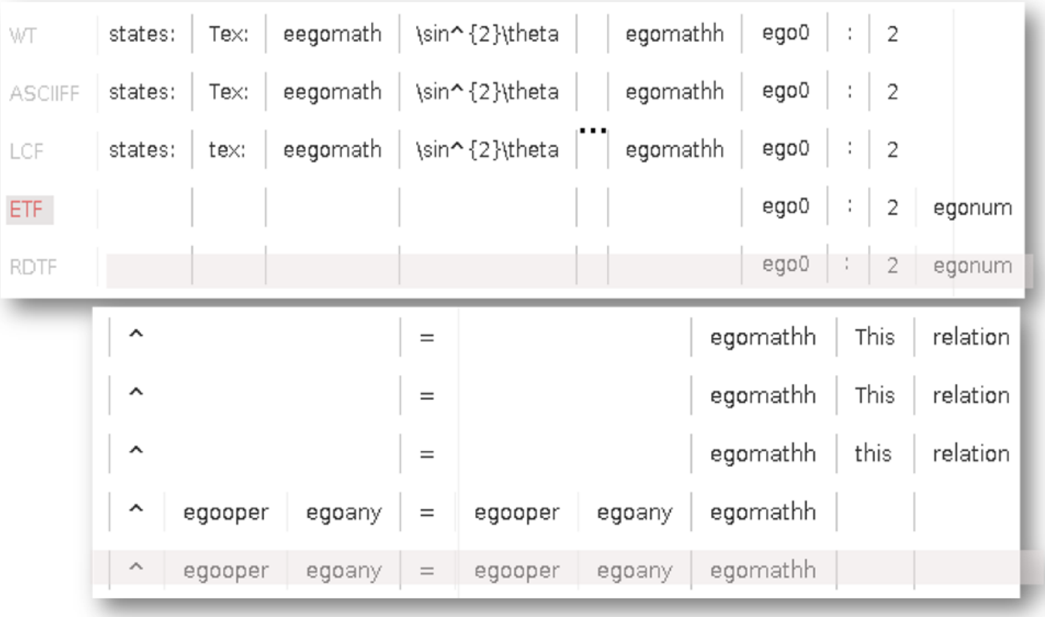
```
...
<fields>
 <field name="math" type="math_text" indexed="true"
  stored="true" multiValued="true"
  ... />
...
</fields>
...
```

```
<types>
 <fieldType name="math_text" class="solr.TextField"
     positionIncrementGap="10000" >
  <analyzer type="index">
   <tokenizer class="solr.WhitespaceTokenizerFactory"/>
   <filter class="solr.ASCIIFoldingFilterFactory"/>
   <filter class="solr.LowerCaseFilterFactory"/>
   <filter class="egomath.solr.analysis.EgomathTypeFilterFactory" />
   <filter class="solr.RemoveDuplicatesTokenFilterFactory"/>
  </analyzer>
  <analyzer type="query">
   <tokenizer class="solr.WhitespaceTokenizerFactory"/>
   <filter class="solr.ASCIIFoldingFilterFactory"/>
   <filter class="solr.LowerCaseFilterFactory"/>
  </analyzer>
 </fieldType>
...
```

The field used to store the mathematics is an array of the mathematical representations and the surrounding text. Each item in the array represents one formula which can be searched for; however, the surrounding text can contain additional formulae but with representation used for visualisation only. The field can be highlighted and therefore the whole content is stored, not just the individual mathematical tokens. Each token stores its position in the text. For each formula, we increase the positions by *positionIncrementGap* to minimise the possible overlap e.g., by proximity queries. The *EgomathTypeFilterFactory* further increases the positions for each representation. This class is responsible for payloads and for augmenting representations with *ego** keywords described below.

Searching for Types Rather than Real Values

EgoMath defines the index time filter factory, which takes tokens produced by the white space tokeniser and performs two steps. Firstly, it ensures that the actual searchable tokens are only those of the mathematical formula or its representation. This is done simple by skipping over and not indexing words outside of "ego + number" representations. Secondly, it allows for other types of generalisation queries. Let us assume we want to search for $\lim_{n\to\infty}(1+\frac{1}{n})^n$ but we do not care about the value of the first number 1. We would like to write $\lim_{n\to\infty}(\text{any number}+\frac{1}{n})^n$. We introduce keywords which are indexed exactly at the same place as specific types: 1) numbers (*egonum*), 2) constants (*egoconst*), 3) variables (*egovar*), 4) operators (*egooper*) and 5) any of the above (*egoany*). All keywords will be collectively denoted as *ego** in the following text. Indexing at the same place means that after indexing $a + 5$, a query of $a + 5$ will match and so will a query for $a + egonum$. The example EgoMath representation from Table 4.2 is, therefore, further augmented. The result of the index can be seen in Figure 4.7. There is one significant difference to functionality offered by the transformation which replaces occurrences of constants and numbers (constants2const) and variables (unknown2id) with appropriate keywords. It is the fact that the *ego** keywords can be used with real values. For example, if 5^2 is indexed then searching for $egonum^2$ would match but searching for $const^2$ would not because the actual representation returned by the transformation is $const^{const}$.



WT	states:	Tex:	eegomath	\sin^{2}\theta		egomathh	ego0	:	2	
ASCIIFF	states:	Tex:	eegomath	\sin^{2}\theta		egomathh	ego0	:	2	
LCF	states:	tex:	eegomath	\sin^{2}\theta	...	egomathh	ego0	:	2	
ETF							ego0	:	2	egonum
RDTF							ego0	:	2	egonum

^			=			egomathh	This	relation
^			=			egomathh	This	relation
^			=			egomathh	this	relation
^	egooper	egoany	=	egooper	egoany	egomathh		
^	egooper	egoany	=	egooper	egoany	egomathh		

Figure 4.7: Indexed text processed by appropriate filters (WT - WhitespaceTokenizer, ASCIIFF - ASCIIFoldingFilter, LCF - LowerCaseFilter, *ETF - EgomathTypeFilter*, RDTF - RemoveDuplicatesTokenFilter). The highlighted line contains the actual tokens that can be searched for. A thick horizontal line indicates token boundaries.

The *ego** generalisation is performed only on the top N representations where no real values were substituted for *const* or *id* keywords.

Because of the original algorithm also using ordering on the first representation, the query does not need to return expected results under specific conditions. Let us assume we are indexing $a + b$ and the first line is “ego0: a b +”. Searching for “egovar+b” is (correctly) reordered to “b egovar +” and does not match with “egovar b

+”. We completely avoid this issue for many formulae by adding the original indexed formula without ordering to the list of representations. We partly fix it for complex formulae where the ordering of sub-formulae (using the *ego** keyword) is changed by the ordering algorithm. In this case, if the user does not search in the correct order, the formula will not be matched. Using the keyword “egoany” is for advanced usage because the ordering cannot be guaranteed at the basic level only - the exact replaced item priority is not known in contrast to the other keywords.

The performance evaluation is discussed in Section 4.9.2.

Searching for Additional Semantics

One of the problematic parts of supporting document sets with multiple formulae per one logical indexed object - document - is including additional information per formula because we store all of the formulae in one field. We store several representations of one formula including generalisation (“ego*” keywords), but do not explicitly include additional information e.g., which variables are bound, the actual limits or the real mathematical model where it was found if available. EgoMath’s latest version supports additional semantic information. The difficulty is mostly in extracting the semantics from the real document sets because of the natural language processing but also the inconsistency where only a fraction of the formulae contains this information directly.

The problem for EgoMath lies in including local information for each formula without breaking the formula search. We keep information about the boundaries of individual representations and if the query is fully matched, the relevance is higher than if it is matched only partially. The solution is to append the semantic information behind the representation in the indexing phase. And also append it in the query phase to the end of the phrase being searched for. Let us assume that x is a bound variable in $a + x$ and we want to indicate this fact. If we want to add additional information we can embed it in the actual representation e.g., *ego1 : a x + egosem x bound egosem abel_group egomathh*. Sub-formula search will not be affected because of the *egosem* keyword but the search for the semantics itself will be matched. Moreover, we can use the same generalisation technique with the *ego** keywords. The interesting part is the question of how to query for such objects and the technical implementation. A query cannot be reasonably restricted to match all sub-queries only to one element in a multivalued field, so a different technique must be used. We use specific proximity searching which we have implemented in our phrase proximity query and join operations. A proximity search looks for objects where two or more separately matching term occurrences are within a specified distance (slop). The implementation uses additional information stored with each term - in this case the positions of the terms inside documents. The query searches for the actual mathematical formula and also for the desired property (properties). For each matched document the positions are incrementally processed and the distance is compared with the desired slop. The only issue which remains is how to define the slop because of the sub-formula search. This parameter can be tuned in accordance with the position increment after each representation inside one mathematical snippet and the position increment after each snippet in the array of mathematical formulae of the math index field, defined by "positionIncrementGap" (see Listing 4.4).

The performance evaluation is discussed in Section 4.9.3.

4.5.1 Justifying the Mathematical Model

One of the most important paradigms in EgoMath is choosing a common mathematical model for all formulae. If more information is available, it is added to the representations but does not replace them. The set of transformations including generalisation and augmentation algorithms is described in Mišutka and Galamboš [Miš08b].

There are several important differences. The current state-of-the-art parser and indexer module in EgoMath can handle constraints properly ($\sum_{i=0}$). Multi-line formulae are split into a set of formulae and are parsed correctly, we take advantage of a basic mathematical sentence parser. Complex structures like matrices are split into individual lines to include at least some level of recognition. However, the consequences in our data sets are minimal e.g., only 0.0017% of Wikipedia formulae are matrices.

Intuitively, every mathematical formula should contain information inside which mathematical structure it is defined and which axioms hold in it. Contrary, EgoMath defines this structure and enables mathematical awareness inside it. This is easily justifiable because no semantics of this kind are available in real-world documents and natural language processing is not mature enough for the task of recognising it[21].

The definition of the mathematical structure is in an XML file by the means of executed transformations which result in individual representations. EgoMath's v3 definition of the first three augmentation algorithms can be found in Appendix I. The

[21] To the best of the author's knowledge.

basic transformations are described in the referenced paper.

In the first algorithm, optional operators e.g., unary plus are removed at the beginning. Then, the same transformations are used as in EgoMath v2, including the well known and probably most common axioms *associativity*, *distributivity* and *commutativity*. We include the additional *replace_id_const_not_change* transformation which was present in v1 but removed in v2. It allows for a specific α-renaming similarity of formulae. However, it does not fit well into the abstraction process. It can happen that there are no results on this particular abstracted level because the numbers do not correspond but there are matches on other levels. If we indexed the formula $1+\pi^2$ and we search for π^2 the indexed formula will not match if we use the abstraction of converting constants to keywords done by the transformation *replace_id_const_not_change*; however, it will match on other abstraction levels. The reason is, that the indexed formula has one additional number which increases the index of const. Formula $1 + \pi^2$ is transformed to $const_1 + \pi^{const_2}$ but formula π^2 is transformed to π^{const_1}.

The second algorithm removes subscripts and is similar to the first one. The reason is to support similarity searching over similar notations e.g., $E_0 = mc^2$ is similar to $E = mc^2$. This was inspired by our work on the Feature based approach described in Chapter 5.

The third algorithm handles equivalences, e.g., $e^{\pi i} = 1$ and $e^{\pi i} - 1 = 0$ are transformed into equal representations.

In the original paper, the paradigm of *mathematical tokenisers* is used which are responsible for defining the smallest searchable unit (the atom of information). If we define that it can be searched only for whole formulae, indexing $a + b$ would result in one term $a@b@+$ ("@" is a glue to overcome default splitting of words) and search for a would not match. We found that the results from the original paper differ to results obtained from larger test set and the evaluation is described in Section 4.9.9. By default, EgoMath v3 intuitively uses the simplest tokeniser which accepts only single items so the example is indexed as three terms $ab+$. However, there could be good reasons, e.g., speed, why to use different tokeniser as described in Section 4.10.

We get to one of the important paradigms used by EgoMath [Mič08b]. *When the correct meaning can not be deduced, the solution is to choose one sole meaning and operate with the symbol identically in both the indexing and searching phase.* We define around 800 symbols and operators including different alternatives and typos in the symbol definition file. If a symbol is not recognised, it is assumed that it is a variable. Let us assume that we encounter $\prod x = 1$ during indexing and we do not know anything about the symbol $\prod$. During formula parsing, EgoMath inserts multiplication between elements which do not have appropriate operators. This directly comes from the perception of ab or $a(b+c)$ which are (by most users) regarded as $a*b$ and $a*(b+c)$ respectively. In the example above, the formula is converted to $\prod *x = 1$ and is indexed appropriately. If searching for $\prod(x)$ is relying on the additional "hint", that the brackets indicate that $\prod$ is a function, the result would be empty. However, we normalise the meanings and treat both occurrences equally ($\prod(x) \rightarrow \prod *(x) \rightarrow \prod *x$) the expected result is returned.

Section 4.9.6 contains the number of returned results for test queries that show the consistency of the different techniques used.

4.6 Ranking

The ranking of relevant objects does not only depend on the information stored during indexing; however, the crucial part lies there. Suppose we have a list of relevant documents. Documents in this list have exact words or phrases which were queried. EgoMath allows for querying both the text and mathematical formulae. There are many ranking functions based on standard information retrieval models or which utilise the similarity functions as described in Chapter 2. The definition of ranking functions often differs only in small details, but the impacts are huge. We will follow up on the well known definition of the cosine similarity used as a ranking function in Section 2.2. It should be repeated that for exploiting the fast access of relevant documents in full text search engines, using the *boolean model* of IR, the ranking function must meet several requirements (see Chapter 2). In EgoMath, the relevant documents are obtained using the *boolean model* of IR and the ranking is based on the well-known VSM model.

Ranking is used at different levels in EgoMath. Because EgoMath can index documents which contain multiple formulae, we rely on the ranking function which takes into account the relevance of particular terms (mathematical tokens) to the document itself. The transformations are already (logically) ranked by the representation number and representations with specific numbers are ranked higher. We also prefer formulas which match as whole. All these fragments create the resulting ranking function described below.

EgoMath exploits the ranking function implemented in Solr. Let us assume that the cosine similarity is defined as in Equation 2.2.1. The similarity function is extended to improve the quality and performance. The cosine similarity normalises each document with $\sqrt{\sum_{i=1}^{n} dw_i^2}$ (where dw_i is the i^{th} weight in the vector of weights representing document d), but this is known to have drawbacks[22] and is replaced. The abstract similarity formula is defined as:

$$score(q,d) = CF(q,d) * QB(q) * \frac{\sum_{i=1}^{n} (dw_i * qw_i)}{\sqrt{\sum_{i=1}^{n} qw_i^2}} * DLN(d) * DB(d) \tag{4.6.1}$$

where CF is the coordination factor and can be used to boost documents returned by a multi term query which contains more terms than others, QB is a query boost factor which can be used during query time preferring some queries to others, DLN is an improved normalisation factor known at the index time, DB is an index time boost for a particular document. The Equation 4.6.1 does not properly define the function in respect to different fields, term boosting and a special boosting technique which uses payloads as described below. However, the application in these cases is intuitive. It is also possible to replace the scoring definition, so that the scores are computed from indexed fields, we use this technique in the browse mode.

EgoMath exploits the query boost factor (QB), index time boosting (DB) and payloads described below.

Index Time Boost

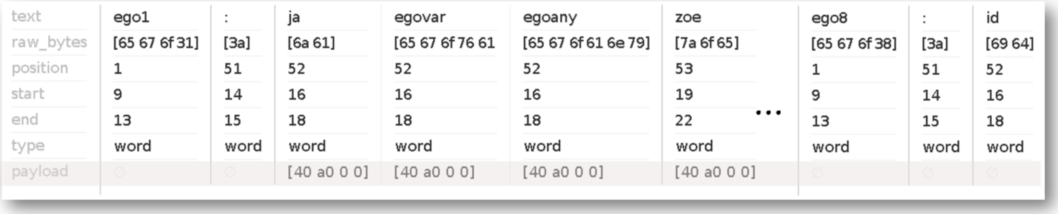
Searching for text and mathematics must be combined together. We chose to boost the field where mathematics is stored at index time in preference of searching for mathematics[23]. One obvious reason is that the text should not be the first ranking decision maker. Imagine we have two (very artificial) documents that contain the word "proof" and a (sub-)formula $a+b$. If the word "proof" is very relevant to one of the documents, it could happen that this score would beat the score of the relevancy of mathematics, which we want to avoid.

[22] Normalisation removes the length information from the vectors. If documents are created from duplicating sections, then the normalisation avoids boosting the relevance of one of the documents. However, if the documents have different lengths with different content (although many equal terms) the normalisation in the original cosine definition performs poorly.

[23] This approach was also used to boost documents using the reference links iteratively during indexing, but was later dropped because the quality improvement was not stable enough.

Payloads

Another technique already employed in 1998 by Brin and Page [BrPa98] is storing additional information for each term in addition to information stored for documents. This allows for different ranking of the same term in different logical parts of the document e.g., heading, footer and body by storing information about the font, size and place[24]. In Solr, the information stored for each index term is called *payload* and can be used for this purpose. We use it for boosting the ranking of particular representations. As mentioned in the original paper by Mišutka and Galamboš, more specific representations of one formula (technically, those are representations with lower ego numbers for each algorithm) are ranked higher, i.e. the terms which are part of a more similar representation to the original, have payloads with boosts as shown in Figure 4.8.



text	ego1	:	ja	egovar	egoany	zoe		ego8	:	id
raw_bytes	[65 67 6f 31]	[3a]	[6a 61]	[65 67 6f 76 61	[65 67 6f 61 6e 79]	[7a 6f 65]		[65 67 6f 38]	[3a]	[69 64]
position	1	51	52	52	52	53		1	51	52
start	9	14	16	16	16	19		9	14	16
end	13	15	18	18	18	22	...	13	15	18
type	word	word	word	word	word	word		word	word	word
payload	∅	∅	[40 a0 0 0]	[40 a0 0 0]	[40 a0 0 0]	[40 a0 0 0]		∅	∅	∅

Figure 4.8: Payloads for the representation (starting from ego1), which are more similar to the original one, in contrast to no payloads with less similar representations (starting from ego8).

In Solr, payloads require custom similarity class which alters the standard way each document is scored. We chose to use the average of the seen payloads which is added to the document score and Equation 4.6.1 is applied. EgoMath's actual implementation does not boost representation with an index number above 2 for every algorithm used (which technically means that the payloads are used for the first three representations modulo 100 e.g., payloads are done for ego0, ego1, ego2, ego100, ego101, ego102 and ego200). Using payloads in EgoMath makes sense for sub-formula search in the representations because we use additional query boosting for the whole representation. Let us assume that we are searching for πa and we have formulae $\pi a+1$ and $\pi(a+b)$ in the index. The first one matches in the first representation, the second one in the third (applying distributivity). We prefer the first one because of the payloads.

The performance evaluation is discussed in Section 4.9.4.

[24] It is interesting to mention that the first implementation of storing arbitrary information for each term occurrence, called payloads in Lucene, is from the middle of 2007 (Lucene 2.2.0).

Additional Techniques

Several types of datasets have internal structure and relations which can be used to improve the ranking. One typical and most successful example is the PageRank link analysis algorithm by Brin and Page [BrPa98] for relevance measurement. Our important document sets include Wikipedia where a similar approach can be used. Internally, EgoMath parses and stores information about the outbound links, but initial experiments have shown that the ranking scheme is complex enough and a simple boost of documents, which are often referenced, does not improve the quality in general. A much deeper analysis is needed which is out of the scope of this thesis.

4.7 Querying

There are four simple cases (when we omit searching for additional semantics) which can occur when a user "hits the default search button". A search for *text only*, a search for *mathematics only*, a search for *both* or search for *none*. Searching for *none* is interpreted as entering the browse mode where we stepwise display a list of all documents. In this case, the sorting order does not really matter; however, EgoMath indexing Wikipedia.org uses "ord(citations_count)^0.5" in the query to display documents referencing more external resources at the top. Searching for text uses the built-in powerful query parser (query language) in Lucene.

Strictly speaking, the mathematical query language in EgoMath is very simple - there is none. Everything the user submits as mathematics is sent to EgoMath and is interpreted as one formula. First, a request handler defines the logic executed for any request. Every handler can have components which are chained together as reusable pieces. We define a search component called *query_math*. The user enters a query and that gets into the *query_math* component. In the pre-processing step, the query is split into the textual and mathematical one, the default scoring chain is set and several other default pre-processing steps are taken in order to support e.g., faceting. Splitting the query is trivial because the fields particular terms belong to are explicitly mentioned.

Let us discuss an example. The user enters a query which is represented in this format: "{!edismax qf='text' lowercaseOperators=false q.op=OR v='lt Ja'} AND math:([1] n*k*a)". The part inside {} can be interpreted as using a specific query parser called "edismax", which queries inside the field "text" with a boolean operator between multiple terms set to "OR" if missing and, finally, with a value of the textual query string. The second part (after "AND") indicates to search inside the field "math" for "[1]n*k*a" marked by the brackets "()". Both of the sub-queries must match in order to mark the document as relevant. The math query is further split into the requested representation level "[1]" and the actual mathematical formula "n*k*a" (the representation level is 1 by default because the unordered original formula is at 0^{th} position).

In the next step, the formula is sent to the EgoMath library where it is parsed by EgoMath's formula recogniser and different representations are returned (the same representations as are returned in the indexing phase). The requested representation index is chosen if available otherwise, the first available after the specified index is chosen. The top level query is a boolean query into which the textual query is inserted.

In EgoMath v2, three mathematical sub-queries were constructed with one boolean query which prefers formulae at the beginning of a representation. This query was removed gaining approx. 25% on the tested queries (see Section 4.9.5). In EgoMath v3, only the two important sub-queries are used. The first one can be rewritten as *math:(": a k * n * egomathh")*[25] and will match only the whole representation. This query is boosted with a factor that can be specified by the user interface. This query is added into the boolean top query as "should" be fulfilled. The second sub-query is full sub-formula search represented by *math:("a k * n *")*.

Finally, the last case is searching for *mathematics only*, which is intuitively derived from the more complex case study above.

The iterative approach is ensured by different facets (filters). We use standard index/query components for faceted searching. We also allow the specifying of equal search only, which means that searching for equal formulae uses only the first mathematical sub-query returned from the EgoMath library.

Refining Search with Additional Semantic Information

There is one modification to the behaviour of query parsing described above when using additional semantics feature. The semantics can be entered after the "semantics" keyword in the formula as shown below. Searching for "$x = a_1 \cdot a_2 \cdots a_k$" can be extended with "$x = a_1 \cdot a_2 \cdots a_k$ semantics $a_i \in T$". The mathematical query is extended with the desired properties in the following way. In Solr there is a concept of query parsers which create a tree like representation of queries. Query parsers can be

[25] Based on the usage of payloads it is either a simple phrase query or a specific payload query with ordered terms which must be exactly beside each other.

concatenated inside one query by specifying the desired parser inside $\{\}$. We use proximity searching as described in the previous section and apply it to the phrase queries which are the core of EgoMath. Semantics are added to the top level boolean query as a required span[26] query, which collects information about phrases. An additional query parser was implemented to allow for such behaviour. A query for a where $a \geq 0$ could be rewritten as *{!egonear df='math' apart=5}"egosem a 0 greaterequal","egomathh" AND math:([1]a)* which can be interpreted as a search for math a, but the semantics $a \geq 0$ must be very near (not more than five words to the right from a) the keyword marking the end of a formula ("egomathh").

Exploiting Similarity Search - the "Abstract math" Feature

One of the important features of the EgoMath querying process is the "abstract math" feature, helping in the iterative search process. If a query returns 0 results, we suggest the user to abstract the formula, which means that the next representation is used.

Let us inspect a real example of searching for $\sum_{n=0}^{\infty} \frac{f^n(a)}{n!}(x - a)^n$. Using the Wikipedia.org dataset, we receive e.g., *Recurrence relation* and *Taylor series* result documents which is correct. However, the UI indicates (displays the actual abstraction level we are in and the maximum level we can reach) that EgoMath created more different abstract representations and we can try to search for them. The last representation created by EgoMath can be visualised as $\sum_{const=id}^{const} \frac{id*id^{id}}{id*id} * (id - id)^{id}$. Because we do not define the *!* operator and therefore, it is converted to id. We treat ∞ as a constant and the ordering algorithm puts it at the beginning ($const = id$). At last, $f(x)$ is treated as $f * x$. Due to these transformations and due to the fact, that we remove subscripts at the beginning of one of the algorithms, we were able to find two additional matches i.e., $f(z) \approx \sum_{k=0}^{\infty} \frac{f^k(c)}{k!}(z - c)^k$ and $T(x) = \sum_{n=0}^{\infty} \frac{f^n(x_0)}{n!}(x - x_0)^n$. Another view on the abstract math feature is that it allows for wider similarity or search in less complex mathematical structure.

Using Click-through Data

One of the state-of-the-art features used to improve the quality of search results, including personalised ones, is tracking user behaviour. We have installed a click-through handler which stores information in an external database. We chose to exploit

[26] A span represents a range of term positions within a document.

Google Analytics (GA)[27] for this approach and used code similar to the one listed in Listing C.1 in the UI to store information where the user navigated to, from which site and page number, including the position of the result. The screenshot of the information in GA is depicted in Appendix C. However, we have not been able to gather a reasonable amount of data for a proper evaluation of this feature.

4.8 UI

A lot of attention has been put into user interface design in the last few years. We have addressed this issue throughout the years but the outcome was ambiguous. A brief survey, including the work in progress, of a new graphical front-end for easy input of mathematical formulae was described by Mišutka and Klíma [MiKl09]. This user input has not been completely finished and it has never been adopted. Because EgoMath is a search engine, we used the paradigm of the giants (Google, Bing and Baidu) to have the user interface as simple as possible. We created two input fields, one for the textual and one for the mathematical query. The result list also resembles snippets from other search engines but they include graphical representations of mathematical formulae. Furthermore, filtering was implemented as additional information to the left of the snippets and interesting information e.g., categories belong below each snippet. Browsing uses pre-processed abstracts stored with each document. Annotated screenshots are available in Appendix D.

We think that most of the work on the GUI is outside of the scope of this thesis. Moreover, for a proper evaluation, the feedback of a relevant number of users is needed and we have not been able to obtain this. There are workshops particular for this research topic (e.g., the Mathematical User Interfaces Workshop at the CICM conference). However, we mention one technique and one technical detail because we think these should be obligatory for most of the mathematical UIs.

EgoMath supports highlighting of the found results or sub-formulae. A mathematical query is processed as a phrase, the documents are ranked and then another search component - highlighting - is executed. For each document from the result list the particular field (“math” in this case) is retrieved and the highlighter traverses through the terms, looks for mathematical formulae and tries to find and highlight the query terms.

[27] Project available at `http://www.google.com/analytics/` (seen March, 2012).

If documents have many mathematical formulae, the fields can get big and a lot of processing needs to be done. EgoMath's highlighter optimises the search for the best snippet by allowing for partial hits in bigger fields[28] which reduces the time by up to 60% (found out by inspecting EgoMath's logs). If the score of a fragment is non zero, it is increasing in time. After some time, if no other fragment is found the score can reach a level where it is accepted. Each found token is highlighted using a predefined mark up. This is returned to the UI which allows for displaying the precise matched elements as depicted in Appendix D.

The support for displaying mathematics in application is often very poor. Major browsers adopted the MML standards with various success but the future is unclear[29]. On the other side, the MathJax project[30] solves this issue by rendering mathematics with standard web technologies. We use MathJax for both the immediate visualisation of the entered formula and for visualisation of mathematics inside snippets. This is exactly the place where the first line of the representations e.g., "Tex: eegomath \sin^2(x)+\cos^2(x)=1 egomathh" is used (MML can be in place of the LaTeX representation when the original formula was in MML). The representation is further processed in UI, changing it to MathJax fragment, which is displayed. For the moment, highlighting fragments are done on the postfix notation of each representation. Because the LaTeX representation is available, even for most of the formulae originally encoded in MML, it would be easy to highlight specific parts. In EgoMath it would require parsing postfix to infix.

4.9 Evaluation

In this section, we show the results of evaluation of several features and techniques used in EgoMath. We show either the performance gain or functionality improvements with acceptable performance penalty. We summarise the results in this section and discuss the implication or justify the key architectural decisions.

[28] Similar techniques are used by all major full text search engines e.g., by limiting the number of characters the hit is searched for and when it is not found only a representative sentence is displayed.

[29] Sadly, the Chrome web browser turned off the MML support around February 2013.

[30] Project available at `http://www.mathjax.org/` (seen March, 2012).

Set-up and Results

The evaluation was performed on a Fujitsu Esprimo E900 with 4 Cores at 3.4 GHz with 12 GB of RAM running Windows 7/64-bit with Apache Tomcat/7.0.12 (Oracle Corporation Java HotSpot 64-Bit Server VM/64-bit 1.7.0_15-b03).

The evaluation was done by performing set of selected queries. The complete list of queries and results can be found in Appendix F. Because Wikipedia.org very likely contains many common knowledge subjects, we tried to compile a list of known and famous queries not restricted by our own research background. We found a web page called "Famous Equations and Inequalities"[31] which we used as the basis. All queries have been performed with caching turned off[32]. There were three warm-up runs before performing the evaluation itself.

Several figures show the file size of different file types used by the underlying search engine. The important file types are described in Appendix H. The query times in the figures below are divided into *prep* (preparation phase) and *proc* (processing phase). The preparation phase usually updated the actual query and the processing performs the query or the desired functionality. The suffix indicates the component whose execution time was measured: *f* means facet, *hl* means highlighting and *qm* means the query_math component described in Section 4.7.

We used two datasets during the evaluation. The first one contains mathematical articles from the English Wikipedia.org as described in Section 4.3.1 and the second one documents with at least one valid mathematical formula from MREC2011.4.439 as desribed in Section 4.3.2.

4.9.1 Postfix Notation With Brackets vs. Without Brackets

Postfix notation does not need brackets when the arity is known (see Section 4.4) but they improve readability and reduce the number of matched results when the indexed or query formula are incorrect (e.g., missing operators or operands). EgoMath v3 does not use brackets because the performance gain is significant.

The results shown in Figure 4.9 are intuitive. The index *with brackets* is bigger because of the additional positions and payloads which must be stored for the bracket

[31] Available at `http://www.math.utah.edu/~pa/math/equations/equations.html` (seen March, 2012).

[32] To be precise, we keep very basic sort of caching which stores fields used during processing of the same query. However, because we are not contributing absolute numbers we leave out the technical details.

terms and the presence in the inverted index itself. Because the brackets are very common, the index size *with brackets* is bigger, approx. by 8.5%. The indexing process is, also intuitively, slightly slower when using brackets.

The important result is the mean query time improvement which is 32.6%. However, in queries 73-78 ($id + id$) we would expect much better results from the version *without brackets*. More interestingly, the *with brackets* version is even faster in queries 73, 75 and 77. The reason is described in Section 4.9.6.

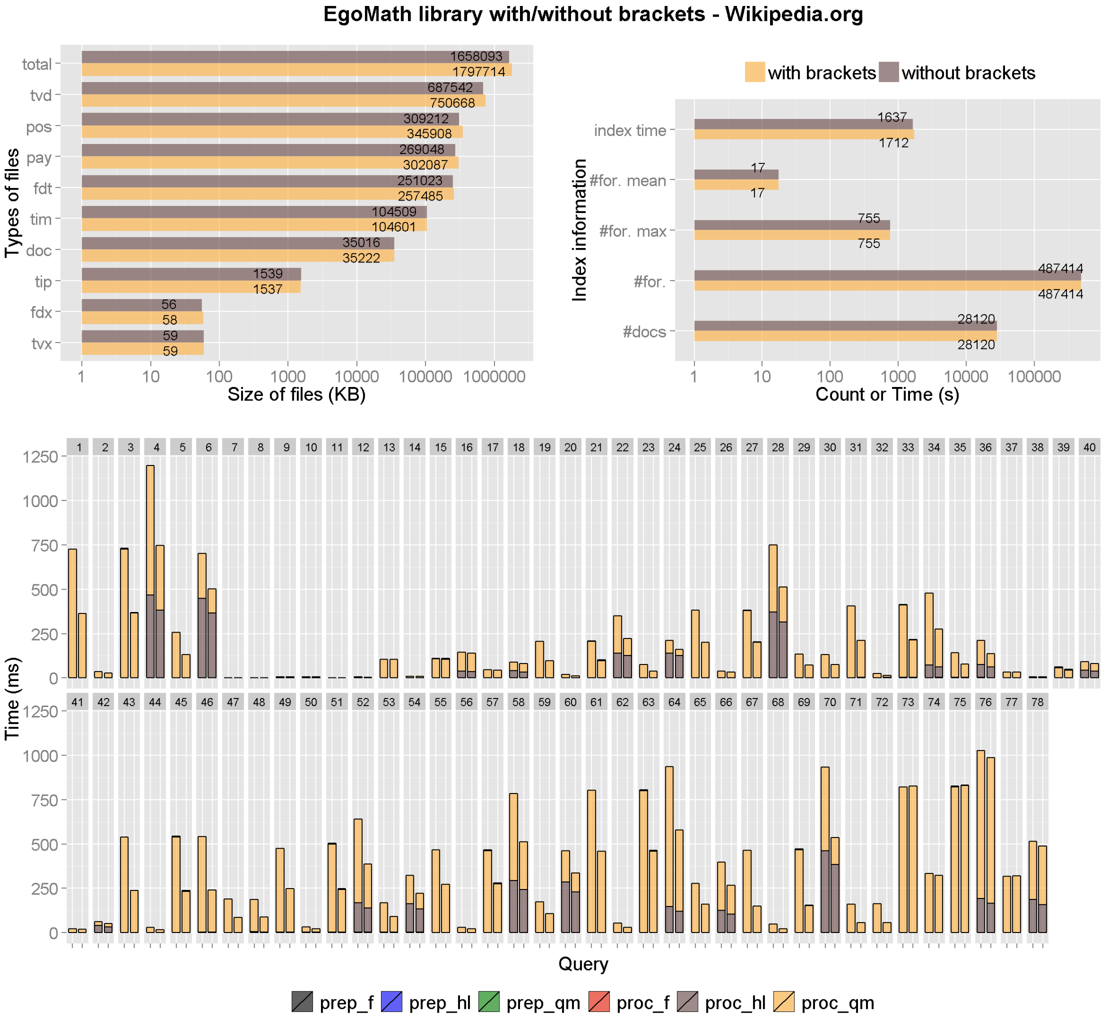


Figure 4.9: Performance of "with" and "without" brackets

4.9.2 Index Term Augmentation With ego* vs. Without ego*

We look at the performance of one of the techniques with great potential used in EgoMath version 3.

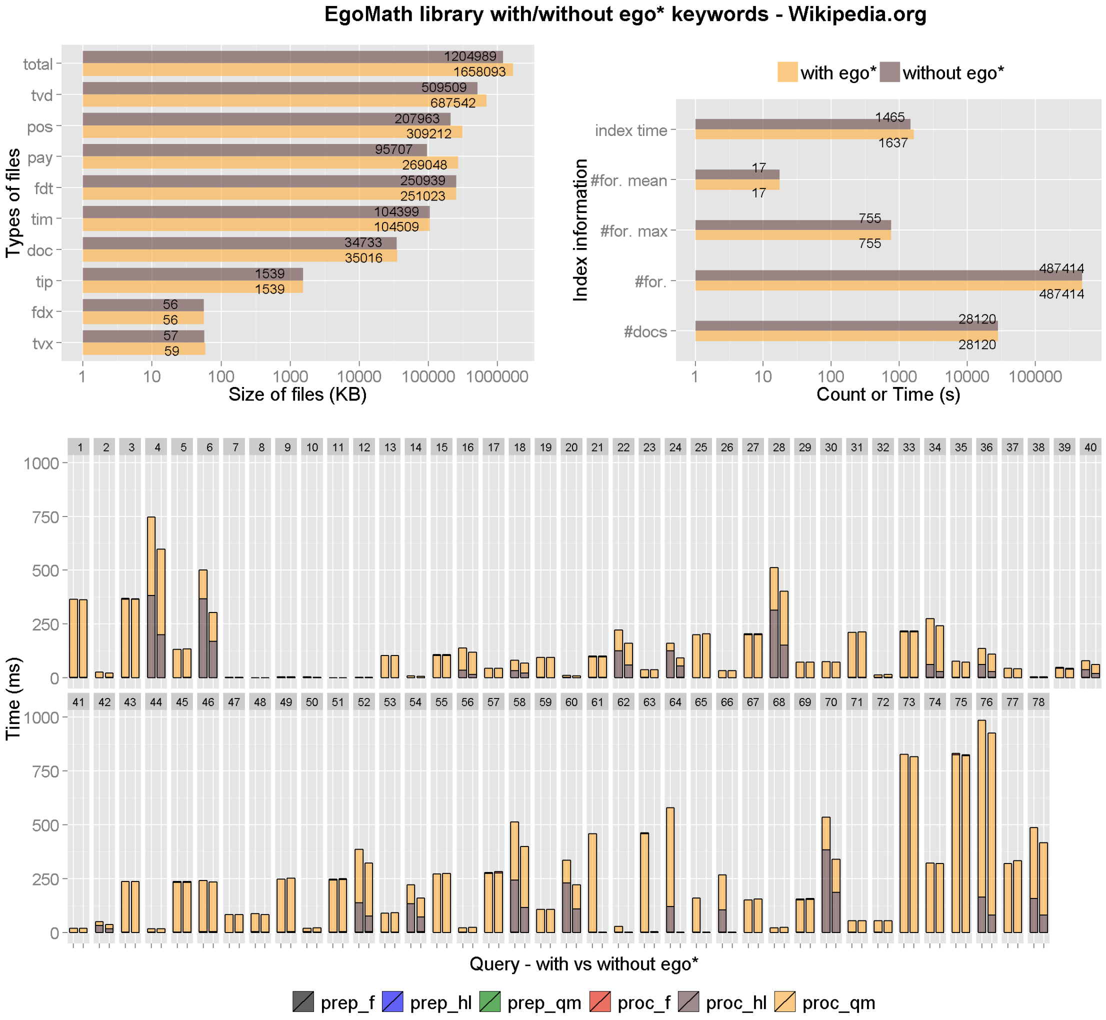


Figure 4.10: Performance of "with" and "without" ego* augmentation keywords

The results shown in Figure 4.10 are also intuitive. The index with *ego** keywords is bigger because of the same reasons as was the brackets version in the evaluation before. However, the index size with *ego** keywords is bigger, approx. by 27%, which is significant. One possible solution is not to include payloads for *ego** keywords which saves approx. 10% of the total space.

The indexing process is slower for the version with *ego** keywords because additional processing during the indexing phase must be done.

The query times in queries $61-66$ ($F = G\frac{m_1 m_2}{egovar^2}$) are significantly different because the queries contain the *egovar* keyword, which in case of no ego* keywords, has 0 results. Each query is executed with 6 different settings; therefore, every 6^{th} query is a new one (1, 7, 13, ...). The 4^{th} and 6^{th} setting use highlighting but the 6^{th} searches only for whole formulae (disabling sub-formula search). The highlighting component is, in cases where the raw query does not take much time, usually the performance bottleneck. When looking for the terms to highlight, more terms must be checked when using the *ego** keywords. It can be seen in queries 4,6, 10,12, 16,18, 28,30 etc. The only difference is query 46 and 48, where almost no highlighting is done. The query times with *ego** keywords are approx. 8.5% slower after removing queries 61-66 which is acceptable for the additional functionality.

4.9.3 Index With Additional vs. Without Additional Semantics

There are only subtle differences in Figure 4.11.

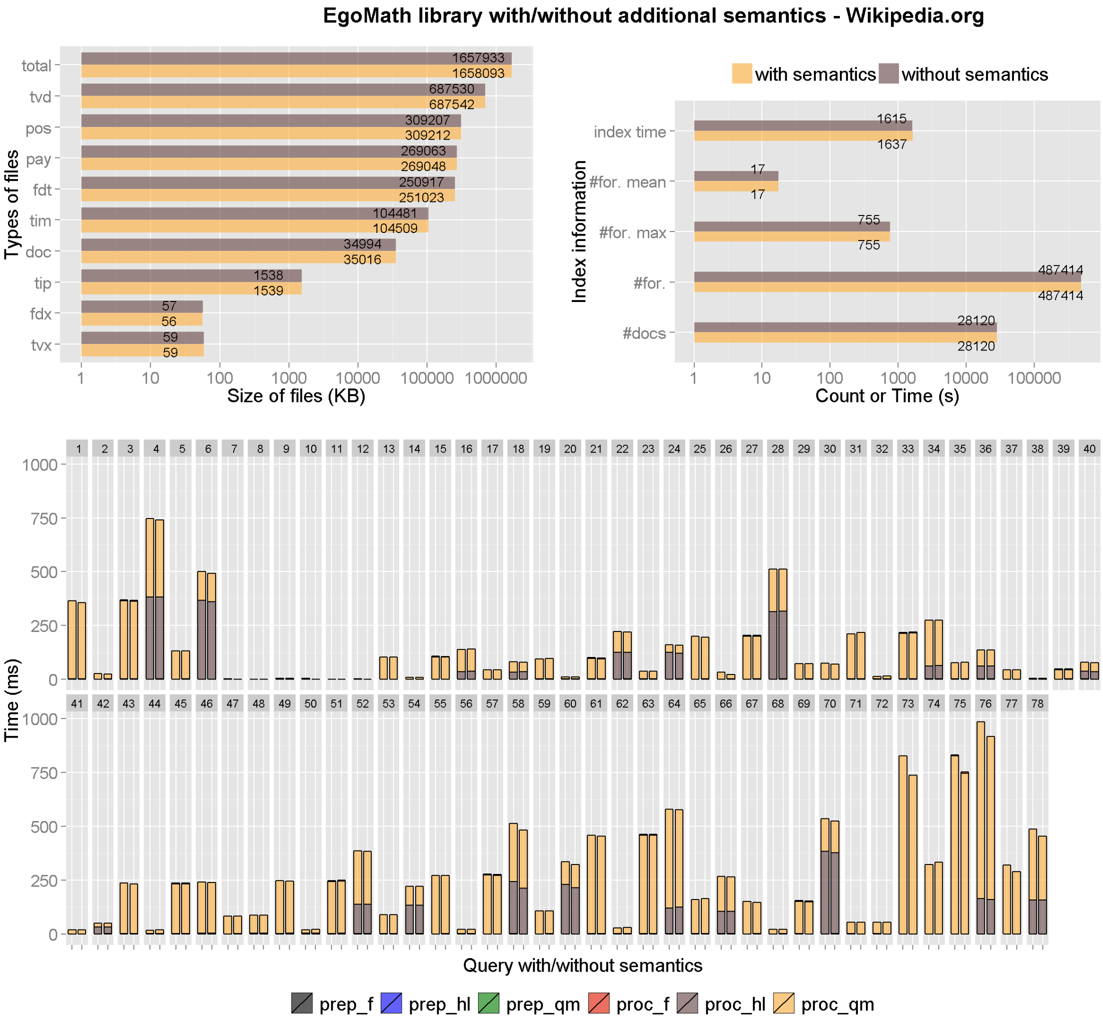


Figure 4.11: Performance of "with" and "without" additional semantics

One of the reasons is the fact, that only approx. 0.5% of formulae contained additional semantics according to our formula semantic parser. The mean query time difference is approx. 0.5% when removing queries 7-12 which are specific for the version containing semantics. We do not expect the performance to drop if the semantics are single words. However, if they are constructed of many words a simple solution is to regard them as one token (similar to using tokenisers as described in Section 4.9.9).

4.9.4 Index With vs. Without Payloads

EgoMath uses payloads to improve the quality of ranking. Payloads are stored for the first two representations for each augmentation algorithm. According to Figure 4.12, the index differs by 16.3% because most of the formula representations contain the first two.

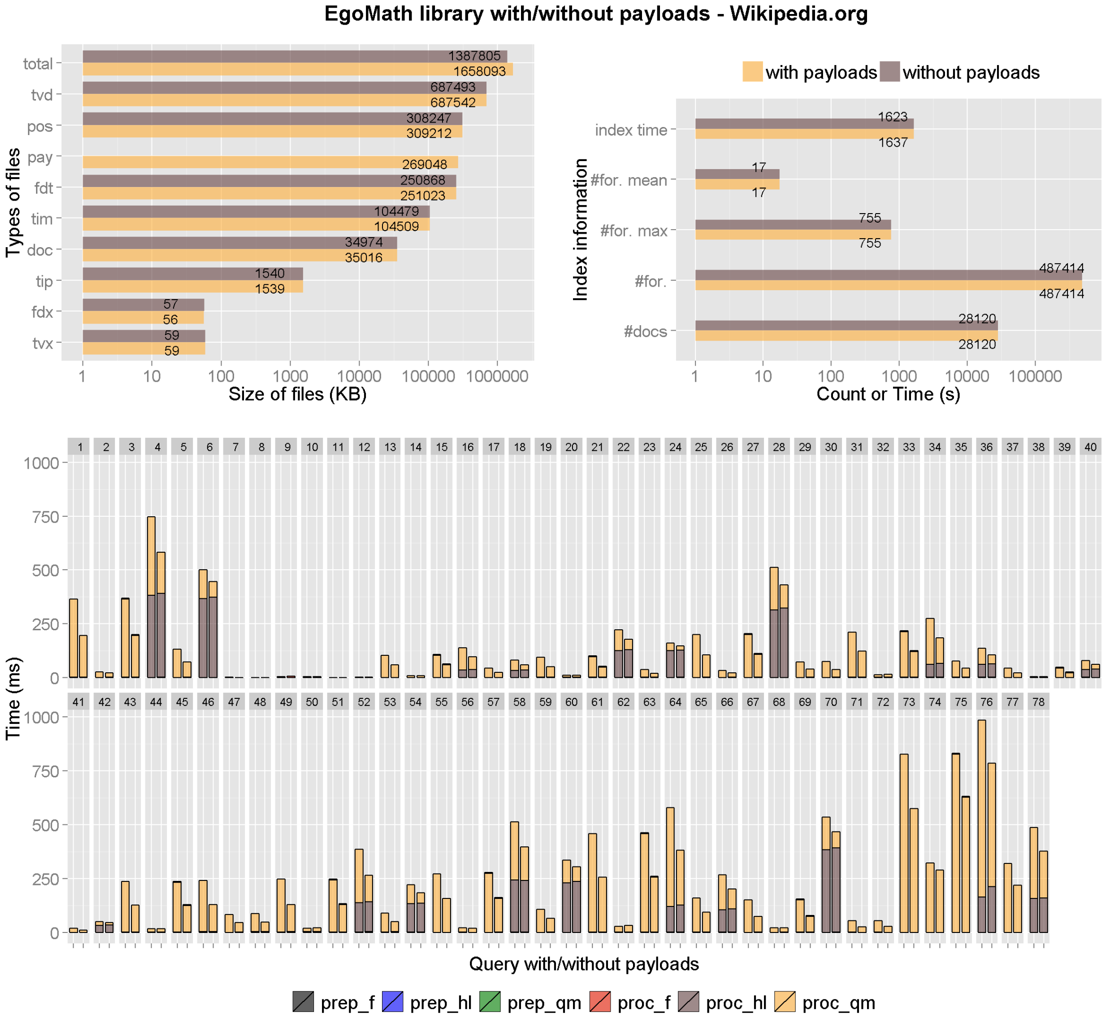


Figure 4.12: Performance of "with" and "without" payloads

Starting from the 2nd query, every 6th (queries 2, 8, 14, ...) does not use payloads and the query times of the *with payloads* and *without payloads* are very similar which is expected. The overall performance using *with payloads* is slower than *without payloads* by 28%.

It is very likely that the performance penalty increases in larger document sets because more relevant objects would have to be ranked. Better indication whether the payloads should be used or not in cases where the performance drops significantly would be to compile the test query set from the actual logs and compare the rankings too.

4.9.5 Query Time of Two vs. Three Sub-queries

In EgoMath v2, three sub-queries formed the actual mathematical query. Query searching for the whole formula, for sub-formula and query used to rank formulae which start with the searched query higher. EgoMath v3 removed the last sub-query because it was not adding anything to the query logic and gained approx. 25% in speed as shown in Figure 4.13.

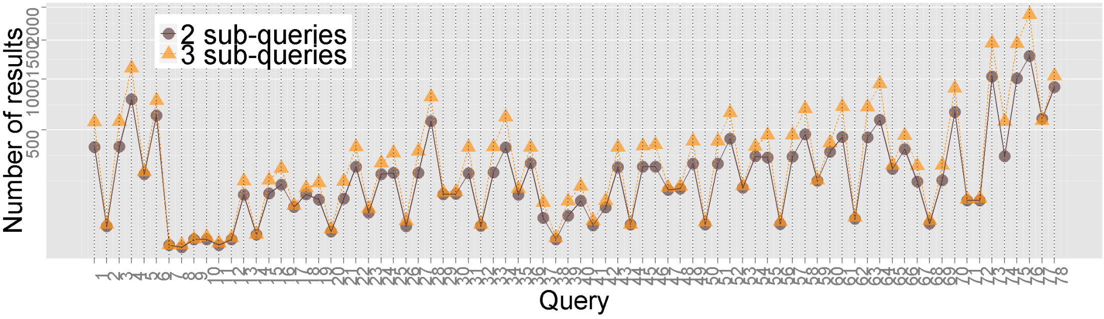


Figure 4.13: Query time performance of two vs. three sub-queries

4.9.6 Number of Results of the Evaluation Queries

The evaluation result depicted in Figure 4.14 shows the consistency of the search results across different settings. There are several important and interesting outcome.

The important outcomes show the effect of different settings. In query 7-12, the "without semantics" version returns 0 results which is expected because the query itself is using additional semantics. There are several other queries which return 0 results; queries 29, 30, 71 and 72 because they search only for whole formulae but there is no such formula in the document set. The last important outcome is that, the *without ego** keywords version returns 0 results in queries 61-66 ($F = G\frac{m_1 m_2}{egovar^2}$). The reason is that the query contains the "egovar" keyword which is not present in the index. EgoMath v3 returned two documents in queries 31 - 36 ($\sum_{n=0}^{\infty} \frac{f^n(a)}{n!}(x-a)^n$)) because the "infin" issue described in Section 5.3 was solved. Queries 55-60 ($E = mc^2$) show the effect of removing subscripts as described in Section 4.5.1 where EgoMath v3 returned more results.

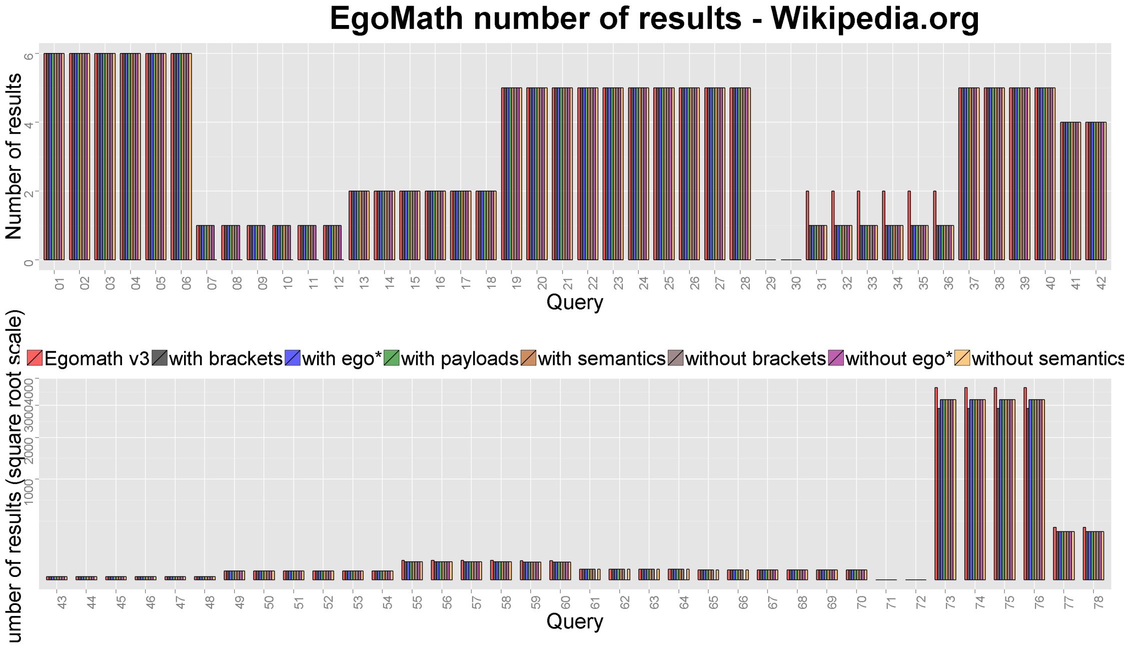


Figure 4.14: The number of results returned executing queries from Appendix F

An interesting outcome can be seen in queries 73-76 where the *with brackets* version returns different results to the others. The reason is already mentioned in Section 4.4 and is the varying number of operators in this case the operator $+$. Let us analyse EgoMath representations of the query $id + id$ and an example formula a^{+b}. The representation of $id + id$ is simply "id id +". The representation of a^{+b} is in the "with brackets" version "ego10 : const id { id + } ˆ +" and in the "without brackets" version "ego10 : id id + ˆ". It can be seen that "id id +" matches the latter one but not the former one. The queries 77 and 78 are not affected because they require the whole formula match. Possible solutions to this problem include forgetting the implicit $+$ or using the same technique as with unary $-$ where we represent it with a different symbol e.g., !+. EgoMath v3 solves it by adding *remove_optional* transformation (see Appendix I).

4.9.7 Query Time of the Evaluation Queries

The query times of a few specific versions compared to EgoMath v3 are shown in Figure 4.15. It is obvious that even with additional transformations and additional functionality, EgoMath v3 outperforms the version "with brackets" and is marginally slower than the other versions if the result list is similar. The only difference are queries 73-78 ($id + id$) where EgoMath v3 returns more results and is slower because of this.

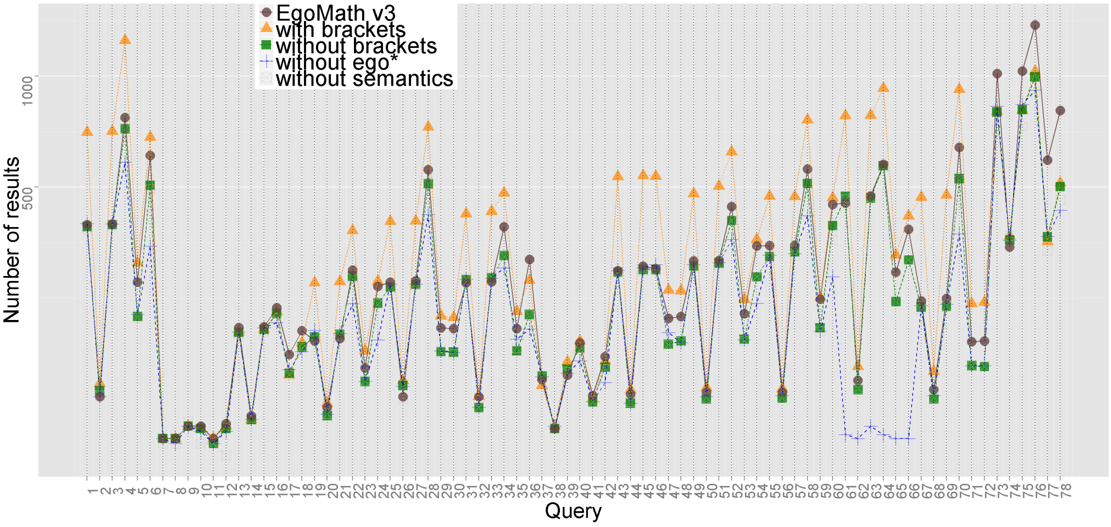


Figure 4.15: The total query time executing queries from Appendix F using different settings of EgoMath

4.9.8 Index All Fields vs. Not the Textual Field

The additional functionality of textual search in EgoMath does effect the index size significantly as shown in Figure 4.16. Only file types which contained a bigger amounts of data were included.

The textual field was indexed but also stored for easy retrieval and the highlighting functionality. The difference between versions with and without storing and indexing of the textual field was substantial; for the Wikipedia.org document set, the index size of the *all fields* was more than double and for the MREC document set, the index size was almost six times bigger than the version without. For the MREC document set, the "fdt" file type (stored fields) is 358GB compared to 50GB which is expected.

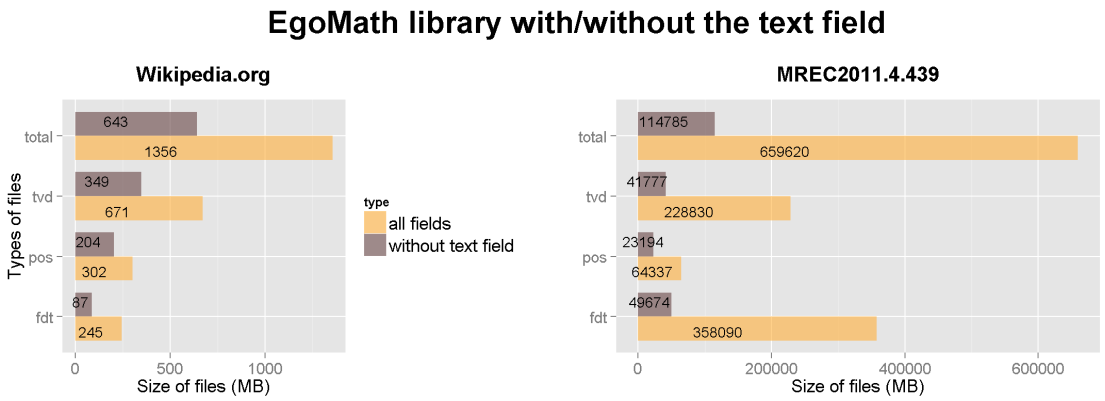


Figure 4.16: Performance of "with" and "without" the text field

There are several reasons why the difference between with and without the textual field for MREC document set was so big. Firstly, we store additional information for each term (e.g. position, offsets). Secondly, documents contain formulae in MML format which are not removed during indexing phase (which would be reasonable to do) producing many unique terms.

4.9.9 Revisiting Tokenisers

We revisit the tokenisers from the paper by Mišutka and Galamboš because there are several differences. Firstly, the original evaluation was done on a much smaller document set. Therefore, the distribution of formulae could have been very extreme e.g., that only very few formulae were textually similar. Moreover, the indexing structures did not have to be optimised and/or the difference was minimal.

The evaluation over Wikipedia.org showed that the tokeniser which allowed only for simple terms resulted in the largest index size. This is exactly the opposite result as in the original paper where it resulted in the smallest index size. This can be due to the fact that in EgoMath v3, each term position is stored and the tokeniser produces most terms. The inverted index row representing one token is big because there are many (matching) documents.

On the other hand, exactly the opposite holds true for the tokeniser which regard each formula as one term. This is expected when taking the above result into account.

In general, the difference to the original paper is also very likely due to the different underlying search servers used (Solr vs. Egothor) and the fact that the originally tested document sets were much smaller.

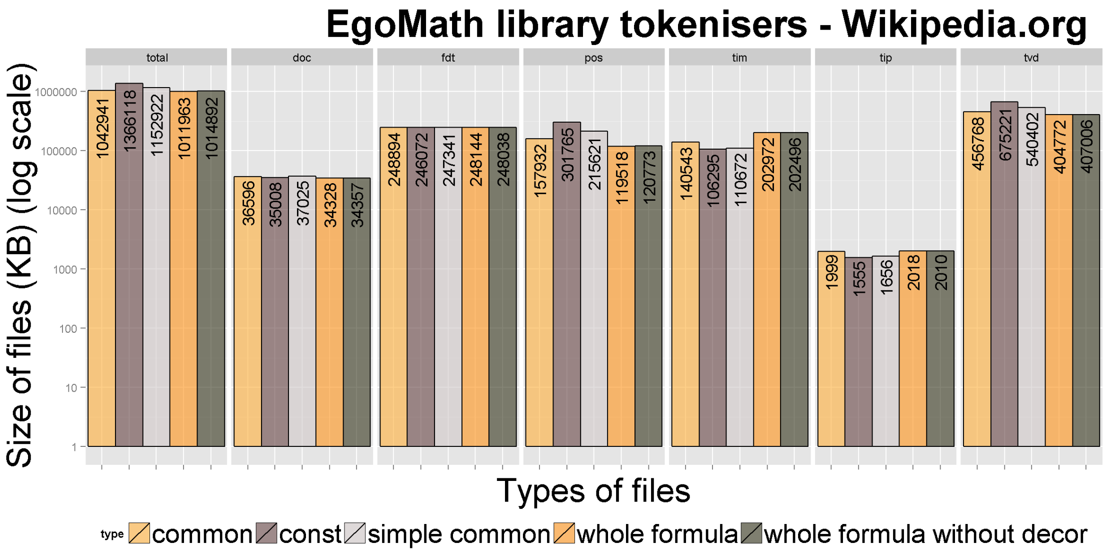


Figure 4.17: Revisiting different tokenisers on Wikipedia.org. The names of the tokenisers from the original paper are as follows: *all* (whole formula), *alls* (whole formula with decor), *com* (common), *simple common* (coms), *const* (const). The *num* tokeniser from the original paper was not used because it is very similar to *const* tokeniser.

4.10 Stand Up to the Giant

One of our goal is to build a mathematical search engine capable of indexing very large amounts of data. Using state-of-the-art full text search engine gives us a good change to succeed. Moreover, we could employ horizontal and vertical scaling which could solve a problem if the performance is not acceptable. However, we do not think that there is a consistent collection of mathematical documents which really requires distributed approach in respect to full text search engines. Therefore, horizontal and vertical scaling outside of the scope of this thesis.

We will describe the problems which can happen and their possible solutions. We will follow the process of indexing the MREC2011.4.439 document set by EgoMath. The document set contains $439,423$ files out of which approx. $433,000$ contain at least one valid mathematical formula. The size of the document set 121GB. We use the testing queries from Appendix F. All tests are performed on the same set up as described in Section 4.9.

The first approach is to index the document set with EgoMath v3 used to index Wikipedia.org. The result index size is approx. 660GB with further details shown in Figure 4.16 (EgoMath v2 used approx. 630GB). Simply said, having large index requires keeping many pointers and accessing disk randomly. We do not show the actual results because the queries took from tens of seconds up to minutes. After investigating, we found the reason to be the extreme distribution of common mathematical elements e.g., simple numbers (0-9), common functions ($\sin$, $\cos$, etc.), common variables (a, x, i, etc.) and EgoMath ’s specific keywords (id, $const$, $ego*$).

Let us inspect a simplified example. Searching for $a+b+c$ requires that a document contains a, b, c, $+$. Documents in MREC collection contain 360 formulae on average. When a document contains all four terms, the occurrences of those terms must be fetched for all the 360 formulae and must be compared together. But when a document has significantly less formulae, it is likely that more documents will be skipped already in the processing of inverted index which is much faster.

In respect to full text search engines, common words are often discarded in the preprocessing phase and are called “stop words”. The impact on performance depends on the document set. One possible solution to this problem, from the full text search engine world, is to index “shingles” (word n-grams[33]) instead of simple words.

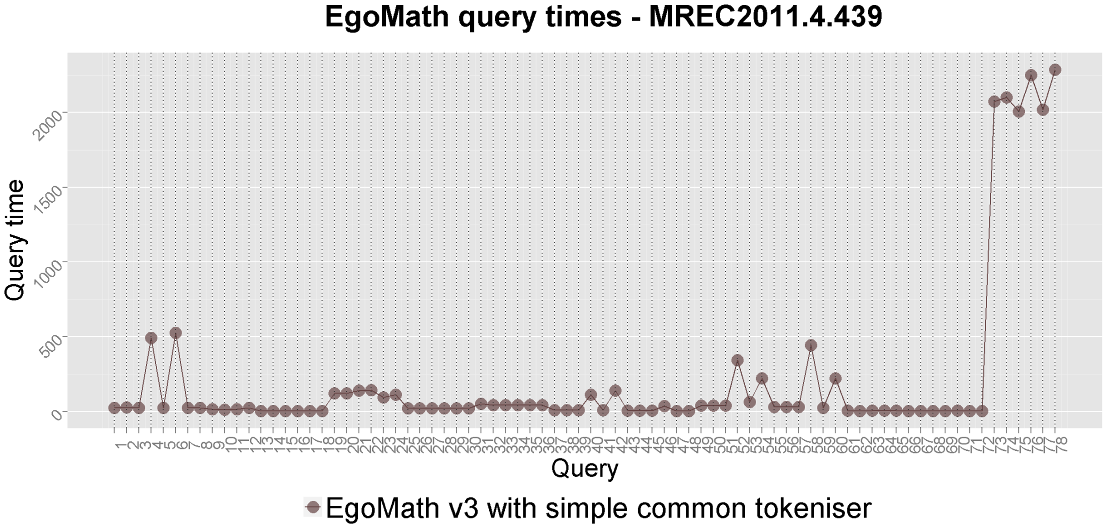


Figure 4.18: Query times of EgoMath v3 with simple common tokeniser over MREC2011.4.439

Raw shingles cannot be used directly because they would break the correctness of sub-formulae; however, we already have the solution in *tokenisers* (see Appendix J). Tokenisers are used to defining the basic information units which can be searched. The results after indexing with *simple common* tokeniser (see Section 4.9.9) are shown

[33] A word n-gram is a contiguous sequence of n words from a given sequence of text.

in Figure 4.18. The query times are outstanding (in respect to the document set size) with only the queries 72-78 ($id + id$) performing slightly above 2 seconds which is expected as this token is very frequent.

4.11 Contribution and Conclusion

EgoMath mathematical search engine was presented in this chapter with focus on algorithms which improve mathematical awareness and the overall quality. While building EgoMath, we implemented a parallel resource importer framework which can be also used with other search engines. We have shown that full text search engine can achieve interesting results in respect to mathematical searching using several *novel algorithms* and *paradigms*. The performance was evaluated on a well known and publicly available document set which represents common mathematical knowledge. All the tests, including results, are described in detail and can be easily repeated. Finally, we describe how to index a huge document set without using horizontal or vertical scaling on a mediocre machine.

EgoMath represents state-of-the-art mathematical searching in real-world documents offering an extensible level of mathematical awareness using recent improvements in full text searching.

CHAPTER 5

Feature Based Mathematical Search Engine

The evaluation of EgoMath has focused on various details and specifics we regarded or found to be important. There have been several interesting approaches to mathematical searching described in the last few years which are very different to the algorithms used in EgoMath. We think that a proper evaluation of mathematical search engines (MSEs) should either use an accepted test collection (which is not available) or be a direct comparison with another approach.

To overcome this restraint[1], we decided to implement another search engine, building on the work by Ma et al. described in the paper "Feature Extraction and Clustering-based Retrieval for Mathematical Formulas" [Ma+10a]. Ma et al. claim to achieve "promising results" by proposing an "effective feature extraction approach" and supporting their claims with an evaluation done on 884 formulae. We chose the feature based extraction algorithms proposed in the paper because we think that feature based similarity research has potential. We will refer to our implementation of the feature based algorithm search engine as FBA.

We have extended the original algorithms proposed by Ma et al. and implemented them inside the Importer framework using formula parse trees, normalisation and ordering algorithms from the EgoMath library. The approach has been studied and evaluated on a real-world data set. We introduced several improvements and tested three different versions of the original algorithm with several interesting results. Finally, we compared the performance with EgoMath using a set of test queries.

[1] It was not possible to use other approaches mainly because they were not mature enough for complete evaluation workflow or the document set was not publicly available.

We will begin with a description of the proposed algorithms by Ma et al. Then, we will describe our contributions to the original algorithms in detail including evaluation and the actual implementation. We will analyse the comparison with EgoMath in the last section.

5.1 Original Algorithm

The original proposed solution relies on the Presentation MathML format used for formulae. Firstly, we describe the important Presentation MathML tags used in the algorithms:

- *<mo>* - represents an operator; besides operators in the strict mathematical meaning, this element also denotes "operators" (e.g., parentheses), separators (e.g., comma) or specific operators (e.g., "absolute values");
- *<mi>* - indicates that the content should be rendered as an identifier such as function names, variables or symbolic constants;
- *<mn>* - denotes a numeric literal which is normally a sequence of digits with an optional separator;
- *<msup>*,*<msub>* - is used to bind a superscript or subscript to an expression;
- *<mrow>* - is used to group sub-expressions, which usually contain one or more operators with their respective operands such as <mi>or <mn>.

It should be explicitly noted that the semantic meaning of formulae is not necessarily mathematically correct in the original paper, but the meaning is guessed from the common mathematical markup e.g., $\int$ is considered to be the symbol for integration and e is a well-known constant. This can lead to the extraction of incorrect features; however, this correlates with the approach used in EgoMath to normalise the meaning of symbols, arguing that when the same normalisation is performed during the indexing and searching phase, then the recall measure will either increase or stay the same.

There are two types of features extracted from a formula: 1) structural and 2) semantic ones. Semantic features are extracted by traversing the formula tree in preorder. There are operators (<mo>), functions inside identifiers (<mi>) and nodes

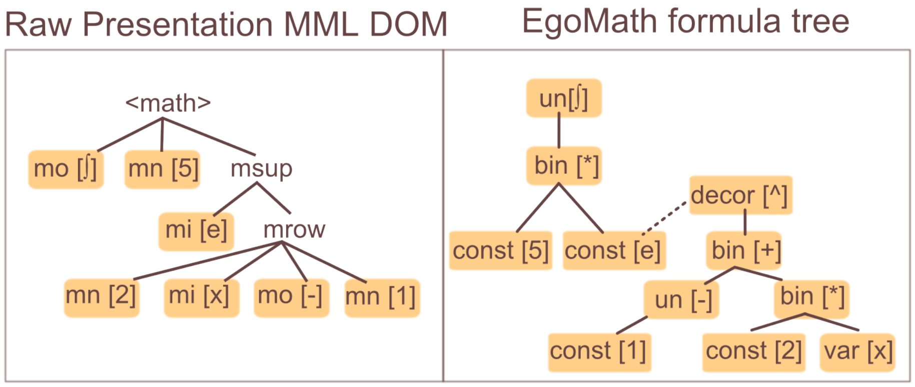


Figure 5.1: Different representation of formula trees. The optimistic Ma et al. raw DOM MathML representation is on the left. EgoMath's representation of $\int 5e^{2x-1}$ is on the right (un - unary function, bin - binary function, const - constant, decor - decoration i.e., upper and lower index, var - variable).

(except <mrow>) that have children containing identifiers (<mi>) extracted. Structural features are obtained from nodes containing semantic features, their depth (in the formula tree) is ≥ 1. Concatenating the names of its parent nodes represents the structural meaning.

Let us consider the formula $\int 5e^{2x-1}$ with MML representation (shown on the left side in Figure 5.1) consisting of these tags and their appropriate values *mo*:$\int, -$, *mn*:$5, 2, 1$, *mrow*, *msup*, *mi*:e, x. We obtain these semantic features by using the semantic feature extraction algorithm from the original paper: $\int$ (mo), e (mi + exponential function[2]), $msup$ (identifier in children), $-$ (mo). After extracting the semantic features, we can use the second algorithm to extract the structural features which are ("/" is used as a separator between individual node names): $msup/mrow/-$ ($-$ is a semantic feature and the rest was obtained by concatenating the values by recursively visiting the parents), $msup/e$[3].

The last set of features represents constants and variables. The original paper claims that "Unlike operators or functions, both number constants and variables are not representative, so their semantic features may not be meaningful" and that "It is obvious that exact value representations are not meaningful, e.g., 12 or x, for formula search purposes". These are rather strong and subjective claims which we discuss in more detail below. The real values of the numbers and constants are replaced by var and cn keywords and the same algorithm as for the extraction of the structural

[2] This particular guess of the semantic meaning in the original paper is problematic because e is, to be precise, a constant and not the exponential function which is e^x.

[3] This is incorrectly written in the original paper as $e/msup$.

features is used. The result features are *msup/mrow/cn* and *msup/mrow/var*[4].

Extracted features are transformed into a feature vector using the tf-idf weighting scheme and normalised using cosine normalisation. First, the weights are computed using the term frequency, defined as

$$tf(feature, formula) = \frac{\text{\# of occurences of feature in a formula}}{\max\{\text{\# of occurences of any feature in a formula}\}}$$

and the document frequency defined as

$$idf(feature, formula\_set) = \log \frac{\text{\# of formulae in a formula set}}{\text{\# of formulae which have the particular feature}}.$$

The result vector $\vec{F}$ is normalised to $\vec{F}_{norm}$ with

$$\vec{F}_{norm} = \frac{1}{\sqrt{\sum_{i=0}^{i=n} f_i^2}} \vec{F}$$

where n is the number of features and f_i is the feature at a particular position. For a small data set, the feature vector can contain all distinct extracted features from all formulae. These normalised feature vectors are used for clustering. The similarity between a query formula and a formula from the data set represented by a normalised feature vector is the well-known cosine similarity (see definition Equation 2.2.1). The original paper evaluated three different clustering algorithms for retrieval: K-means, self organised maps and agglomerative hierarchical clustering. However, we will not use clustering because it does not directly add to the mathematical awareness. The original evaluation was done on 884 formulae with 20 training and 20 test samples. A more detailed description of the extraction algorithms is available in the original paper [Ma+10a].

5.2 Modifications

On one hand, the contribution of the original paper by Ma et al. [Ma+10a] is clear. On the other hand, we must point out several problems we encountered which question the feasibility of the original approach to larger data sets. We try to solve these issues and propose a modified version of the algorithms.

[4] Another mistake in the original paper states that $msup/mrow/var$ was produced by $msup/mrow/5$ but it could only have been produced either by $msup/mrow/2$ or $msup/mrow/1$.

We evaluate the approach on a real-word document set - Wikipedia.org. The Wikipedia.org document set contains formulae encoded in LaTeX. Each formula is parsed into the formula tree using the EgoMath library and the *ordering algorithm* is applied (see Appendix J). Features are extracted from the formula tree.

It should also be noted, that the example in the original paper uses an optimistic representation of the original formula $\int 5e^{2x-1}$ in the MML presentation format. The "tex2xml" converter service produces a representation with two additional $mrow$ elements making the features different and longer. By using normalised EgoMath's formula trees we avoid most of the notation differences which clearly improves the quality in comparison with the original algorithm. The difference between MML representation and the formula tree is illustrated in Figure 5.1.

The original algorithm produced $271,103$ unique features using formula trees over the whole of Wikipedia.org. The precision of this approach with the evaluation set up was poor (with most of the evaluated queries having zero precision). The reasons are that variables and constants were considered unimportant and many relevant features were missing. This was the motivation to introduce limits on the extraction algorithms to lower the number of unique features.

The overall idea is the same as in the original paper. The extraction and concatenation of the names of parent nodes is replaced by a more complex algorithm which: skips binary functions with the same node depth (e.g., $a + b - c$ all leaves have only one parent whose value is extracted), can remove common nodes like $*$, can limit the number of parents traversed and can replace numbers and variables with keywords. 23 out of the 50 most used features contained "*". We added many "*" through ab to $a * b$ normalisation; because of this, and because of the limits set to the feature extraction depth, we added the option to skip "*" nodes when extracting the values from parents.

The algorithm for extracting semantic features is in Listing 5.1.

Listing 5.1: Semantic feature extraction implemented in Java

```java
private static boolean has_semantic( INode node ) {
  // 1. operator => feature
  // 2. function => feature
  if ( INode.is_op( node ) || INode.is_fnc( node ) )
    return true;
  // 3. node containing variable in children => feature
  Iterator<INode> it_inner = new Formula( node ).iterator( algorithm.
      DEPTH, true );
  while( it_inner.hasNext() )  {
    if ( INode.is_var( it_inner.next() ) )
      return true;
  }
  return false;
}

private List<String> extract_semantic( Formula f ) {
  // ...
  Iterator<INode> it = f.iterator( algorithm.DEPTH, true );
  while( it.hasNext() ) {
    INode node = it.next();
    if ( has_semantic( node ) ) {
      String content = get_normalised_content( node, false );
      features.add( content );
    }
  // ...
```

The structural algorithm is in Listing 5.2.

Listing 5.2: Structural feature extraction implemented in Java

```
private List<String> extract_structural( Formula f ) {
  // ..
  while( it.hasNext() ) {
    INode node = it.next();
    if ( node.depth() > 1 && has_semantic( node ) ) {
      String ftr = extract_structural_path( node.parent(), node.depth(),
           true );
      if ( 0 < ftr.length() )
        features.add( ftr + INode.get_content(node) );
    }
  // ..
```

We define three algorithms with these settings: 1) FBA 1 - skips $*$ from traversing and replaces constants and variables; 2) FBA 2 - skips $*$; 3) FBA 3 - neither replaces any values nor skips $*$.

The final semantic features extracted from $\int 5e^{2x-1}$ by FBA 3 are "$\int$, $*$, e, $\hat{}$ (upper index or msup in MML), $!-$ (unary -), $+$, $*$" (in comparison to "$\int$, e, $\hat{}$, $-$" from the original paper). The final structural features extracted are $int/*$, $int/*/e$, $int/*/e/\hat{}$, $e/\hat{}/+/!-$, $*/e/\hat{}/+$, $e/\hat{}/+/*$, $int/*/5$, $int/*/e$, $\hat{}/+/!-/1$, $e/\hat{}/*/2$, $e/\hat{}/*/x$ (in comparison to $\hat{}/e$, $\hat{}/mrow/-$, $\hat{}/mrow/cn$, $\hat{}/mrow/var$ from the original paper).

The algorithm for extracting numbers and variables visits all nodes and if the node is either a constant, a number or a variable extracts a path similar to the structural extraction. The feature is then appended to the feature vector. In contrast to the original algorithm, we also extract features from nodes with a depth equal to 1 because in the formula tree, a root node is part of the formula.

We claim that our changes and usage of EgoMath's formula trees improve the quality of the extraction algorithms in general. We base our claim on several case studies. Firstly, for formulae having only one element, like 5, sin would have zero features in the original algorithm but in our approach they have one feature - the node value (or replaced keyword) itself. Mathematically equal (in common mathematical structures) formulae $-2 + a$ and $a - 2$ have different feature sets in the original algorithm but equal feature sets in our algorithm because of the ordering algorithm being applied. The same holds for $2 * x$ and $2x$ due to the normalisation algorithms used. Furthermore, 1^x and x^1 have equal feature vectors in the original algorithm but - correctly - different ones in FBA because of the more mathematically precise formula trees. We overcome MML representation which does not respect operator priority, and therefore introduces additional $mrow$ elements, by using the formula trees an skipping same priority operators when traversing to the root node.

The main algorithm works with a vector of numbers where each position represents a different feature. Finding the relevant features which should be used to represent every formula in the data set requires either the processing of the whole data set or at least a representative subset. Afterwards, a set of U distinct features is selected and is used to represent each formula.

According to the statistics of the Wikipedia.org document set in Section 4.3.1, 90% (approx. $261,618$) of formulae are shorter than 100 characters, including spaces and brackets etc. and 70% are shorter than 50. Despite this fact, the number of unique features using the original feature extraction algorithms over EgoMath's formula trees was in the hundreds of thousands as shown in Figure 5.2.

In the first run, we try to reduce the number of unique features by processing only formulae smaller then M characters. Furthermore, the average number of features extracted for the three feature types were 8.2, 5.8 and 8.5 respectively which gives the first estimation for the number of the representative feature count. We also limit the maximum number of extracted features for each type by N. Finally, the maximum number of parents visited while concatenating the node names was limited by O. The depth limit set to three means that the formula depth is limited to four nodes in child-parent relations but they must also have different depths (otherwise, they would be skipped). All formulae with a bigger depth have some features stripped. A different depth means that operators on the path to the root node in the formula tree have different priorities.

The number of features extracted with different parameters is depicted in Figure 5.2. Finally, we decided to further evaluate $M = 100$, $N = 20$ and $O = 3$ because the number of features considering all FBA algorithms was acceptable compared to the others and was more than double the average number of features. The total number of extracted features using these settings with FBA 3 was $239,675$ ($4,795,780$ features were extracted all together) but only $28,539$ using FBA 1. The FBA 1 number of features ($28,539$) is significantly lower than the number of extracted features by the original algorithm ($271,103$).

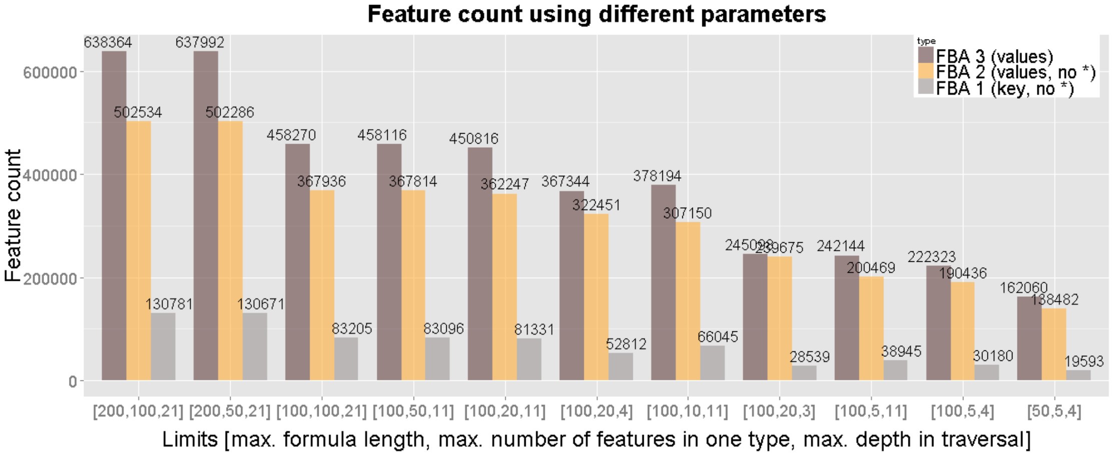


Figure 5.2: Number of features using different limits

We take the first $10,000$[5] features sorted by the count of occurrences in the whole data set. It is reasonable to do so because the number of occurrences of the first 30 features dramatically falls, as can be seen in Figure 5.3, and the features become relevant. Starting from the 29th feature (sorted by the number of occurrences), the occurrence count is in the order of thousands. Starting from the 412th, in the order of hundreds and starting from 3346th, in the order of tens. We have experimented with a lower feature count (100, 1000, 5000) but the precision of the result was smaller. On the other

[5] A count of $10,000$ features was used because it is common for the similarity search performance framework we worked with.

hand, larger feature sets ($15,000$, $20,000$) have not improved the evaluated precision. Using $100,000$ feature count introduced significant performance penalties (and little precision improvement). Because we focus on applicability we will not consider feature count above $10,000$.

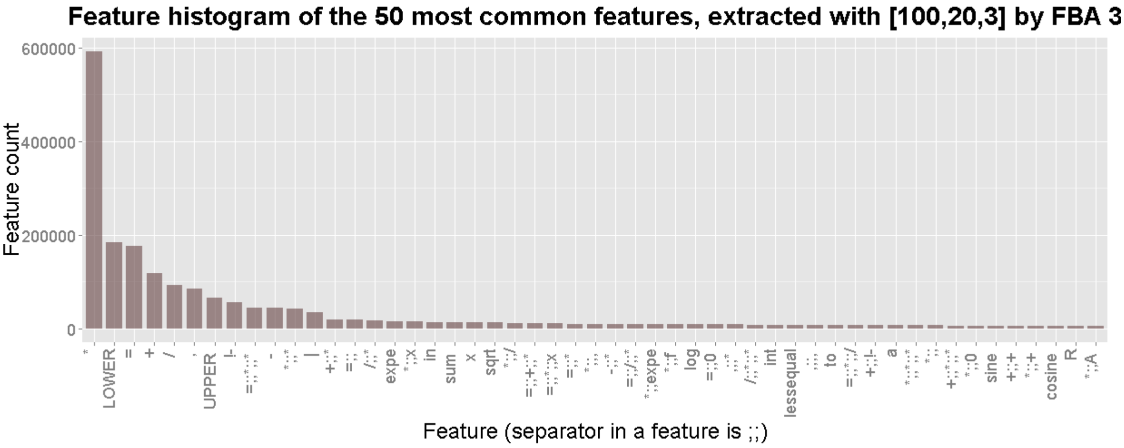


Figure 5.3: Feature histogram of formulae in the Wikipedia.org document set

Before we discuss the performance of the retrieval system, we should investigate the underlying data set. The clusters can be either so big that a near sequential scan is needed or so small that many valid results are skipped. However, clustering algorithms, like those ones mentioned in the original paper, are out of the scope of this thesis. We focus on the performance of similarity based searching. The following section briefly describes the data set and its indexability as defined in Section 2.2.1.

5.2.1 Examining the Indexability of the Wikipedia.org Document Set

We want to perform similarity searching of mathematical formulae relying on the extracted features. A trivial choice is a sequential computation of similarity of all the items in the document set. However, we want an applicable solution so we investigate the performance limits. Intrinsic dimension theory cannot be directly applied to our FBA using cosine similarity. The problem is that cosine similarity is not a metric function. However, it is easy to use angular similarity based on cosine similarity which is already a metric function. Angular similarity for vectors with positive only values is defined as

$$1 - (2 * \frac{cos^{-1}(similarity)}{\pi})$$

where the similarity of two vectors A and B is defined as

$$similarity = cos(\theta) = \frac{A.B}{||A||\ ||B||}.$$

Applying the theory of intrinsic dimensionality to the Wikipedia.org document set with our enhancements and implementation of the feature extraction algorithm gives us Figure 5.4. Intrinsic dimensionality using FBA 1 on an random subset of 5000 formulae with $M = 100$, $N = 20$ and $O = 3$ is 12.81 which means the distribution should allow for acceptable separability, and in a general case, using metric indexing techniques should greatly improve the searching speed. Intrinsic dimensionality using FBA 3 is 10.3.

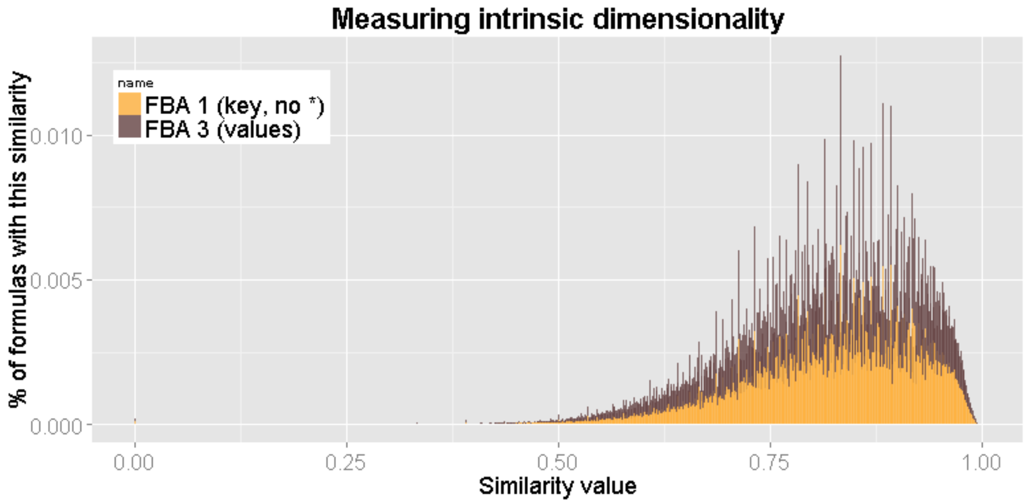


Figure 5.4: Feature-based angular similarity distribution histogram of formulae from Wikipedia.org document set from 5000 random formulae. The non-similar values (distance is 1) are omitted improving the visualisation.

5.3 Evaluating and Comparing with EgoMath

Let us first consider the actual retrieval task we want to cross-evaluate. The outcome of this evaluation should be the comparison of two approaches used for mathematical searching. It is not necessary nor feasible to state the absolute performance numbers because we also think that, for this purpose, we would need one of the user-oriented measures (or side-by side evaluation) [BaRi11] with a representative number of evaluators.

We want to overcome the common problem with trivial or very small evaluation document collections and also with the lack of representative evaluators. We will use mathematical documents from Wikipedia.org as extracted using the Importer framework. This data set contains approx. $28,100$ mathematical articles with $290,872$ unique formulae, out of which, $261,618$ are valid for feature extraction having textual length smaller than 100. Out of these, $258,404$ have non empty feature sets (e.g., ")" and "$d\_$" have empty features).

The well-known precision and recall measures have been extensively used to evaluate the retrieval performance of IR algorithms. However, these measures have several shortcomings which apply to our problem as well. First, the proper estimation of maximum recall requires detailed knowledge of all the documents which is not possible for our evaluation data set - Wikipedia.org. Secondly, it is desirable to measure the performance of a single query rather than a whole set, so individual strengths are visible. We will use P@1 and P@5[6] measurements and the query rank visualisation (used by the Spearman Coefficient correlation measure) will be used for visualisation of behaviour showing the correlation (if any) between the compared algorithms.

Set-up and Results

The evaluation was performed on a Fujitsu Esprimo E900 with 4 Cores at 3.4 GHz with 12 GB of RAM running Windows 7/64-bit with Python 2.7.2/32-bit (because of JNI integration problems).

We optimised the solution to work with packed arrays and a numeric index of the actual features. This reduced the index size by 60% and the query times by 40%. The performance of all queries using non-parallel execution of the top-k algorithm was between 4-7 seconds with an average of 5.1 second[7].

The evaluation was done by performing a set of queries on both search approaches. The same arguing was used as in Section 4.9 for the selection of the testing queries. The complete list of queries and results can be found in Appendix G.1 and G.2.

The result P@1 and P@5 precisions are depicted in Figure 5.5. The decision whether a result is relevant was made by three people and can be easily verified with the original results available in Appendix G.1 and G.2. The table shows the original query, results returned by EgoMath and the results returned by the individual FBA. The re-

[6] Precision ($\frac{|\{\text{relevant documents}\}\cap\{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|}$) takes all retrieved documents into account, but it can also be evaluated considering only the topmost results returned by the system.

[7] Preliminary testing with a similarity search framework, provided by the SIRET research group [SRG], showed that the framework was more than 3x faster than the sequential scan.

sults are sorted according to the ranking. A horizontal line in the EgoMath results marks the results returned when searched for higher level of abstraction using the *abstract math* feature. For the P@5 measurement, the first n results were used where n is the minimum of the returned results by both approaches without the *abstract math* feature (e.g., $n = 1$ in query 1 and 3). Because EgoMath for Wikipedia.org searches for documents instead of mathematical formulae, it can happen, that one formula is returned multiple times in different documents with other results between them. In this case, the returned result contains multiple occurrences of the same hit (e.g., query 6) but is not considered in computing precision. More importantly, we consider only the first found (highlighted) formula from each document in EgoMath. This is why documents which contain more similar formulae can lower the precision for EgoMath.

FBA algorithms returned many equal formulae with the same high similarity at the beginning, proving that the formula trees improved the quality e.g., "\pi = c / d", "\pi = \frac{c}{d}". Because the same happens in EgoMath, we treat these formulae as one.

The comparison is shown in Figure 5.5. In P@1, the precision is either 1 or 0. EgoMath's precision is 100%, FBA 1 returned non relevant results in query 2 ($\pi = c/d$), query 4 ($\frac{d}{dx}e^x$), query 6 ($Ax = \lambda x$), query 9 ($E = mc^2$) and query 10 ($cos^2(x) + sin^2(x)$). The problematic queries are those which rely on the actual values of elements because there are too many structurally equal ones. The same applies to P@5. This issue is fixed by FBA 2 and FBA 3, and as can be seen, both outperform FBA 1. Only query 4 proved to be problematic because it is too simple and the relevant features were not in the selected $10,000$.

As mentioned above, several differences in the returned results between EgoMath and FBA were because they were in the same document e.g., query 8 FBA found $\|v + u\| \leq \|v\| + \|u\|$ which EgoMath did not because it is in the same document as $\|x+y\| \leq \|x\| + \|y\|$.

The results in Figure 5.5 are surprisingly outstanding for FBA 2 and 3 given the fact that only around 4% of all features were used to characterise each formula. However, we must stress that there are cases when a formula has important features not present in the first $10,000$ features like in query 4 (or even that there is none).

Despite the precision penalty put on EgoMath because of the evaluation details, it is slightly better than FBA 2 and FBA 3. However, it must be noted that the "less" mathematical approach to searching did perform very well and also found interest-

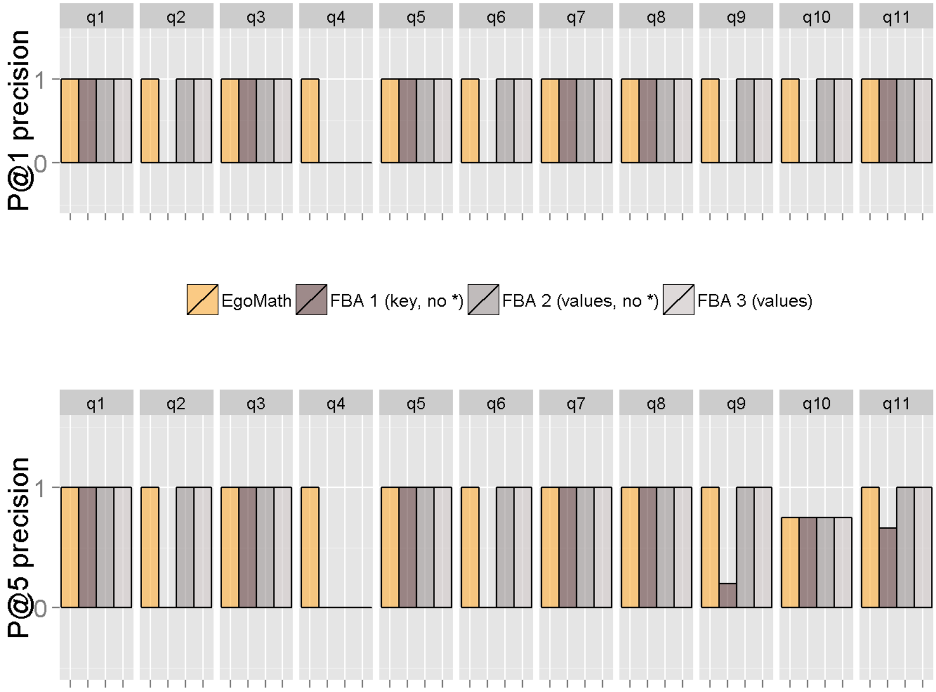


Figure 5.5: P@1 and P@5 precision comparison between EgoMath and FBA algorithms

ing similarities e.g., query 5 FBA found $\sum_{n=0}^{\text{infin}} \frac{f^n(a)}{n!}(x-a)^n$ which EgoMath did not because "infin"[8] was not recognised as a constant (which has been fixed in EgoMath version 3 used in final evaluation). Furthermore, several indices were disregarded and interesting "similarities" were found e.g., $\sum_{n=0}^{\infty} \frac{f^n(a)}{n!}(x-a)^n$ and $T(x) = \sum_{n=0}^{\infty} \frac{f^n(x_0)}{n!}(x-x_0)^n$. The last example was a motivation for extending EgoMath with another abstraction algorithm described in Section 4.5.1.

The EgoMath and FBA 3 correlation based on the query results is shown in Figure 5.6. In most cases the first returned hit was the same but otherwise, even though the precisions are almost equal, the returned lists are different. The cause from the EgoMath point of view is the already mentioned one hit per document in some cases. In FBA, it is caused by disregarding common subscripts because of low weight (e.g., query 9 $E = mc^2$, $E_0 = mc^2$, $E_0 = m_0c^2$) and due to missing features (e.g., $cos^2(x) + sin^2(x)$, $\cos(t)^2 + \sin(t)^2 = 1$).

[8] Wikipedia.org accpets non standard LaTeX commands e.g., \R, \infin, \Alpha. In order to support these commands, they must be defined in EgoMath's symbol definition as alternative.

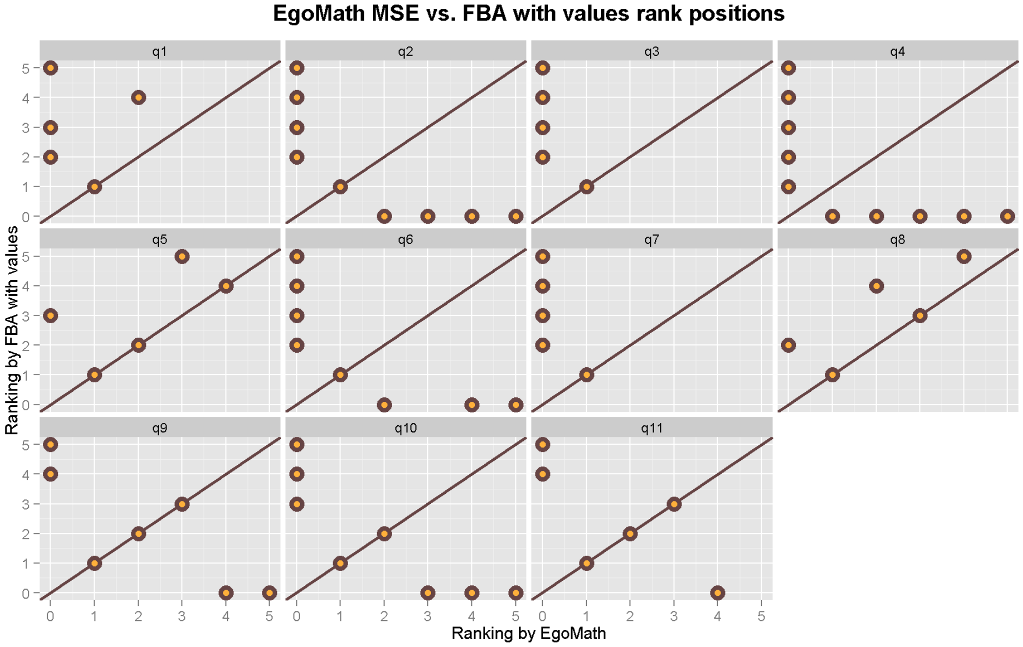


Figure 5.6: Graphical representation of query result correlation between EgoMath and FBA with values

5.4 Contribution and Conclusion

In this chapter, we have presented an improved implementation of the feature based search engine originally proposed by Ma et al. which can search for mathematical formulae. We have shown that the original algorithm extracts far too many features and introduced boundaries which limit the number of extracted features. We found out that, for our testing queries, using 4% of all features extracted by FBA 3 was enough for obtaining interesting results on our testing data set. Furthermore, we have shown that relying on EgoMath's formula trees rather than on the Presentation MML makes it possible to find many notationally equal formulae.

To the best knowledge of the author, we have done the first comparison of two different mathematical search engines; moreover, the comparison was done on a data set which is not artificial, is big enough to be representative and is intuitive.

We have shown that feature based retrieval can be used for specific mathematical searching in data sets with a size similar to Wikipedia.org in respect to both performance and quality. Additional testing is required for very large document sets. In parallel to the performance measures used, the actual results showed in a side-by-side table in Appendix G.2 are very interesting. Some of the results have been the motivation for improving EgoMath.

We have made several contributions. First, we verified the validity of the feature extraction approach in mathematical searching by evaluating it on a real word data set. This was possible due to our improvements which improved the quality and made the algorithms robust enough. Finally we described the comparison with EgoMath on a query set of well known mathematical formulae.

CHAPTER 6

Conclusion

We have focused on mathematical searching in this thesis. We have described the EgoMath mathematical search engine in detail. In order to build a flexible system, we have created a parallel resource importer framework and built a database of mathematical formulae, including different representations containing approx. 50 million unique formulae. EgoMath uses a new indexing technique which creates multiple representations of one formula according to a predefined set of transformations. These transformation rules define the mathematical structure where the indexed formulae live and where they can be searched for. Several novel algorithms and techniques have been proposed and implemented allowing for mathematical awareness in full text search engines. We showed various aspects and implications of different techniques and that our approach has deterministic results. We have used a large representative mathematical knowledge base - mathematical articles from English Wikipedia.org. In this respect, the goal **G2**, stated in Chapter 1 has been fulfilled.

There had not been any comprehensive overview of approaches and techniques used in mathematical searching. We have filled this gap with an extensive survey which we set as our goal **G1**.

In order to fulfil the goal **G3**, we have built another mathematical search engine based on features extracted from mathematical formulae based on work by Ma et al. We found out that it is not applicable for large document sets and for real queries. We have proposed important improvements and were able to obtain interesting results. Finally, we have compared this approach with EgoMath in the first ever cross-evaluation of mathematical search engines based on different approaches, including the first five results for each tested query.

Future Work

We think that a wider accepted evaluation process flow including larger document sets should be the next step. It should define the state-of-the-art quality and results which can be used as the basis for evaluations in the future.

Despite our efforts, we had little success with convincing publishers and other institutions to grant us access to their document sets. We think that tighter collaboration in this community is needed to promote the achieved results more effectively.

Bibliography

[AbSt72] Milton Abramowitz and Irene A. Stegun, *Handbook of Mathematical Functions*. Dover Publications, New York, 1972.

[Ad+08] Muhammad Adeel, Hui Siu Cheung and Sikandar Hayat Khiyal, *Math GO! prototype of a content based mathematical formula search engine*. Journal of Theoretical and Applied Information Technology, Vol. 4, p. 1002–1012, 2008.

[Ad+12] Muhammad Adeel, Muhammad Sher and Malik S. H. Khiyal, *Efficient Cluster-Based Information Retrieval from Mathematical Markup Documents*. World Applied Sciences Journal Vol. 17(5), p. 611–616, 2012.

[AlYo07] Moody Ebrahem Altamimi and Abdou Youssef, *Wildcards in Math Search, Implementation Issues*. Proceedings of the ISCA 20th International Conference on Computer Applications in Industry and Engineering, p. 90–96, ISCA, 2007.

[Anc07] Ştefan Anca, *MaTeSearch A combined Math and Text search engine*. Bachelor's Thesis, Jacobs University Bremen, 2007.

[Anc09] Ştefan Anca, *Natural Language and Mathematics Processing for Applicable Theorem Search*. Master thesis, Jacobs University Bremen, 2009.

[Arch] arXiv.org e-print service at Cornell University, available at `http://www.archive.org` (seen March, 2012).

[Arx12] *Project arXMLiv*. Project home page at http://arxmliv.kwarc.info/ (seen March, 2012).

[As+04] Andrea Asperti, Ferruccio Guidi, Claudio S. Coen, Enrico Tassi and Stefano Zacchiroli, *A content based mathematical search engine: whelp*. In: Postproceedings of the Types 2004 International Conference, LNCS 3839, p. 17–32, Springer-Verlag, 2004.

[BaRi11] Ricardo A. Baeza-Yates, *Integrating Contents and Structure in Text Retrieval*. ISBN: 978-0-321-41691-9, Pearson Education Ltd., 2011.

[Bae96] Ricardo A. Baeza-Yates and Berthier A. Ribeiro-Neto, *Modern Information Retrieval - the concepts and technology behind search, Second edition*. ACM SIGMOD 25(1), p. 67–79, 1996.

[Bi+12] E. V. Birialtsev, M. R. Galimov, N. G. Zhiltsov and O. A. Nevzorova, *A novel approach to semantic search of mathematical expressions in the scientific documents*. OSTIS, p. 245–256, 2012.

[BrPa98] Sergey Brin and Lawrence Page , *The Anatomy of a Large-Scale Hypertextual Web Search Engine*. Computer Networks and ISDN Systems 30, p. 107-–117, 1998.

[Bu+92] Forbes J. Burkowski, *An Algebra for Hierarchically Organized Text-Dominate Databases*. Information Processing & Management 28(3), p. 333–348, 1992.

[Bu+10] Stefan Büttcher, Charles L. A. Clarke and Gordon V. Cormack, *Information Retrieval: Implementing and Evaluating Search Engines*. ISBN: 9780262026512, Mit Press, 2010.

[Cai04] Paul Cairns, *Informalising Formal Mathematics: Searching the Mizar Library with Latent Semantics*. LNCS 311, p. 58–72, Springer Berlin Heidelberg, 2004.

[CaCe04] Gerardo Canfora and Luigi Cerulo, *A Taxonomy of Information Retrieval Models and Tools*. Journal of Computing and Information Technology CIT 12, p. 175–194, 2004.

[Chá+01] Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates and José Luis Marroquín, *Searching in metric spaces*. ACM Comput. Surv. 33(3), p. 273–321, DOI: 10.1145/502807.502808, ACM, 2001.

[EiFa95] T. H. Einwohner and Richard J. Fateman, *Searching techniques for integral tables*. Proceedings of the 1995 international symposium on Symbolic and algebraic computation, p. 133–139, DOI: 10.1145/220346.220364, ACM, 1995.

[Ga+04] Harald Ganzinger, Robert Nieuwenhuis and Pilar Nivela, *Fast Term Indexing with Coded Context Trees*. Journal of Automated Reasoning 32(2), p. 103–120, DOI: 10.1023/B:JARS.0000029963.64213.ac, Kluwer Academic Publishers, 2004.

[Gi+11] Deyan Ginev, Heinrich Stamerjohanns, Bruce R. Miller and Michael Kohlhase, *The LaTeXML Daemon: Editable Math on the Collaborative Web*. Intelligent Computer Mathematics, p. 292–294, DOI: 10.1007/978-3-642-22673-1_25, Springer Berlin Heidelberg, 2011.

[Gra96] Peter Graf, *Term indexing*. LNAI, 1996.

[Gui03] Ferruccio Guidi, *Searching and Retrieving in Content-based Repositories of Formal Mathematical Knowledge*. PhD thesis, Università di Bologna, 2003.

[Gu+12] Mingjie Guan, Xuedong Tian, Fang Yang and Songqiang Yang, *A Mathematical Formula Retrieval Method Using Structure Sub-tree*. Advances in Intelligent Systems Research, p. 583–586, Atlantis Press, 2012.

[Hi+07] Yoshinori Hijikata, Hideki Hashimoto and Shogo Nishida, *An Investigation of Index Formats for the Search of MathML Objects*. Proceedings of the 2007 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology - Workshops, p. 244–248, IEEE Computer Society, 2007.

[HjSa03] Gisli R. Hjaltason and Hanan Samet, *Index-driven similarity search in metric spaces (Survey Article)*. ACM Trans. Database Syst. Vol. 28 (4). p. 517–580, DOI: 10.1145/958942.958948, ACM, 2003.

[Ima04] Robert Miner, *Enhancing the Searching of Mathematics*. A position paper based on the proceedings of the Enhancing the Searching of Mathematics Workshop, Available at `http://www.ima.umn.edu/complex/spring/math-searching.html` (seen March, 2012), 2004.

[Ka+12] Shahab Kamali, Johnson Apacible and Yasaman Hosseinkashi, *Answering Math Queries With Search Engines*. Proceedings of the 21st international conference companion on World Wide Web, p. 43–52, ACM, 2012.

[KaTo09] Shahab Kamali and Frank W. Tompa, *Improving Mathematics Retrieval*. Towards a Digital Mathematics Library, p. 37–48, Masaryk University Press, 2009.

[KaTo10] Shahab Kamali and Frank W. Tompa, *A New Mathematics Retrieval System*. Proceedings of the 19th ACM international conference on Information and knowledge management, p. 1413–1416, DOI: 10.1145/1871437.1871635, ACM, 2010.

[Ka+04] Howard Katz, Don Chamberlin, Denise Draper, Mary Fernandez, Michael Kay, Jonathan Robie, Michael Rys, Jerome Simeon, Jim Tivy and Philip Wadler, *XQuery from the Experts: A Guide to the W3C Xml Query Language*. ISBN: 9780321180605, ADDISON WESLEY Publishing Company Incorporated, 2004.

[Ki+12] Shinil Kim, Seon Yang and Youngjoong Ko, *Mathematical equation retrieval using plain words as a query*. Proceedings of the 21st ACM international conference on Information and knowledge management, p. 2407–2410, DOI: 10.1145/2396761.2398653, ACM, 2012.

[Ki+05] Sadaya Kishimoto, Takafumi Nakanishi, Tetsuya Sakurai and Takashi Kitigawa *An Implementation Method of Composite Association Retrieval System for Data of Mathematical Formula*. DBSJ Letters 4, 2005.

[KiHe93] Pekka Kilpeläinen and Heikki Mannila, *Retrieval from hierarchical texts by partial patterns*. Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval, p. 214–222, ACM, 1993.

[KoFr01] Micheal Kohlhase and Andreas Franke, *MBase: representing knowledge and context for the integration of mathematical software systems*. Journal of Symbolic Computation - Calculemus-99: integrating computation and deduction, p. 365–402, DOI: 10.1006/jsco.2000.0468, Academic Press, Inc., 2001.

[KoKo07] Andrea Kohlhase and Michael Kohlhase, *Reexamining the MKM Value Proposition: From Math Web Search to Math Web ReSearch*. Towards Mechanized Mathematical Assistants, p. 313–326, DOI: 10.1007/978-3-540-73086-6_25, Springer Berlin Heidelberg, 2007.

[KoŞu06] Michael Kohlhase and Ioan A. Şucan, *A search engine for mathematical formulae*. Proc. of Artificial Intelligence and Symbolic Computation, LNAI 4120, p. 241–253, Springer, 2006.

[Ko+12] Michael Kohlhase, Bogdan A. Matican and Corneliu C. Prodescu, *MathWebSearch 0.5: Scaling an Open Formula Search Engine*. LNCS 7362, p. 342–357, DOI: 10.1007/978-3-642-31374-5_23, Springer Berlin Heidelberg, 2012.

[KoIa+12] Michael Kohlhase and Mihnea Iancu, *Searching the Space of Mathematical Knowledge*. DML and MIR 2012, Masaryk University, 2012.

[Kur12] Dominik Kuropka, *Modelle zur Repräsentation natürlichsprachlicher Dokumente. Ontologie-basiertes Information-Filtering und -Retrieval mit relationalen Datenbanken*. Advances in Information Systems and Management Science, 10th issue, ISBN: 3-8325-0514-8, Logos Verlag, 2004.

[Kwa12] *Project KWARC*. Project home page at http://kwarc.info (seen March, 2012).

[Líš10] Martin Líška, *Searching in mathematical text*. Original title is "Vyhledávání v matematickém textu", Bachelor's Thesis, Masaryk University, 2010.

[Lí+11] Martin Líška, Petr Sojka, Michal Růžička and Peter Mravec, *Web Interface and Collection for Mathematical Retrieval: WebMIaS and MREC*. Towards a Digital Mathematics Library, 2011.

[LiMe06a] Paul Libbrecht and Erica Melis, *Semantic Search in LeActiveMath*. Proceedings of the WebALT 2006 Conference, 2006.

[LiMe06b] Paul Libbrecht, Erica Melis, *Methods to access and retrieve mathematical content in ActiveMath*. Proceedings of the Second international conference on Mathematical Software, p. 331–342, DOI: 10.1007/11832225_33 Springer-Verlag, 2006.

[Ma+10a] Kai Ma, Siu Cheung Hui and Kuiyu Chang, *Feature extraction and clustering-based retrieval for mathematical formulas*. Software Engineering and Data Mining (SEDM), p. 372–377, IEEE, 2010.

[Ma+10b] Michael MacCandless, Erik Hatcher and Otis Gospodnetić, *Lucene in action, Second Edition*. ISBN: 9781933988177, Manning Publications Company, 2010.

[Ma+02] K. Maly and M. Zubair and M. Nelson and X. Liu and H. Anan and J. Gao and J. Tang and Y. Zhao, *Archon - A Digital Library that Federates Physics*

Collections. Proceedings of the 2002 international conference on Dublin core and metadata applications: Metadata for e-communities: supporting diversity and convergence, p. 27–34, Dublin Core Metadata Initiative, 2002.

[Ma+08] Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, *Introduction to Information Retrieval*. Cambridge University Press. 2008.

[MiIg08] Yoshinori Miyazaki and Yoshihide Iguchi, *Development of Information-Retrieval Tool for MathML-based Math Expressions*. ICCE Proceedings, p. 419–426, 2008.

[Mil04] Robert Miner, *Math Searching and MathML in the NSDL*. Presentation available at `http://www.ima.umn.edu/talks/workshops/4-26-27.2004/miner/keynote/` (seen March, 2012), 2004.

[MiMu07] Robert Miner and Rajesh Munavalli, *An Approach to Mathematical Search Through Query Formulation and Data Normalization*. Towards Mechanized Mathematical Assistants, LNCS 4573, p. 342–355, DOI: 10.1007/978-3-540-73086-6_27, Springer Berlin Heidelberg, 2007.

[Mil08] Bruce R. Miller, *LaTeXML: A LATEX to XML Converter*. Available at `http://dlmf.nist.gov/LaTeXML/` (seen March, 2012).

[MiYo03] Bruce R. Miller and Abdou Youssef, *Technical Aspects of the Digital Library of Mathematical Functions*. Ann. Math. Artif. Intell. p. 121–136, Kluwer Academic Publishers, 2003.

[MiYo08] Bruce R. Miller and Abdou Youssef, *Augmenting Presentation MathML for Search*. Proceedings of the 9th AISC international conference, the 15th Calculemas symposium, and the 7th international MKM conference on Intelligent Computer Mathematics, p. 536–542, DOI: 10.1007/978-3-540-85110-3_43, Springer-Verlag, 2008.

[Miš05] Jozef Mišutka, *Math Search Engine*. `http://is.cuni.cz/studium/eng/dipl_st/index.php?doo=detail&did=44036` (seen March, 2012).

[Miš07] Jozef Mišutka, *Math Search Engine*. Diploma Thesis, Charles University, 2007.

[Miš08b] Jozef Mišutka and Leo Galamboš, *Extending Full Text Search Engine for Mathematical Content*. Towards Digital Mathematics Library, p. 55–67, Masaryk University, 2008.

[Miš08b] Jozef Mišutka and Leo Galamboš, *System Description: EgoMath2 As a Tool for Mathematical Searching on Wikipedia.org*. p. 307–309, DOI: 10.1007/978-3-642-22673-1_30, Springer Berlin Heidelberg, 2011.

[MiKl09] Jozef Mišutka and Jaroslav Klíma, *Mathematical User Interface*. p. 56–61, Proceedings of the 18th Annual Conference of Doctoral Students, MATFYZPRESS, 2009.

[MuKh11] A. Muhammad and M. S. H. Khiyal, *Search Systems for Math Information Retrieval: A Survey*. Journal of computing, volume 3, issue 2, p. 203–208, ISSN 2151-9617, 2011.

[Ng+12a] Tam T. Nguyen, Kuiyu Chang and Siu Cheung Hui, *A math-aware search engine for math question answering system*. Proceedings of the 21st ACM international conference on Information and knowledge management, p. 724–733, DOI: 10.1145/2396761.2396854, ACM, 2012.

[Ng+12b] Tam T. Nguyen, Siu Cheung Hui and Kuiyu Chang, *A lattice-based approach for mathematical search using Formal Concept Analysis*. Expert Syst. Appl., p. 5820–5828, DOI: 10.1016/j.eswa.2011.11.085, Pergamon Press, Inc., 2012.

[NoKo07] Immanuel Normann and Michael Kohlhase, *Extended Formula Normalization for ϵ-Retrieval and Sharing of Mathematical Knowledge*. Proceedings of the 14th symposium on Towards Mechanized Mathematical Assistants: 6th International Conference, p. 356–370, DOI: 10.1007/978-3-540-73086-6_28, Springer-Verlag, 2007.

[Nor08] Immanuel Normann, *Automated Theory Interpretation*. Jacobs University Bremen, 2008.

[Nsf02a] *Indexing, Searching & Retrieval of Mathematical Contents award by NSF*. Details available at `http://www.nsf.gov/awardsearch/showAward?AWD_ID=0208818` (seen March, 2012), 2002.

[Pe+89] Shmuel Peleg, Michael Werman and Hillel Rom, *A unified approach to the change of resolution: space and gray-level*. Pattern Analysis and Machine Intelligence Vol. 11 (7), p. 739–742, DOI: 10.1109/34.192468, IEEE, 1989.

[Pes07] Vladimir Pestov, *Intrinsic dimension of a dataset: what properties does one expect?*. International Joint Conference on Neural Networks IJCNN 2007, p. 2959–2964, DOI: 10.1109/IJCNN.2007.4371431, IEEE, 2007.

[Pes10] Vladimir Pestov, *Intrinsic Dimensionality*. SIGSPATIAL Special Vol. 2 (2), p. 8–11, DOI: 10.1145/1862413.1862416, 2010.

[RiVo00] Alexandre Riazanov and Andrei Voronkov, *Partially Adaptive Code Trees*. Logics in Artificial Intelligence 1919, p. 209–223, DOI: 10.1007/3-540-40006-0_15, Springer Berlin Heidelberg, 2000.

[RoTa60] David J. Rogers and Taffee T. Tanimoto, *Efficient Cluster-Based Information Retrieval from Mathematical Markup Documents*. Science Vol. 132(3434), p. 1115–1118, DOI: 10.1126/science.132.3434.1115, American Association for the Advancement of Science, 1960.

[Sa+11] Kunal Sain, Abhishek Dasgupta and Utpal Garain, *Efficient Cluster-Based Information Retrieval from Mathematical Markup Documents*. International Journal on Document Analysis and Recognition Volume 14, p. 75–85, DOI: 10.1007/s10032-010-0121-9, Springer-Verlag, 2011.

[Sa+75] Gerard Salton, Andrew Wong and Chung-Shu Yang, *A vector space model for automatic indexing*. Commun. ACM Vol. 18 (11), p. 613–620, DOI: 10.1145/361219.361220, ACM, 1975.

[SaHu09] Sidath Harshanath Samarasinghe and Siu Cheung Hui, *Mathematical Document Retrieval for Problem Solving*. In Proceedings of the 2009 International Conference on Computer Engineering and Technology, p. 583–587, DOI: 10.1109/ICCET.2009.69, IEEE Computer Society, 2009.

[Se+01] R. Sekar, I.V. Ramakrishnan and Andrei Voronkov, *Term indexing*. Handbook of automated reasoning, p. 1853–1964, Elsevier Science Publishers BV, 2001.

[SRG] *SIRET (SImilarity RETrieval) research group (SRG)*. Department of Software Engineering, Faculty of Mathematics and Physics, Charles University in Prague, available at `http://siret.ms.mff.cuni.cz/`.

[SkBe+11] Tomáš Skopal and Benjamin Bustos, *On nonmetric similarity search problems in complex domains*. ACM Comput. Surv. Vol. 43(4), p. 34:1–34:50, DOI: 10.1145/1978802.1978813, ACM, 2011.

[SoLí11] Petr Sojka and Martin Líška, *Indexing and Searching Mathematics in Digital Libraries Architecture, Design and Scalability Issues*. In proceedings of Calculemus/MKM 2011, LNAI 6824, p. 228–243, Springer-Verlag, 2011.

[St+10] Heinrich Stamerjohanns, Michael Kohlhase, Deyan Ginev, Catalin David and Bruce R. Miller, *Transforming Large Collections of Scientific Publications to XML*. Mathematics in Computer Science, p. 299–307, 2010.

[StKo08] Heinrich Stamerjohanns and Michael Kohlhase, *Transforming the arχiv to XML*. In Proceedings of the 9th AISC international conference, the 15th Calculemas symposium, and the 7th international MKM conference on Intelligent Computer Mathematics, p. 574–582, DOI: 10.1007/978-3-540-85110-3_46, Springer, 2008.

[Sti89] Mark E. Stickel, *The Path Indexing Method for Indexing Terms*. Tech. Rep. 473, Artificial Intelligence Center, SRI International, 1989.

[Th+05] Pang-Ning Tan, Michael Steinbach and Vipin Kumar, *Introduction to Data Mining*. Elsevier, 2005.

[Th+06] Frank Theiß, Volker Sorge, and Martin Pollet, *Interfacing to computer algebra via term indexing*. In Proceedings of Calculemus, p. 1–15, Elsevier, 2006.

[Tro05] Michael Trott, *Mathematical Searching of The Wolfram Functions Site*. The Mathematica Journal, Vol. 9(4), p. 713–726, 2005.

[TrWe12] Michael Trott and Eric Weisstein, *Mathematical Search*. Presentation from Wolfram—Alpha LLC available at `http://www.cicm-conference.org/2012/slides/StephenWolfram.mov` (seen March, 2012), 2012.

[Urb06] Josef Urban, *MoMM - Fast Interreduction and Retrieval in Large Libraries of Formalized Mathematics*. International Journal on Artificial Intelligence Tools, p. 109–130, 2006.

[XuWu08] Rui Xu and Donald C. Wunsch, *Clustering*. IEEE, 2008.

[You04] Abdou Youssef, *Advanced math search: issues & techniques*. Available at `http://www.ima.umn.edu/talks/workshops/4-26-27.2004/youssef/youssef.html` (seen March, 2012), 2004.

[You05] Abdou Youssef, *Information search and retrieval of mathematical contents: Issues and methods*. Proceedings of the ISCA 14th International Conference on Intelligent and Adaptive Systems and Software Engineering, p. 100–105, ISCA, 2005.

[You06] Abdou Youssef, *Roles of math search in mathematics*. Proceedings of the 5th international conference on Mathematical Knowledge Management, ISBN: 3-540-37104-4, 978-3-540-37104-5, p. 2–16, DOI: 10.1007/11812289_2, Springer-Verlag, 2006.

[You07] Abdou Youssef, *Methods of Relevance Ranking and Hit-content Generation in Math Search*. Proceedings of the 14th symposium on Towards Mechanized Mathematical Assistants: 6th International Conference, p. 393–406, DOI: 10.1007/978-3-540-73086-6_31, Springer-Verlag, 2007.

[YoAi09] Keisuke Yokoi and Akiko Aizawa, *An approach to similarity search for mathematical expressions using MathML*. Towards digital mathematics library, p. 27–35, Masaryk University, 2009.

[YoSh06] Abdou Youssef and Mohammed Shatnawi, *Math Search with Equivalence Detection Using Parse-tree Normalization*. The 4th ICCSIT, 2006.

[Yu10] Li Yu, *Image-based math retrieval using handwritten queries*. Master thesis, Rochester Institute of Technology, 2010.

[ZaYu11a] Richard Zanibbi and Bo Yuan, *Keyword and image-based retrieval for mathematical expressions*. Proc. Document Recognition and Retrieval XVIII, p. OI1–OI9, 2011.

[ZaBl12] Richard Zanibbi and Dorothea Blostein, *Recognition and retrieval of mathematical expressions*. International Journal on Document Analysis and Recognition, p. 331–357, DOI: 10.1007/s10032-011-0174-4, Springer-Verlag, 2012.

[Ze+06] Pavel Zezula, Giuseppe Amato, Vlastislav Dohnal and Michal Batko, *Similarity Search: The Metric Space Approach*. ISBN: 0-387-29146-6, Springer, 2006.

[Zez12] Pavel Zezula, *Future Trends in Similarity Searching*. LNCS 7404, p. 8–24, DOI: 10.1007/978-3-642-32153-5_2, Springer Berlin Heidelberg, 2012.

[Zh+08] Jin Zhao, Min-Yen Kan and Yin Leng Theng, *Math information retrieval: user requirements and prototype implementation*. Proceedings of the 8th ACM/IEEE-CS joint conference on Digital libraries, p. 187–196, DOI: 10.1145/1378889.1378921, ACM, 2008.

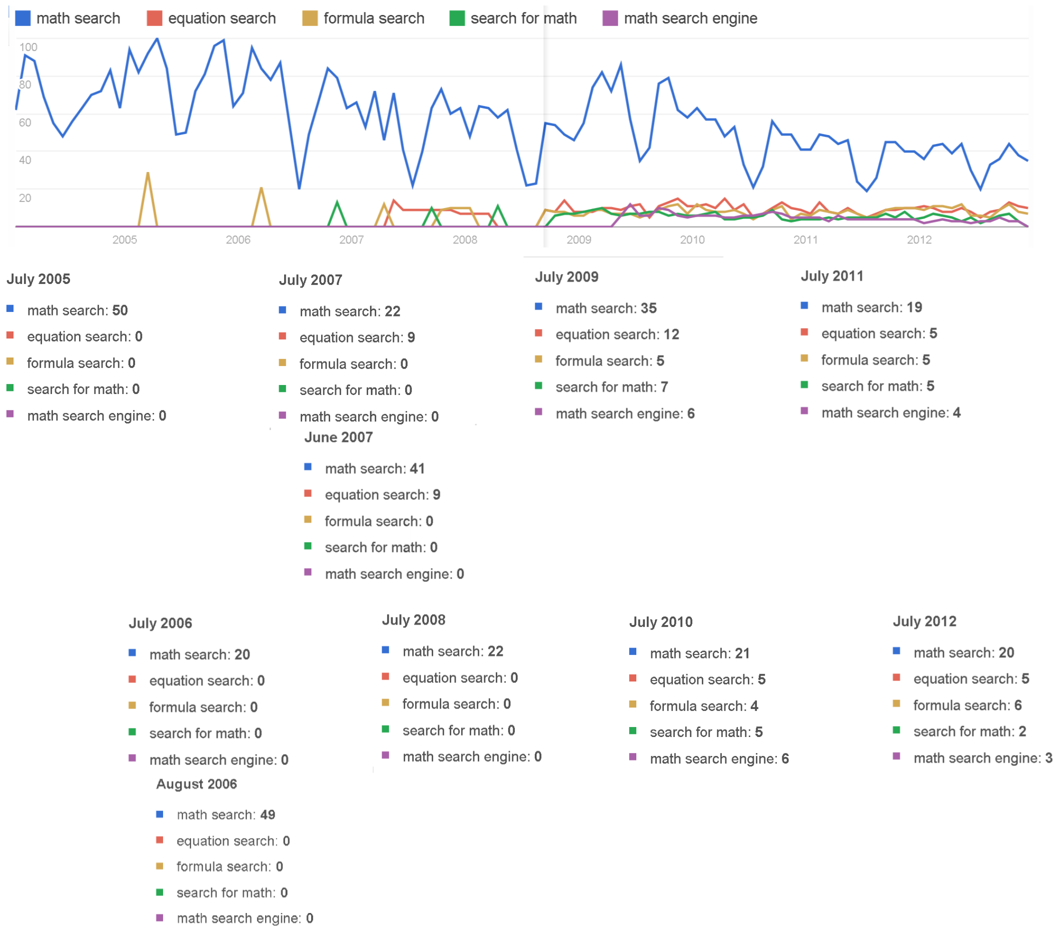
[Zh12] Nikita Zhiltsov, *MocaSSIN: A Mathematical Semantic Search EngINe*. available at `http://cll.niimm.ksu.ru/repository/default/content/lab/posters/Zhiltsov_MocaSSIN_Mathematical_Semantic_Search_EngINe.pdf` (seen March, 2012), 2012.

[Zo+98] Justin Zobel, Alistair Moffat and Kotagiri Ramamohanarao, *Inverted files versus signature files for text indexing*. ACM Trans. Database Syst., DOI: 10.1145/296854.277632, p. 453–490, ACM, 1998.

Appendices

APPENDIX A

Trends in Mathematical Searching Research Field



July 2005
- math search: **50**
- equation search: **0**
- formula search: **0**
- search for math: **0**
- math search engine: **0**

July 2007
- math search: **22**
- equation search: **9**
- formula search: **0**
- search for math: **0**
- math search engine: **0**

June 2007
- math search: **41**
- equation search: **9**
- formula search: **0**
- search for math: **0**
- math search engine: **0**

July 2009
- math search: **35**
- equation search: **12**
- formula search: **5**
- search for math: **7**
- math search engine: **6**

July 2011
- math search: **19**
- equation search: **5**
- formula search: **5**
- search for math: **5**
- math search engine: **4**

July 2006
- math search: **20**
- equation search: **0**
- formula search: **0**
- search for math: **0**
- math search engine: **0**

August 2006
- math search: **49**
- equation search: **0**
- formula search: **0**
- search for math: **0**
- math search engine: **0**

July 2008
- math search: **22**
- equation search: **0**
- formula search: **0**
- search for math: **0**
- math search engine: **0**

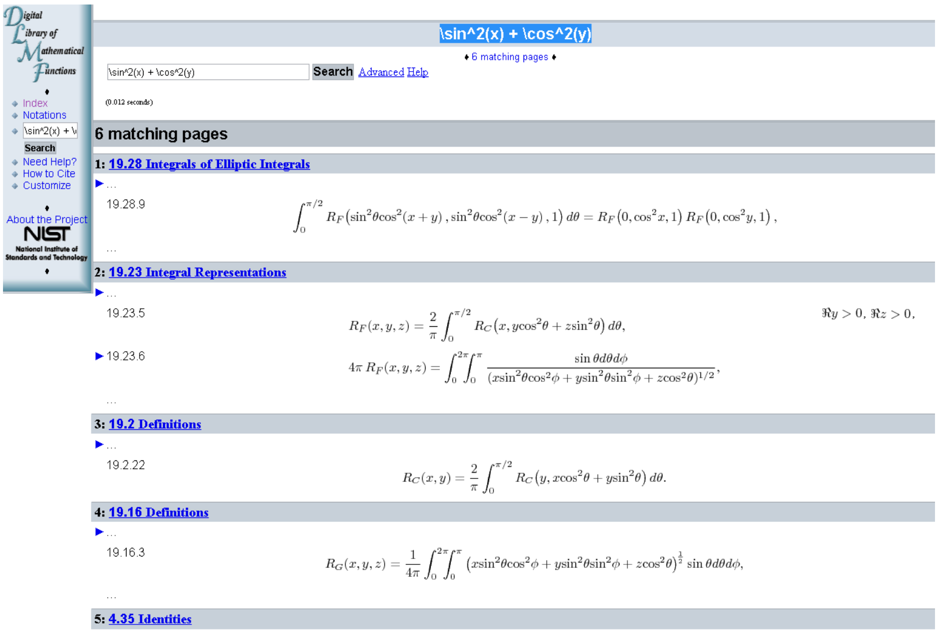
July 2010
- math search: **21**
- equation search: **5**
- formula search: **4**
- search for math: **5**
- math search engine: **6**

July 2012
- math search: **20**
- equation search: **5**
- formula search: **6**
- search for math: **2**
- math search engine: **3**

Figure A.1: Compiled on the 1st of Dec. 2012 from `http://www.google.com/trends/explore` with these parameters: category=Science (0-174), time=2004-present, location=Worldwide, search terms=web search.

APPENDIX B

Mathematical Search Engines Available Online



\sin^2(x) + \cos^2(y)

♦ 6 matching pages ♦

\sin^2(x) + \cos^2(y)

(0.012 seconds)

6 matching pages

1: 19.28 Integrals of Elliptic Integrals

▶ …

19.28.9 $$\int_0^{\pi/2} R_F\left(\sin^2\theta\cos^2(x+y), \sin^2\theta\cos^2(x-y), 1\right) d\theta = R_F\left(0, \cos^2 x, 1\right) R_F\left(0, \cos^2 y, 1\right),$$

…

2: 19.23 Integral Representations

▶ …

19.23.5 $$R_F(x,y,z) = \frac{2}{\pi}\int_0^{\pi/2} R_C\left(x, y\cos^2\theta + z\sin^2\theta\right) d\theta, \qquad \Re y > 0,\ \Re z > 0,$$

▶ 19.23.6 $$4\pi\, R_F(x,y,z) = \int_0^{2\pi}\int_0^{\pi} \frac{\sin\theta d\theta d\phi}{(x\sin^2\theta\cos^2\phi + y\sin^2\theta\sin^2\phi + z\cos^2\theta)^{1/2}},$$

…

3: 19.2 Definitions

▶ …

19.2.22 $$R_C(x,y) = \frac{2}{\pi}\int_0^{\pi/2} R_C\left(y, x\cos^2\theta + y\sin^2\theta\right) d\theta.$$

4: 19.16 Definitions

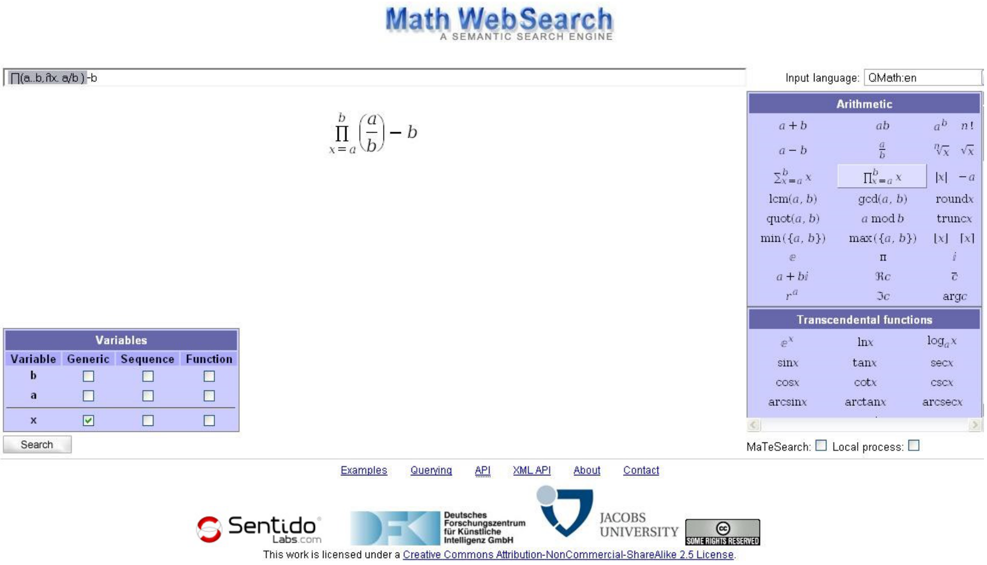
▶ …

19.16.3 $$R_G(x,y,z) = \frac{1}{4\pi}\int_0^{2\pi}\int_0^{\pi} \left(x\sin^2\theta\cos^2\phi + y\sin^2\theta\sin^2\phi + z\cos^2\theta\right)^{\frac{1}{2}} \sin\theta d\theta d\phi,$$

…

5: 4.35 Identities

Figure B.1: DLMF result page after searching for $\sin^2(x) + \cos^2(y)$



∏(a..b,λx. a/b) -b

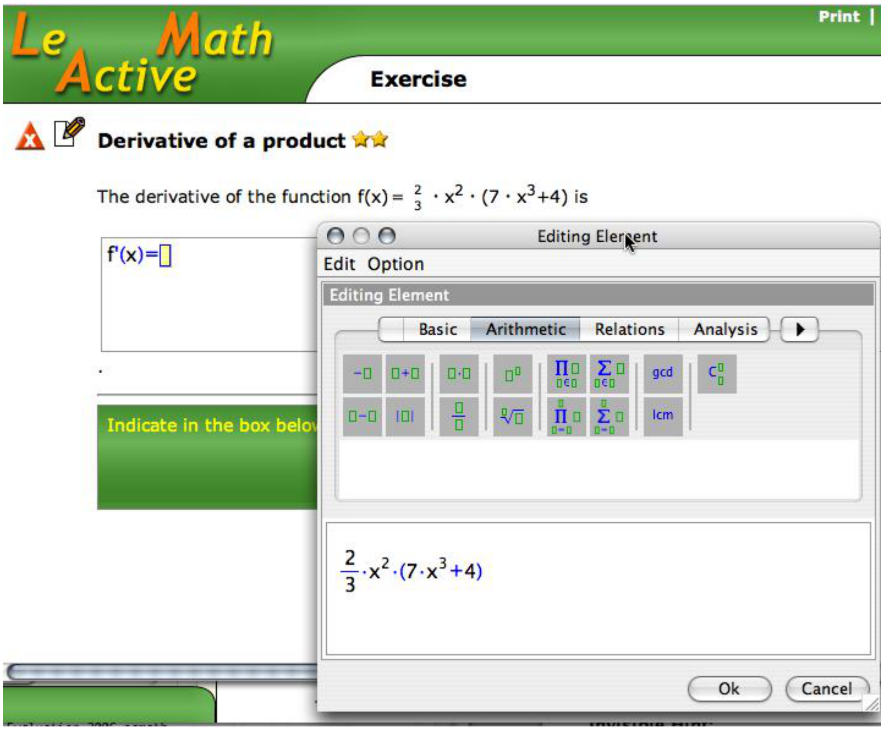
Input language: QMath:en

$$\prod_{x=a}^{b}\left(\frac{a}{b}\right)-b$$

Variable	Generic	Sequence	Function
b	□	□	□
a	□	□	□
x	☑	□	□

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 2.5 License.

Figure B.2: Math Web Search input of $\Pi_{x=a}^{b}(\frac{a}{b})-b$



Derivative of a product

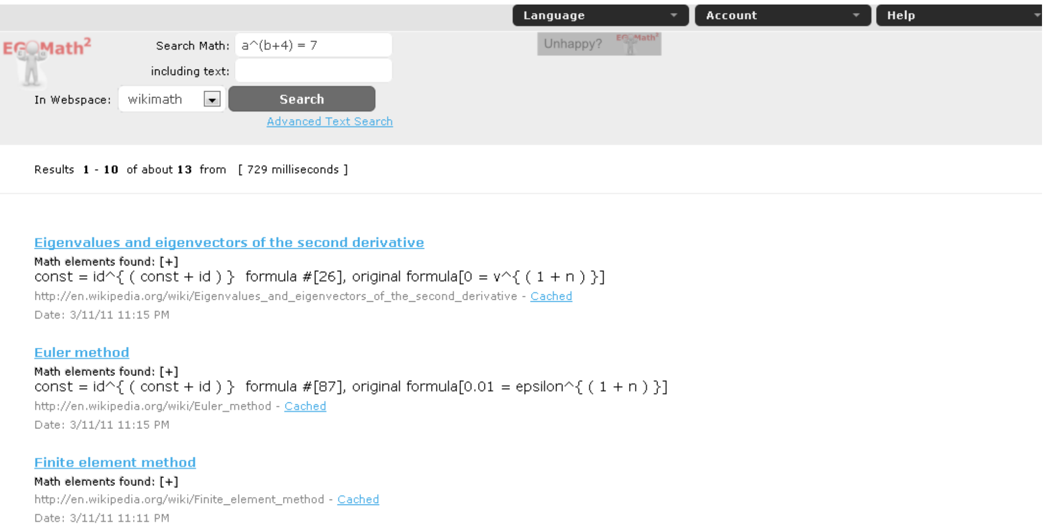
The derivative of the function f(x) = $\frac{2}{3}\cdot x^2\cdot(7\cdot x^3+4)$ is

f'(x)=□

Indicate in the box belo

$\frac{2}{3}\cdot x^2\cdot(7\cdot x^3+4)$

Figure B.3: Active Math input of $\frac{2}{3}*x^2*(7*x^3+4)$



Search Math: a^(b+4) = 7

including text:

In Webspace: wikimath

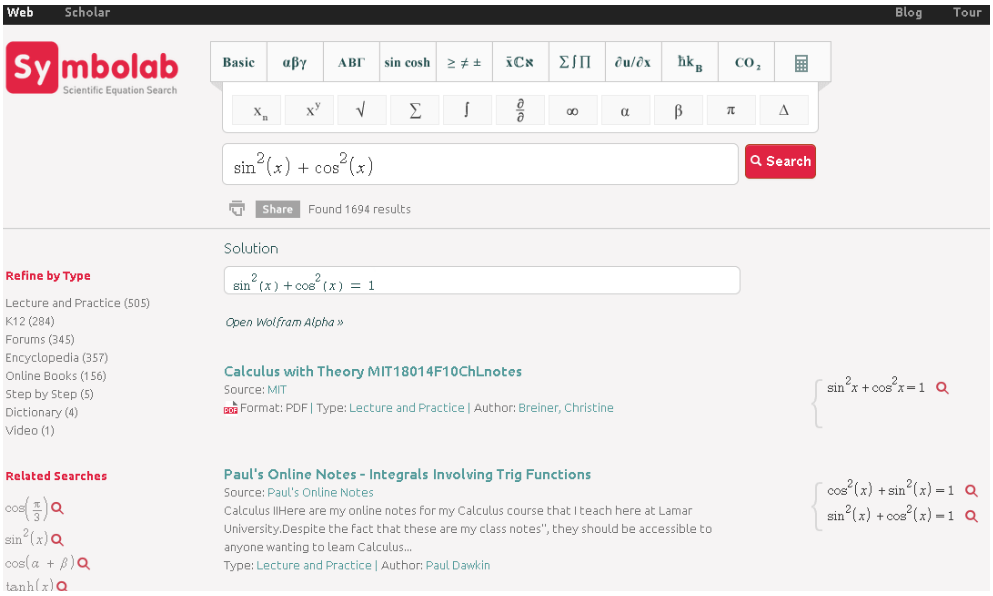
Results **1** - **10** of about **13** from [729 milliseconds]

Eigenvalues and eigenvectors of the second derivative
Math elements found: [+]
const = id^{ (const + id) } formula #[26], original formula[0 = v^{ (1 + n) }]
http://en.wikipedia.org/wiki/Eigenvalues_and_eigenvectors_of_the_second_derivative - Cached
Date: 3/11/11 11:15 PM

Euler method
Math elements found: [+]
const = id^{ (const + id) } formula #[87], original formula[0.01 = epsilon^{ (1 + n) }]
http://en.wikipedia.org/wiki/Euler_method - Cached
Date: 3/11/11 11:15 PM

Finite element method
Math elements found: [+]
http://en.wikipedia.org/wiki/Finite_element_method - Cached
Date: 3/11/11 11:11 PM

Figure B.4: EgoMath v1 (later release) result page after searching for $a^{b+4} = 7$



$\sin^2(x) + \cos^2(x)$

Found 1694 results

Solution

$\sin^2(x) + \cos^2(x) = 1$

Open Wolfram Alpha »

Refine by Type
Lecture and Practice (505)
K12 (284)
Forums (345)
Encyclopedia (357)
Online Books (156)
Step by Step (5)
Dictionary (4)
Video (1)

Related Searches
$\cos\left(\frac{\pi}{3}\right)$
$\sin^2(x)$
$\cos(\alpha + \beta)$
$\tanh(x)$

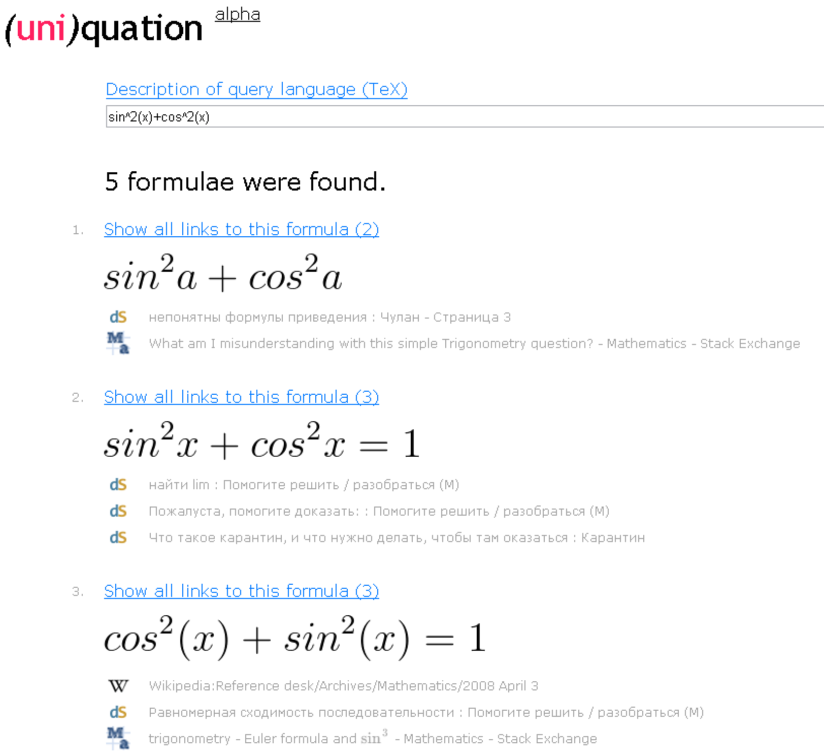
Calculus with Theory MIT18014F10ChLnotes
Source: MIT
Format: PDF | Type: Lecture and Practice | Author: Breiner, Christine

$\sin^2 x + \cos^2 x = 1$

Paul's Online Notes - Integrals Involving Trig Functions
Source: Paul's Online Notes
Calculus IIHere are my online notes for my Calculus course that I teach here at Lamar University.Despite the fact that these are my class notes", they should be accessible to anyone wanting to learn Calculus...
Type: Lecture and Practice | Author: Paul Dawkin

$\cos^2(x) + \sin^2(x) = 1$
$\sin^2(x) + \cos^2(x) = 1$

Figure B.5: Symbolab result page after searching for $\sin^2(x) + \cos^2(y)$



sin^2(x)+cos^2(x)

5 formulae were found.

1. Show all links to this formula (2)

$$sin^2 a + cos^2 a$$

dS непонятны формулы приведения : Чулан - Страница 3

What am I misunderstanding with this simple Trigonometry question? - Mathematics - Stack Exchange

2. Show all links to this formula (3)

$$sin^2 x + cos^2 x = 1$$

dS найти lim : Помогите решить / разобраться (М)

dS Пожалуста, помогите доказать: : Помогите решить / разобраться (М)

dS Что такое карантин, и что нужно делать, чтобы там оказаться : Карантин

3. Show all links to this formula (3)

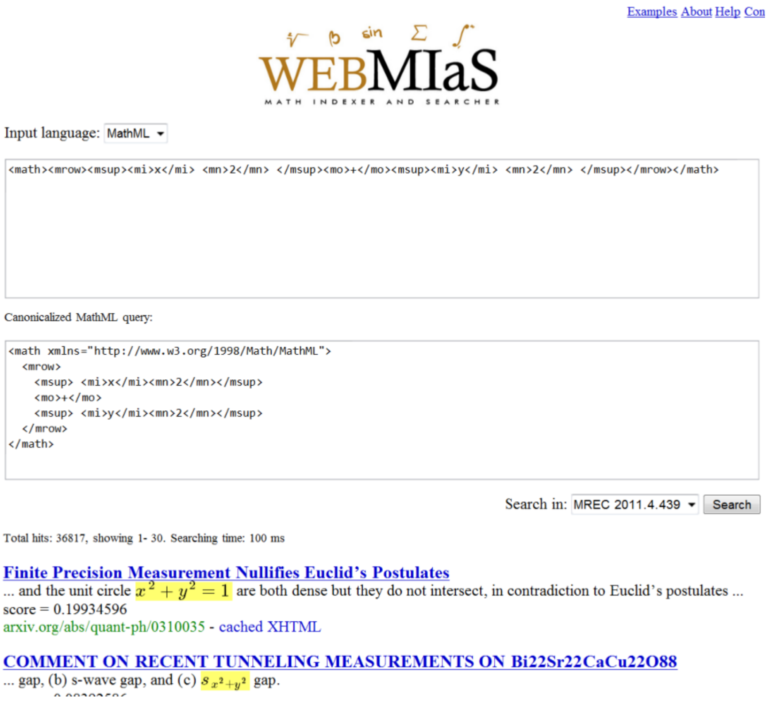
$$cos^2(x) + sin^2(x) = 1$$

Wikipedia:Reference desk/Archives/Mathematics/2008 April 3

dS Равномерная сходимость последовательности : Помогите решить / разобраться (М)

trigonometry - Euler formula and $\sin^3$ - Mathematics - Stack Exchange

Figure B.6: (uni)quation result page after searching for $\sin^2(x) + \cos^2(y)$



Input language: MathML

```
<math><mrow><msup><mi>x</mi> <mn>2</mn> </msup><mo>+</mo><msup><mi>y</mi> <mn>2</mn> </msup></mrow></math>
```

Canonicalized MathML query:

```
<math xmlns="http://www.w3.org/1998/Math/MathML">
  <mrow>
    <msup> <mi>x</mi><mn>2</mn></msup>
    <mo>+</mo>
    <msup> <mi>y</mi><mn>2</mn></msup>
  </mrow>
</math>
```

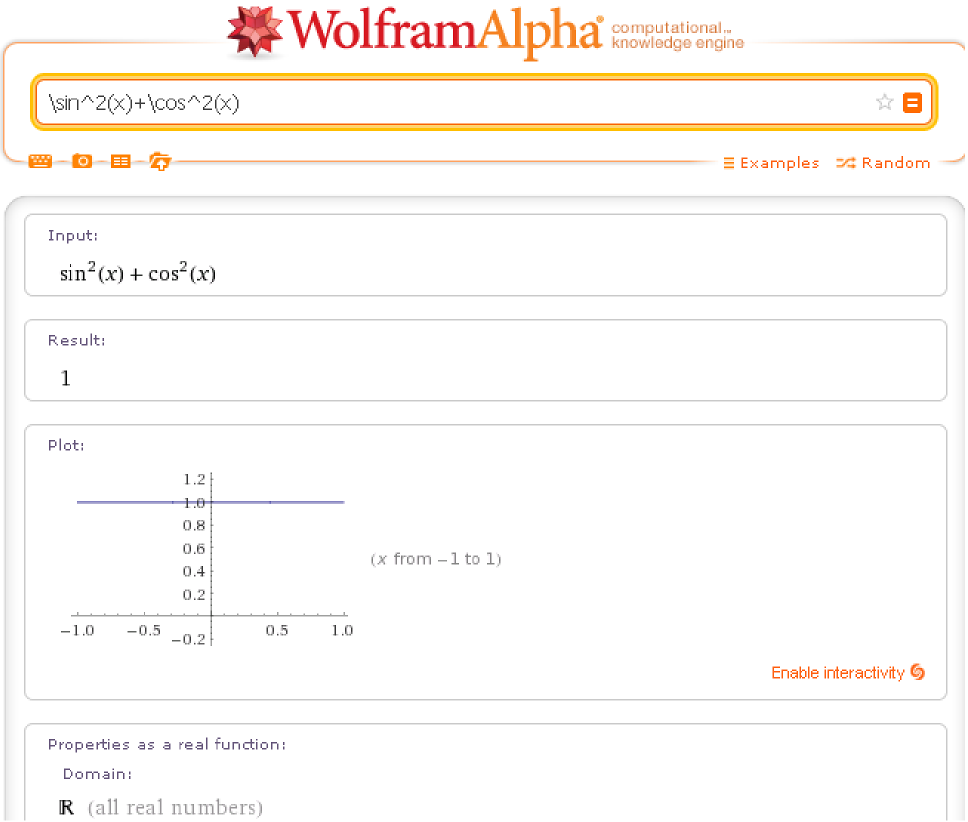
Search in: MREC 2011.4.439 [Search]

Total hits: 36817, showing 1- 30. Searching time: 100 ms

Finite Precision Measurement Nullifies Euclid's Postulates
... and the unit circle $x^2 + y^2 = 1$ are both dense but they do not intersect, in contradiction to Euclid's postulates ...
score = 0.19934596
arxiv.org/abs/quant-ph/0310035 - cached XHTML

COMMENT ON RECENT TUNNELING MEASUREMENTS ON Bi22Sr22CaCu22O88
... gap, (b) s-wave gap, and (c) $s_{x^2+y^2}$ gap.

Figure B.7: WEBMIaS result page after searching for $x^2 + y^2$



\sin^2(x)+\cos^2(x)

Input:

$\sin^2(x) + \cos^2(x)$

Result:

1

Plot:

(x from −1 to 1)

Properties as a real function:

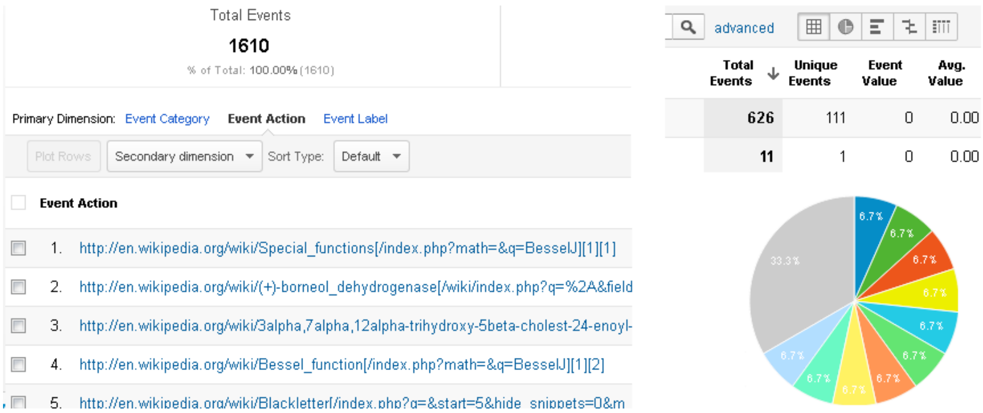
Domain:

$\mathbb{R}$ (all real numbers)

Figure B.8: WolframAlpha result page after searching for $\sin^2(x) + \cos^2(y)$

APPENDIX C

Click-through Implementation



Total Events

1610

% of Total: 100.00% (1610)

Primary Dimension: Event Category **Event Action** Event Label

Plot Rows | Secondary dimension | Sort Type: Default

advanced

Event Action
1. http://en.wikipedia.org/wiki/Special_functions[/index.php?math=&q=BesselJ][1][1]
2. http://en.wikipedia.org/wiki/(+)-borneol_dehydrogenase[/wiki/index.php?q=%2A&field
3. http://en.wikipedia.org/wiki/3alpha,7alpha,12alpha-trihydroxy-5beta-cholest-24-enoyl-
4. http://en.wikipedia.org/wiki/Bessel_function[/index.php?math=&q=BesselJ][1][2]
5. http://en.wikipedia.org/wiki/Blackletter[/index.php?q=&start=5&hide_snippets=0&m

Total Events ↓	Unique Events	Event Value	Avg. Value
626	111	0	0.00
11	1	0	0.00

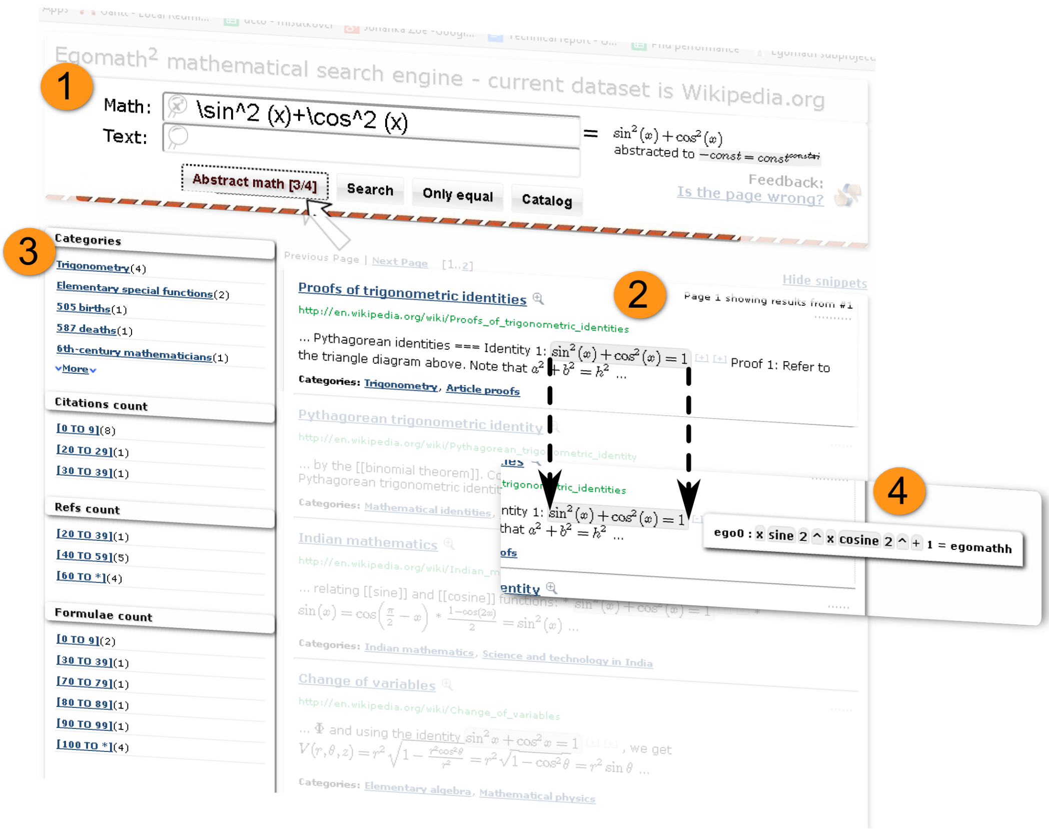
Figure C.1: Outbound links information visualisation from Google Analytics

Listing C.1: Click-through implementation in JavaScript

```
function track_clicks( link, url, referer, page_pos, position ){
 try {
  var pageTracker=_gat._getTracker( 'UA-XXXXXX-X' );
  pageTracker._trackEvent( 'Outbound Links', url+"["+referer+"]["+
      page_pos+"]["+position+"]" );
  setTimeout('document.location = "' + link.href + '"', 100);
 }catch(err){}
}
```

APPENDIX D

EgoMath v3



Egomath² mathematical search engine - current dataset is Wikipedia.org

Math: \sin^2 (x)+\cos^2 (x) = $\sin^2(x) + \cos^2(x)$
abstracted to $-const = const^{const}$

Text:

Previous Page | Next Page [1..2]

Page 1 showing results from #1

Categories
- Trigonometry(4)
- Elementary special functions(2)
- 505 births(1)
- 587 deaths(1)
- 6th-century mathematicians(1)
- More

Citations count
- [0 TO 9](8)
- [20 TO 29](1)
- [30 TO 39](1)

Refs count
- [20 TO 39](1)
- [40 TO 59](5)
- [60 TO *](4)

Formulae count
- [0 TO 9](2)
- [30 TO 39](1)
- [70 TO 79](1)
- [80 TO 89](1)
- [90 TO 99](1)
- [100 TO *](4)

Proofs of trigonometric identities
http://en.wikipedia.org/wiki/Proofs_of_trigonometric_identities
... Pythagorean identities === Identity 1: $\sin^2(x) + \cos^2(x) = 1$ [+] [+] Proof 1: Refer to the triangle diagram above. Note that $a^2 + b^2 = h^2$ ...
Categories: Trigonometry, Article proofs

Pythagorean trigonometric identity
http://en.wikipedia.org/wiki/Pythagorean_trigonometric_identity
... by the [[binomial theorem]]. C... Pythagorean trigonometric identit...
Categories: Mathematical identities, ...

Indian mathematics
http://en.wikipedia.org/wiki/Indian_m...
... relating [[sine]] and [[cosine]] functions: * $\sin^2(x) + \cos^2(x) = 1$ * $\sin(x) = \cos\left(\frac{\pi}{2} - x\right) * \frac{1-\cos(2x)}{2} = \sin^2(x)$ ...
Categories: Indian mathematics, Science and technology in India

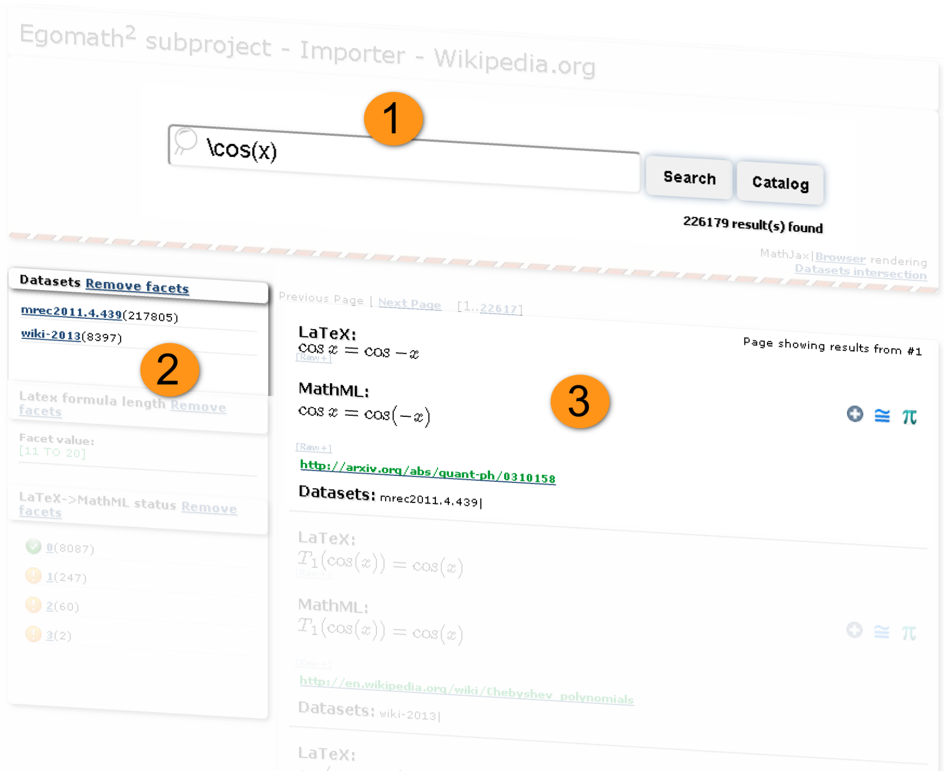
Change of variables
http://en.wikipedia.org/wiki/Change_of_variables
... Φ and using the identity $\sin^2 x + \cos^2 x = 1$ [+] [+], we get $V(r, \theta, z) = r^2 \sqrt{1 - \frac{r^2 \cos^2\theta}{r^2}} = r^2 \sqrt{1 - \cos^2\theta} = r^2 \sin\theta$ ...
Categories: Elementary algebra, Mathematical physics

ego0 : x sine 2 ^ x cosine 2 ^ + 1 = egomathh

Figure D.1: EgoMath v3 graphical user interface screenshot after preforming mathematical search for $\sin^2(x) + \cos^2(x)$. The input fields used to enter mathematics and textual queries including the formula preview are marked by number 1. Number 2 shows one returned result including highlighted formula and categories it belongs to. Number 3 shows the various categorisation of the relevant documents for this particular query and number 4 shows the precise highlighted formula in postfix representation produced by EgoMath.

APPENDIX E

Importer Formula Search Engine



Egomath^2 subproject - Importer - Wikipedia.org

\cos(x)

226179 result(s) found

MathJax|Browser rendering
Datasets intersection
Datasets Remove facets

mrec2011.4.439(217805)
wiki-2013(8397)

2
Latex formula length Remove facets

Facet value:
[11 TO 20]

LaTeX->MathML status Remove facets

0(8087)
1(247)
2(60)
3(2)

Previous Page | Next Page [1..22617]

Page showing results from #1

LaTeX:
$\cos x = \cos -x$

MathML:
$\cos x = \cos(-x)$

http://arxiv.org/abs/quant-ph/0310158

Datasets: mrec2011.4.439|

LaTeX:
$T_1(\cos(x)) = \cos(x)$

MathML:
$T_1(\cos(x)) = \cos(x)$

http://en.wikipedia.org/wiki/Chebyshev_polynomials

Datasets: wiki-2013|

LaTeX:

Figure E.1: Importer framework graphical user interface. The input field used to enter words are marked by number 1. Number 2 shows the different datasets the relevant results belong to. Number 3 shows one returned result including LaTeX and MathML visualisation and example documents where it was found.

APPENDIX F

Queries Used in EgoMath's Evaluation

Below is the list of 13 mathematical queries which are used in the index evaluation. Each query is executed with six different settings totalling to 78 different queries. The first line of each item in the list below is the graphical representation of the query, the second line is the actual query value and the third one are the query numbers for this particular query.

1. $e^{i\pi} = -1$
   math:([1]e^{i\pi}=-1)
   queries: 1-6
2. $\{!egonear\ df = "math"\ apart = 5\}"egosem\ 0\ k\ noteq", "egomathh"$
   {!egonear df="math" apart=5}"egosem 0 k noteq","egomathh"
   queries: 7-12
3. $\pi = c/d$
   math:([1]\pi=c/d)
   queries: 13-18
4. $e = \lim_{n\to\infty}(1 + 1/n)^n$
   math:([1]e=\lim_{n\to\infty}(1+1/n)^n)
   queries: 19-24
5. $\frac{d}{dx}e^x$
   math:([1]\frac{d}{dx}e^x)
   queries: 25-30
6. $\sum_{n=0}^{\infty} \frac{f^n(a)}{n!}(x - a)^n$

   math:([1]\sum_{n=0}^{\infty}\frac{f^{n}(a)}{n!}(x-a)^n)

   queries: 31-36

7. $Ax = \lambda x$

   math:([1]Ax=\lambda x)

   queries: 37-42

8. $z_{n+1} = z_n^2 + c$

   math:([1]z_{n+1} = z_n^2 + c)

   queries: 43-48

9. $|x + y| \leq |x| + |y|$

   math:([1]|x+y|\ leq |x|+ |y|)

   queries: 49-54

10. $E = mc^2$

    math:([1]E=m c^2)

    queries: 55-60

11. $F = G\frac{m_1 m_2}{egovar^2}$

    math:([1]F=G\frac{m_1 m_2}{egovar^2})

    queries: 61-66

12. $cos^2(x) + sin^2(x)$

    math:([1]cos^2(x) + sin^2(x))

    queries: 67-72

13. $id + id$

    math:([1]id + id)

    queries: 73-78

Each query is executed with several default parameters and four mutable settings. The default parameters are: search from the beginning and return the contents of the fields "id" and "title" in the response. The four mutable settings are:

- **use_payload** - if true, the query should use payloads specifying term boosts at particular positions;
- **use_facet** - if true, the facets for category, citations_count, refs_count and math_count should be returned;
- **use_hl** - if true, the highlighting is performed returning one highlight occurrence;
- **use_only_equal** - if true, no sub-formula search is performed.

Each query is executing with the settings in Table F.1:

	use_payload	**use_facet**	**use_hl**	**use_only_equal**
1.				
2.	•			
3.	•	•		
4.	•		•	
5.	•			•
6.	•		•	•

Table F.1: Different settings used for every query. Each row (except the header) represents one query if concatenated with the mathematical query listed above. Symbol • indicates the option is set to true.

APPENDIX G

EgoMath and FBA evaluation queries

G.1 Comparison of EgoMath and FBA

Query	EgoMath v3	FBA-3
1) $e^{i\pi} = -1$	$e^{i\pi} = -1$ $e^{i\pi} + 1 = 0$	$e^{i\pi} = -1$ $e^{i\pi} = -1 + 0i$ $e^{\xi} - 1 = u\xi$ $e^{i\pi} + 1 = 0$ $DAF = 1 + e^{-c\pi}$
2) $\pi = c/d$	$\pi = c/d$ $DV/Dt = 0$ $(\frac{p}{q}) = 1$ $\left(\frac{D}{N}\right) = 1$ $\frac{dt}{ds} = 1$	$\pi = c/d$ $PI = \frac{mass}{height^3}$ $\pi = \frac{355}{113}$ c/d $\pi = \frac{3927}{1250}$
3) $e = \lim\limits_{n\to\infty}(1+1/n)^n$	$e = \lim_{n\to\infty}(1+1/n)^n$	$e = \lim_{n\to\infty}(1+\frac{1}{n})^n$ $y = \lim_{n\to\infty}(1+\frac{x}{n})^n$ $e^x = \lim_{n\to\infty}(1+\frac{x}{n})^n$ $e^x \equiv \lim_{p\to\text{infin}}(1+1/p)^{px}$ $\exp(x) = \lim_{n\to\infty}(1+\frac{x}{n})^n$

4) $\frac{d}{dx}e^x$	$\frac{d}{dx}e^x = e^x$ $(*)e^0 = 1, \frac{d}{dx}e^x = e^x, e^x > 0, x \in \mathbb{R}$ $\frac{k}{d} \cdot 2^d$ $\frac{p}{q}e^t = pe^t + 1 - p$ $H_x = L\frac{\delta T}{\delta y}^{TM} + \frac{1}{j\omega\mu}\frac{\mathrm{dL}}{\mathrm{d}z}\frac{\delta T}{\delta x}^{TE} =$ $L\frac{\delta T}{\delta y}^{TM} - \frac{k_z}{\omega\mu}L\frac{\delta T}{\delta x}^{TE}(30)$	$e^x \sin y e^x \cos y$ $e^x \cos y$ $e^x \sin y$ $S(e^x)$ $E^f \gg E^c$
5) $\sum_{n=0}^{\infty} \frac{f^n(a)}{n!}(x-a)^n$	$\sum_{n=0}^{\text{infin}} \frac{f^n(a)}{n!}(x-a)^n$ $\sum_{n=0}^{\infty} \frac{f^n(a)}{n!}(x-a)^n$ $f(z) \approx \sum_{k=0}^{\infty} \frac{f^k(c)}{k!}(z-c)^k$ $T(x) = \sum_{n=0}^{\infty} \frac{f^n(x_0)}{n!}(x-x_0)^n$	$\sum_{n=0}^{\text{infin}} \frac{f^n(a)}{n!}(x-a)^n$ $\sum_{n=0}^{\infty} \frac{f^n(a)}{n!}(x-a)^n$ $\sum_{n=0}^{\text{infin}} \frac{f^n(0)}{n!}x^n$ $T(x) =$ $\sum_{n=0}^{\infty} \frac{f^n(x_0)}{n!}(x-x_0)^n$ $f(z) \approx \sum_{k=0}^{\infty} \frac{f^k(c)}{k!}(z-c)^k$
6) $Ax = \lambda x$	$Ax = \lambda x$ $B^{-1}Ax = \lambda x$ $Ax = \lambda x$ $F = kX$ $F{=}f(G)$	$Ax = \lambda x$ $[A][x] = [x]\lambda$ $y = \lambda x$ $\Lambda(x)$ $Aw = \lambda Bw$
7) $z_{n+1} = z_n^2 + c$	${z_{n+1}} = {z_n}^2 + c$	${z_{n+1}} = {z_n}^2 + c$ $z_{k+1} =$ $z_k + hf((z_{k+1} + z_k)/2)$ $z_{k+1} = z_k + hf(z_{k+1})$ $z_{k+1} = z_k + hf(z_k)$ $z_{k+1} = z_k + hf(q_k, p_{k+1})$
8) $\|x+y\| \leq \|x\| + \|y\|$	$\|x+y\| \leq \|x\| + \|y\|$ $\|a+b\| \leq \|a\| + \|b\|$ $\\|A+B\\| \leq \\|A\\| + \\|B\\|$ $\|z_1+z_2\| \leq \|z_1\| + \|z_2\|$	$\\|x+y\\| \leq \\|x\\| + \\|y\\|$ $\\|v+u\\| \leq \\|v\\| + \\|u\\|$ $\\|A+B\\| \leq \\|A\\| + \\|B\\|$ $\|a+b\| \leq \|a\| + \|b\|$ $\|z_1+z_2\| \leq \|z_1\| + \|z_2\|$
9) $E = mc^2$	$E = mc^2$ $E_0 = mc^2$ $E_0 = m_0c^2$ $E_0 = mc^2 = \hbar\omega_0$ $E_k = \frac{1}{2}mv^2; E = mc^2; E = pv; E =$ hc/λ	$E = mc^2$ $E_0 = mc^2$ $E_0 = m_0c^2$ $E_{rest} = mc^2$ $E_{\text{rest}} = E_0 = mc^2$

10) $cos^2(x) + sin^2(x)$	$\sin^2(x) + \cos^2(x) = 1$ $z = \sin^2(x) + \cos^2(x)$ $(\tan(x))' = \left(\frac{\sin(x)}{\cos(x)}\right)' = \frac{\cos^2(x)+\sin^2(x)}{\cos^2(x)} = \frac{1}{\cos^2(x)} = \sec^2(x)$ $\cos^2 x + \sin^2 x = 1$ and $\cos 2x = \cos^2 x - \sin^2 x$ $A = 2\pi \int_0^\pi \sin(t)\sqrt{(\cos(t))^2 + (\sin(t))^2}dt$	$\sin^2(x) + \cos^2(x) = 1$ $z = \sin^2 x + \cos^2 x$ $\cos(z) + \sin(z)$ $\cos^2 \gamma + \sin^2 \gamma = 1$ $\cos(t)^2 + \sin(t)^2 = 1$
11) $F = G\frac{m_1m_2}{egovar^2}$	$F = G\frac{m_1m_2}{r^2}$ $F = G\frac{m_1m_2}{r^2} = \left(G\frac{m_1}{r^2}\right)m_2$ $F = G \cdot \frac{m_1 \cdot m_2}{d^2}$ $\mathbf{F} = \frac{q_1q_2}{r^2}\hat{\mathbf{r}}$	$F = G\frac{m_1m_2}{r^2}$ $F = G\frac{m_1m_2}{r^2} = (G\frac{m_1}{r^2})m_2$ $F = G \cdot \frac{m_1 \cdot m_2}{d^2}$ $F = G\frac{Mm}{r^2}$ $8000m = \frac{120m}{15mil} \times 1000$

G.2 FBA

The results for all tested FBA algorithms are below listed side-by-side for better visualisation. FBA 1 is algorithm which uses keywords without multiplication, FBA 2 uses real values without multiplication and FBA 3 uses real values with multiplication.

FBA 1	FBA 2	FBA 3
1) $e^{i\pi} = -1$		
$e^{i\pi} = -1$ $e^{\xi} - 1 = u\xi$ $e^{i\pi} = -1 + 0i$ $\Delta = e^D - 1$ $r = e^i - 1$	$e^{i\pi} = -1$ $e^{i\pi} = -1 + 0i$ $r = e^i - 1$ $r(e^{i\alpha}) = -\cot(\alpha/2)$ $e^{\xi} - 1 = u\xi$	$e^{i\pi} = -1$ $e^{i\pi} = -1 + 0i$ $e^{\xi} - 1 = u\xi$ $e^{i\pi} + 1 = 0$ $DAF = 1 + e^{-c\pi}$
2) $\pi = c/d$		
$w/h = 3.3$ $\frac{a}{a} = 1$ $dQ/dt{=}0$ $div = 0$ $d/\lambda = 0.6$	$\pi = c/d$ $PI = \frac{mass}{height^3}$ $\pi = \frac{355}{113}$ c/d $\pi = \frac{3927}{1250}$	$\pi = c/d$ $PI = \frac{mass}{height^3}$ $\pi = \frac{355}{113}$ c/d $\pi = \frac{3927}{1250}$
3) $e = \lim_{n\to\infty}(1 + 1/n)^n$		

$e = \lim_{n\to\infty}(1+\frac{1}{n})^n$ $\exp(x) = \lim_{n\to\infty}(1+\frac{x}{n})^n$ $y = \lim_{n\to\infty}(1+\frac{x}{n})^n$ $e^x = \lim_{n\to\infty}(1+\frac{x}{n})^n$ $\lim_{N\to\infty}(1+\frac{r}{N})^{Nt} = e^{rt}$	$e = \lim_{n\to\infty}(1+\frac{1}{n})^n$ $\exp(x) = \lim_{n\to\infty}(1+\frac{x}{n})^n$ $y = \lim_{n\to\infty}(1+\frac{x}{n})^n$ $e^x = \lim_{n\to\infty}(1+\frac{x}{n})^n$ $e^x \equiv \lim_{p\to\text{infin}}(1+1/p)^{px}$	$e = \lim_{n\to\infty}(1+\frac{1}{n})^n$ $y = \lim_{n\to\infty}(1+\frac{x}{n})^n$ $e^x = \lim_{n\to\infty}(1+\frac{x}{n})^n$ $e^x \equiv \lim_{p\to\text{infin}}(1+1/p)^{px}$ $\exp(x) = \lim_{n\to\infty}(1+\frac{x}{n})^n$
4) $\frac{d}{dx}e^x$		
$\frac{1}{C}e^{\eta_1}$ $\frac{\lambda^k}{k!}\cdot e^{-\lambda}$ $\frac{e^{-\lambda}\lambda^j}{j!}$ $e^{-\lambda}\frac{\lambda^k}{k!}$ $\frac{e^{-c}c^k}{k!}$	$e^x \sin y e^x \cos y$ $e^x \cos y$ $e^{S(x)}$ $e^{\psi(x)}$ $e^{r_1 x}$	$e^x \sin y e^x \cos y$ $e^x \cos y$ $e^x \sin y$ $S(e^x)$ $E^f \gg E^c$
5) $\sum_{n=0}^{\infty}\frac{f^n(a)}{n!}(x-a)^n$		
$\sum_{n=0}^{\infty}\frac{f^n(a)}{n!}(x-a)^n$ $\sum_{n=0}^{\text{infin}}\frac{f^n(a)}{n!}(x-a)^n$ $\sum_{n=0}^{\infty}\frac{x^n}{n!}$ $\sum_{n=0}^{\infty}\frac{z^n}{n!}$ $\sum_{n=0}^{\infty}\frac{m_n t^n}{n!}$	$\sum_{n=0}^{\text{infin}}\frac{f^n(a)}{n!}(x-a)^n$ $\sum_{n=0}^{\infty}\frac{f^n(a)}{n!}(x-a)^n$ $\sum_{n=0}^{\text{infin}}\frac{f^n(0)}{n!}x^n$ $\sum_{n=0}^{\infty}\frac{(it)^n\lambda^n}{n!}\Gamma(1+n/k)$ $\sum_{n=0}^{\infty}\frac{m_n t^n}{n!}$	$\sum_{n=0}^{\text{infin}}\frac{f^n(a)}{n!}(x-a)^n$ $\sum_{n=0}^{\infty}\frac{f^n(a)}{n!}(x-a)^n$ $\sum_{n=0}^{\text{infin}}\frac{f^n(0)}{n!}x^n$ $T(x) = \sum_{n=0}^{\infty}\frac{f^n(x_0)}{n!}(x-x_0)^n$ $f(z) \approx \sum_{k=0}^{\infty}\frac{f^k(c)}{k!}(z-c)^k$
6) $Ax = \lambda x$		
$M = \angle zcy$ $V = V(r)$ $C = Ab(X)$ $f(m) = n$ $t = O(\epsilon)$	$Ax = \lambda x$ $Ax = \lambda Bx$ $[A][x] = [x]\lambda$ $\dot{x} = Ax$ $x = a\lambda$	$Ax = \lambda x$ $[A][x] = [x]\lambda$ $y = \lambda x$ $\Lambda(x)$ $Aw = \lambda Bw$
7) $z_{n+1} = z_n^2 + c$		
$z_{n+1} = {z_n}^2 + c$ $y_{k+1} = y_k + m$ $x_{i+1} = x_i + h$ $t_{k+1} = t_k + h$ $t_{n+1} = t_n + h$	$z_{n+1} = {z_n}^2 + c$ $z_{k+1} = z_k + hf(z_k)$ $z_{k+1} = z_k + hf(z_{k+1})$ $z_{k+1} = z_k + hf((z_{k+1}+z_k)/2)$ $z_{k+1} = z_k + hf(q_k, p_{k+1})$	$z_{n+1} = {z_n}^2 + c$ $z_{k+1} = z_k + hf((z_{k+1}+z_k)/2)$ $z_{k+1} = z_k + hf(z_{k+1})$ $z_{k+1} = z_k + hf(z_k)$ $z_{k+1} = z_k + hf(q_k, p_{k+1})$
8) $\lvert x+y\rvert \leq \lvert x\rvert + \lvert y\rvert$		

$\|x+y\| \leq \|x\|+\|y\|$ $\\|v+u\\| \leq \\|v\\|+\\|u\\|$ $\\|x+y\\| \leq \\|x\\|+\\|y\\|$ $\|a+b\| \leq \|a\|+\|b\|$ $\\|A+B\\| \leq \\|A\\|+\\|B\\|$	$\\|x+y\\| \leq \\|x\\|+\\|y\\|$ $\\|v+u\\| \leq \\|v\\|+\\|u\\|$ $\\|A+B\\| \leq \\|A\\|+\\|B\\|$ $\|a+b\| \leq \|a\|+\|b\|$ $\|z_1+z_2\| \leq \|z_1\|+\|z_2\|$	$\\|x+y\\| \leq \\|x\\|+\\|y\\|$ $\\|v+u\\| \leq \\|v\\|+\\|u\\|$ $\\|A+B\\| \leq \\|A\\|+\\|B\\|$ $\|a+b\| \leq \|a\|+\|b\|$ $\|z_1+z_2\| \leq \|z_1\|+\|z_2\|$
9) $E = mc^2$		
$\nabla^4\psi = 0$ $l(x^2) = 2$ $V = 0.26 \times D^3$ $\nabla^2 p = 0$ $E = \pm mc^2$	$E = mc^2$ $E_0 = mc^2$ $E_0 = m_0c^2$ $E = \gamma mc^2$ $E_{rest} = mc^2$	$E = mc^2$ $E_0 = mc^2$ $E_0 = m_0c^2$ $E_{rest} = mc^2$ $E_{\text{rest}} = E_0 = mc^2$
10) $cos^2(x) + sin^2(x)$		
$\cos(z) + \sin(z)$ $\sin^2(X) + \cos^2(X) = 1$ $\cos^2\theta + \sin^2\theta = 1$ $\sin^2(x) + \cos^2(x) = 1$ $z = \sin^2 x + \cos^2 x$	$\sin^2(x) + \cos^2(x) = 1$ $z = \sin^2 x + \cos^2 x$ $\cos x + i \sin x$ $\cos^2 x + \sin^2 x =$ 1 and $\cos 2x = \cos^2 x - \sin^2 x$ $\cos(z) + \sin(z)$	$\sin^2(x) + \cos^2(x) = 1$ $z = \sin^2 x + \cos^2 x$ $\cos(z) + \sin(z)$ $\cos^2\gamma + \sin^2\gamma = 1$ $\cos(t)^2 + \sin(t)^2 = 1$
11) $F = G\frac{m_1m_2}{egovar^2}$		
$F = G\frac{m_1m_2}{r^2}$ $q = \lfloor n_1/n_0 \rfloor$ $F = G\frac{m_1m_2}{r^2} = (G\frac{m_1}{r^2})m_2$ $F = \frac{q_1q_2}{r^2}\hat{r}$ $f(z) = \frac{u_1(z)}{u_2(z)}$	$F = G\frac{m_1m_2}{r^2}$ $F = G\frac{m_1m_2}{r^2} = (G\frac{m_1}{r^2})m_2$ $F = G \cdot \frac{m_1 \cdot m_2}{d^2}$ $F = G\frac{Mm}{r^2}$ $T = \frac{2gm_1m_2}{m_1+m_2}$	$F = G\frac{m_1m_2}{r^2}$ $F = G\frac{m_1m_2}{r^2} = (G\frac{m_1}{r^2})m_2$ $F = G \cdot \frac{m_1 \cdot m_2}{d^2}$ $F = G\frac{Mm}{r^2}$ $8000m = \frac{120m}{15\text{mil}} \times 1000$

APPENDIX H

Index Formats

The important file formats used in EgoMath's index (based on Solr) are described below:

- doc - frequencies and skip data;
- fdx - field index - contains pointers to field data;
- fdt - field data - the stored fields for documents;
- fnm - stores information about fields;
- pay - stores term payloads;
- pos - stores term positions;
- tim - stores term info in term dictionary;
- tip - the index into the term dictionary;
- tvd - contains information about each document that has term vectors;
- tvx - stores offset into the document data file.

APPENDIX I

EgoMath's Definition of the Underlying Mathematical Structure

Listing I.1: EgoMath's `algorithms.xml`

```xml
<algorithm name="basic">
  <transformation name="remove_optional" />
    <output priority="0" />
  <transformation name="order" />
    <output />
  <transformation name="eval" />
  <transformation name="approximate" />
  <transformation name="order" />
    <output />
  <transformation name="distributivity" />
  <transformation name="multiplying" />
  <transformation name="own_denominator" />
  <transformation name="order" />
    <output />
  <transformation name="replace_id_const_not_change" />
    <output />
  <transformation name="constants2const" />
  <transformation name="order" />
    <output />
  <transformation name="unknown2id" />
  <transformation name="order" />
```

```
    <output />
</algorithm>

<algorithm name="sim_sub" start="100">
    <output />
  <transformation name="remove_optional" />
  <transformation name="remove_simple_sup" />
    <output />
...

  <algorithm name="sim_eq" start="200">
    <output />
  <transformation name="remove_optional" />
  <transformation name="remove_simple_sup" />
  <transformation name="change_eq" />
    <output />
...
```

APPENDIX J

Extending Full Text Search Engine For Mathematical Content by Mišutka and Galamboš

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,
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.

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 Fig. 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.

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

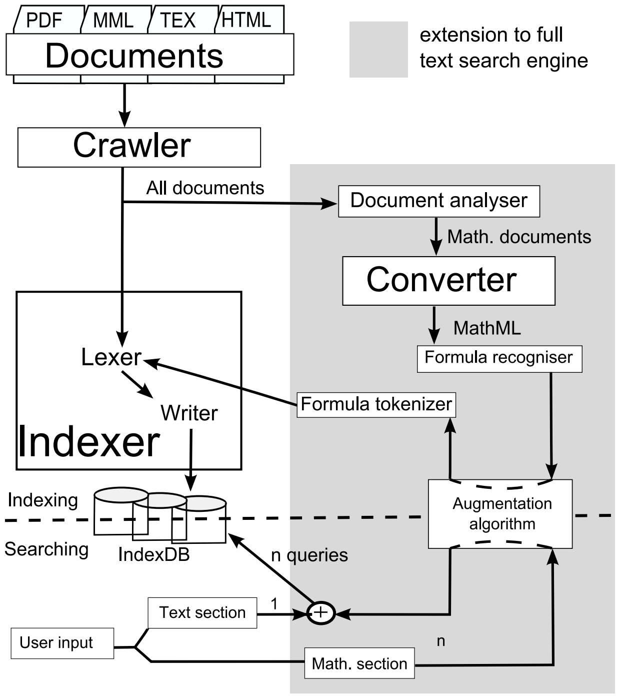


Figure 1. Architecture of an arbitrary full text search engine

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 $\varPi$ 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, $\mathit{\Pi}$, 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 ($1/x = \frac{1}{x} = (x)^{-1}$, $\pi = \mathit{\Pi} = Pi$), 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 $F^N := F_1 \times F_2 \times, ..., \times F_N$. The function Q produces N formulae $f_1, f_2, ..., f_N$ for one input formula. The number N is a predefined constant dependent on the generalisation and transformation rules. The function Q is defined as $Q : F \rightarrow F^N$, $Q(f) = [f_1, ..., f_N]$ 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 of textual words. This list is an example of transformation and generalisation rules:

1. *Partial evaluation:* $7 + a + 5 \xrightarrow{(converted\,to)} 12 + a$
2. *Approximate numerical constants:* $5.82 \doteq 6$
3. *Remove brackets using distributivity:* $a * (b + c) \rightarrow a * b + a * c$
4. *Multiply tokens:* $\frac{a+b}{2} * \mathit{\Pi} \rightarrow \frac{\mathit{\Pi} a + \mathit{\Pi} b}{2}$
5. *Assign each numerator its own denominator:* $\frac{\mathit{\Pi} a + \mathit{\Pi} b}{2} \rightarrow \frac{\mathit{\Pi} a}{2} + \frac{\mathit{\Pi} b}{2}$
6. *Replace constants with* const *symbol:*
   $74 + a^2 + b^2 \rightarrow const + a^{const} + b^{const}$
7. *Replace unknown constants, variables with* id *symbol:*
   $a^2 - b^2 + 2bc \rightarrow id_1^2 - id_1^2 + 2id_1id_2$
   or $\rightarrow id_1^2 - id_2^2 + 2id_1id_2$ ...

Another problem which must be addressed is that mathematically equivalent formulae with the same but permuted operators or operands would be considered

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

as different when compared letter by letter. Ordering algorithm 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 id or more $id_1, id_2, \ldots, id_n$ symbols, can be applied either on the whole formula at once or subsequently on all formula grains. All variables must be substituted for one 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 $id_1 + id_2$. 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 id_1, but we should be searching for id_2. If we apply the rule on each grain separately, we get $id_1 + id_1$ 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: $R(f_1) \geq R(f_2) \geq ... \geq R(f_N)$. 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 inputting 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 (32306 indexed formulae) totalling 99 MB, 2) part of the arXiv[3] (referred to as *ARXIV* in the remainder) with 1915 mathematical documents (852388 indexed formulae) totalling 252MB. 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].

[2] http://cnx.org/

[3] http://arxiv.org/

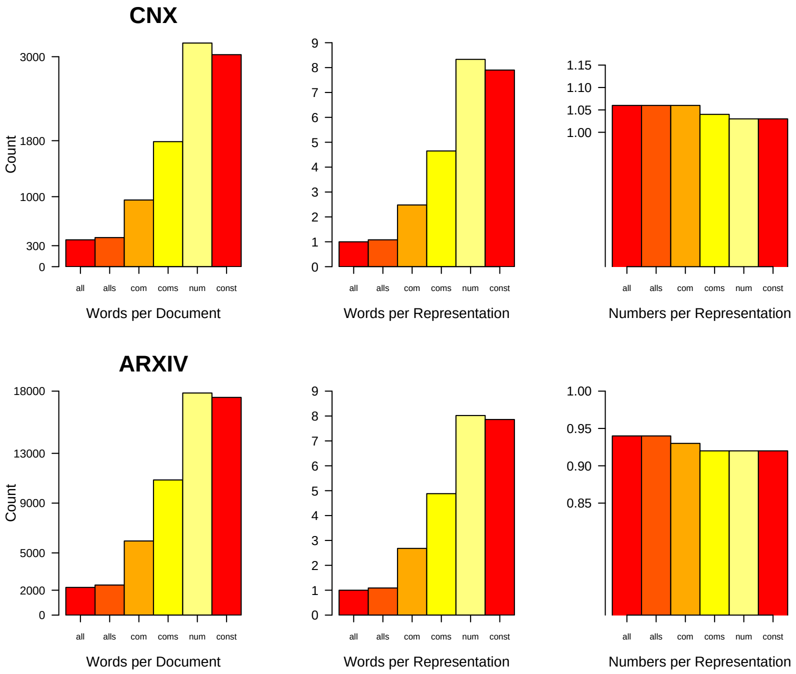


Figure 2. Average characteristics of document sets

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* - 4261940, 2) *CNX* - 161530. The difference between the maximum and minimum number of different representations is very small: 1) *ARXIV* - max 2121729 with *all*, min 2120319 with *num* and *const*, 2) *CNX* - max 78630 with *all*, min 78518 with *num* and *const*.

Average characteristics are shown in Fig. 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 is difficult to reasonably define similar subformulae, and 2) higher word count in one formula can theoretically cause performance problems. The last graph

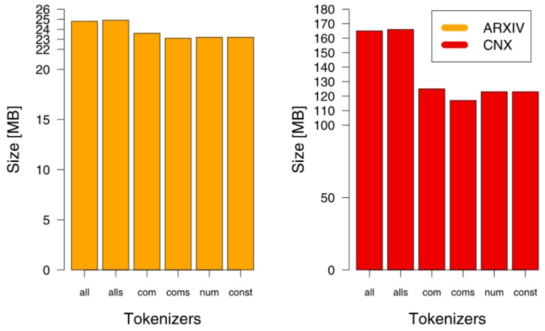


Figure 3. Index database size using different formula tokenizers

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.

Fig. 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 *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 Infty 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.

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

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
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, 121Ű-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)
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)

Index

List of Abbreviations

API	Application programming interface
FBA	Feature Based Algorithm Search Engine
FCA	Formal Concept Analysis
GUI	Graphical User Interface
Idf	Inverse document frequency
IR	Information retrieval
JNI	Java Native Interface
LSA	Latent Semantic Analysis
MML	MathML
MSE	Mathematical search engine
SE	Search Engine
SVM	Support Vector Machine
Tf	Term frequency
UI	User Interface
VSM	Vector space model

List of Figures

List of Listings