EIDGENÖSSISCHE TECHNISCHE HOCHSCHULE LAUSANNE
POLITECNICO FEDERALE DI LOSANNA
SWISS FEDERAL INSTITUTE OF TECHNOLOGY LAUSANNE

School of Computer and Communication Sciences
Computer Science Section | Distributed Systems Laboratory (LSIR)

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

CERN Digital Library

MASE(CDS): A Math Aware Search Engine (for CDS)

Master Thesis Project

Autor:
Arthur Oviedo

Supervisors:
CERN: Nikolaos Kasioumis
EPFL: Karl Aberer

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Chapter 2

Theoretical Background

Information Retrieval (IR) consists on the activity of identifying relevant documents from a collection based on some input parameters. Different models have been developed through the evolution of the field. In the subsequent sections, the most relevant to this work are presented.

2.1 Boolean Model

The Boolean model of Information Retrieval (BIR) was the first one to be developed and is currently one of the most used in the current implementation of IR systems because of its simplicity. It is based on the mathematical concepts of Boolean Logic and Set Theory. A formalization of this framework can be stated as follows:

Given a set of terms $T = t_1, t_2, ..., t_n$ which are the terms that will be indexed (In a standard search engine this can be the set of words in a language after some normalization steps), a set of documents $D = d_1, d_2, ..., d_n$ where $d_i \in \mathcal{P}(T)$ that the user will search for. A query element q is a Boolean expression over the index terms and the operators AND , OR and NOT. We define the predicate $Relevant(q, d_i)$ if a document d_i satisfies the query expression q. Finally the result for query over a collection of documents can now be written as $Result(q, D) = \{d|Relevant(q, d)\}$ The benefits of this model are its simplicity, both in its formalism and in its implementation. The main disadvantage of BIR are the fact that all terms are given the same importance which makes it difficult to rank results. Since all results have same importance, it is difficult also to limit the number of retrieved documents and therefore the output set can be difficult to process for the final user.

One of the key data structures for an efficient implementation of this model is the inverted index where for a given term, the index stores which documents contain it. This structure is used to efficiently solve queries by only scanning document that contain at least one of the terms in the query. Scanning a document can be done by using a standard (Also called forward) index, where for a given document, the index stores the terms associated with it. Both indexes can be implemented using hash tables.

2.2 Vector Space Model

The vector space model[?] which was developed between the 1960s and 1970s alongside the SMART Information Retrieval System. In this model, documents (from a collection D) and queries are represented as vectors d and q in a vector space where each index term corresponds to a dimension. If a term t_i occurs in a given query or document, its i-th coordinate will have a non zero value. Index terms can be keywords or phrases in the context of textual search or any other type of feature that can be extracted from the elements in a given specific context. The scoring of a document for a given query is computed as the cosine of the angle between both vectors which can be expressed as:

$$score(d,q) = cos\theta = \frac{d \cdot q}{\parallel d \parallel \parallel q \parallel} = \frac{\sum_{i=1}^{n} weight(d_i,d) * weight(q_i,q)}{\sqrt{\sum_{i=1}^{n} weight^2(d_i,d)}\sqrt{\sum_{i=1}^{n} weight^2(q_i,q)}}$$

Different weighting schemes can be employed, and the most standard one is called tf-idf (Term frequency - Inverse document frequency) where terms that appear less frequently in the corpus, are assigned higher values (Intuitively the term ”the” will be less useful for identifying a relevant document than the term ”Shakespeare” in the context of an English word based search engine). Therefore:

$$weight(t,d) = tf(t,d)idf(t)$$

There are several variations for each of the two components. Some approaches for tf include the raw frequency; Boolean frequency: $tf(t,d) = 1$ if t occurs in d or 0 if not; logarithmically scaled: $tf(t,d) = \log(frequency(t,d)+1)$ and the augmented frequency: $tf(t,d) = 0.5 + \frac{0.5 * frequency(t,d)}{\max\{frequency(t',d)|t' \in d\}}$. For computing the inverse document frequency the standard approach follows $idf = \log \frac{|D|}{1+\{d \in D : t \in d\}}$

The main drawbacks of this approach reside in the assumption that the terms are independent.

2.3 Probabilistic Relevance Model

The probabilistic relevance model was developed in 1976 by Robertson and Jones[?]. The similarity of a document d to a query q is given by the probability of d being relevant to q. It assumes there is a set R that is preferred to be the answer set for q. The model assumes that the document in R are relevant to q and the documents in $\bar{R}$ are not relevant. R then is the set that maximizes:

$$sim(d,q) = \frac{P(R|d)}{P(\bar{R}|d)}$$

The importance of this model is that it serves as a solid base for state of the art information retrieval models.

2.4 Okapi BM25

The Okapi BM25 ranking function is one of the current state-of-the-art models in information retrieval[?]. The actual scoring function is called BM25 and Okapi refers to the first system implementing this function in the 1980s. Given a query q as a vector of keywords $q_1, q_2, ..., q_n$ then the score for a document d is:

$$score(d,q) = \sum_{i=1}^{n} IDF(q_i) * \frac{frec(q_i, D)}{frec(q_i, D) + k_1 * ((1-b) + b * \frac{|D|}{avgdl})}$$

and

$$IDF(q_i) = \log \frac{N - df(q_i) + 0.5}{df(q_i) + 0.5}$$

where $frec(q_i, D)$ is the number of occurrences of keyword q_i in D, $avgdl$ is the average length (In keywords) of a document, and k_1 and b are free parameters. Usually $k_1 = 2$ and $b \in [0, 1]$ controls how important is the length normalization, being set normally to 0.75. N is the size of the collection, $df(q_i)$ is the number of documents that contain term q_i

A further variation that takes into account structure in the form of a document containing different fields and each of them having its own length and importance (boost) can be formulated as follows, known as BM25F can be expressed as follows:

$$score(d,q) = \sum_{i=1}^{n} \frac{weigth(q_i, d)}{k + weigth(q_i, d)} * IDF(q_i)$$

and

$$weight(q_i, d) = \sum_{j=1}^{m} \frac{frec(q_i, D, field_j) * boost_j)}{((1 - b_j) + b_j * \frac{|l_j|}{avgfl_j})}$$

where m is the number of different fields, $boost_j$ is the relative importance for each field, b_j is analogous to the b constant in the BM25 group of equations, $|l_j|$ is the length of the $field_j$ and $avgfl_j$ is the average length of the $field_j$.

2.5 Tree Edit Distance

As it will be discussed, one of the most common formats to represent mathematical equations, MathML, is a XML based format, and whose structure represents a rooted tree. Because of this, it makes sense to consider some models to compare and analyse mathematical expressions taking into account their natural tree structure. The most natural way of addressing this, is through the well studied abstract problem of Tree Edit Distance problem, where the idea is to compute a distance between a pair of trees. It should be noted that the development of this problem is very similar to the String Edit Distance problem. The Tree Edit Distance problem can be formulated as follows: Let Σ be a finite alphabet and let $\lambda \notin \Sigma$ be a special empty symbol. Finally let $\Sigma_\lambda = \Sigma \cup \lambda$. Given two trees T_1 and T_2 that are labelled (That is, each node N in the tree has assigned a label $l(N) \in \Sigma$) and ordered (That is, we are provided an order between sibling nodes), we define the following operations $(l(N_1) \rightarrow l(N_2))$ where $(l(N_1), l(N_2)) \in ((\Sigma \cup \lambda) \times (\Sigma \cup \lambda) \setminus (\lambda, \lambda))$:

Relabelling: If $l(N_1) \neq \lambda \wedge l(N_2) \neq \lambda$. We change the label of a node in the tree T

Deletion: If $l(N_2) = \lambda$. The children of N_1 become the children of N_1's parent (N_1 is not the root of the tree) and occupy its position in the sibling ordering.

Insertion: If $l(N_1) = \lambda$. This is the complement operation of deletion.

For each of this operations $\omega = (l(N_1) \rightarrow l(N_2))$, we assign a cost function $cost(\omega)$ and for a given sequence of this transformations $\Omega = (\omega_1, \omega_2, ..., \omega_n)$ we assign a cost function $cost(\Omega) = \sum_{i=1}^{i=n} cost(\omega_i)$ We also assume that $cost(\omega)$ is a distance metric. With the previous definition, the distance between T_1 and T_2 can be expressed now as $dist(T_1, T_2) = min \{cost(\Omega) | \Omega \text{ is a sequence of operations that transform } T_1 \text{ into } T_2\}$

A recursive formulation of the problem can be defined in the following way:

$$
\begin{aligned}
dist(T,T) &= 0\\
dist(F_1,T) &= dist(F_1 - v, T) + cost(v \to \lambda)\\
dist(T,F_2) &= dist(T, F_2 - w) + cost(\lambda \to w)\\
dist(F_1,F_2) &= \begin{cases} dist(F_1 - v, F_2) + cost(v \to \lambda)\\ dist(F_1, F_2 - w) + cost(\lambda \to w)\\ dist(F_1(v), F_2(w)) + dist(F_1 - T_1(v), F_2 - T_2(w)) + cost(v \to w) \end{cases}
\end{aligned}
$$

Where F is a forest, v and w nodes from a free, $F - v$ the forest F with the deletion of the node v, $T(v)$ the tree rooted at v and $F - T(v)$ the forest obtained by removing all the nodes in $T(v)$ from F

The presented set of equations give a simple way to compute the edit distance between a pair of trees, and the most known algorithms like Zhang and Shasha's[**?**] whose worst time case time bound is $O(|T_1|^2|T_2|^2)$, Klein's[**?**] that achieves a lower bound of $O(|T_1|^2|T_2|log|T_2|)$ or Demaine's[**?**], take into advantage the fact that the recursion relies in the solution of smaller sub-problems, so by carefully choosing which of this sub-problems to solve.

Chapter 3

Technical Considerations

In the previos chapter we presented different abstract models which can be suitable in the building of a math based search system. In this section, we present different aspects that need to be considered when mapping from these theoretical models to a concrete implementation.

3.1 Digital Representations of Mathematics

3.1.1 MathML

3.1.2 OpenMath

3.1.3 LaTeX

3.2 Math Extraction Tools

One of the main steps in any information retrieval task is the extraction of the searchable content from the collection of documents. At CDS the most common storage format for publications is PDF. A big part of the harvested documents in CDS comes from the ArXiv pre-prints service and from here, the source code (LaTeX) of the submissions is available for download. Smaller portions of other documents are stored in other formats like Microsoft Word, OpenOffice Documents. In practice, the main options from where to do the extraction from, are PDFs and LaTeXfiles. Also taking into account the previous discussion on the available format for representing mathematics, we focused our attention in tools that output MathML.
A conscious exploration of available software for extracting mathematical content from this two types of documents allowed us to identify different tools with different approaches. LaTeXfiles are easier to process and all of the

tools implement their own compiler that XML + MathML outputs instead of the typical. Since the original math equation is provided in the document, there is relatively little ambiguity in the equation, and the major issues that the systems struggle with is in the coverage of all the possible commands and packages that are available.
For extracting equations from PDF files, the process turns into an Optical Character Recognition (OCR). The reconstruction of the equation has to infer the mathematical structure based on physical positioning and size of the symbols, the length of the lines and so on.

Table 3.1 presents the main features of the reviewed software.

Software	**Type**	**Tested version**	**Input Format(s)**	**Output Format(s)**	**Licence Type**	**Latest Update**
LaTeXML[?]	Standalone Executable	0.7.9alpha	LaTeX	XML + MathML	Open Source	13 Feb 2014
TeX4ht[?][?]	Standalone Executable	2009-01-31-07	LaTeX	XML + MathML	Open Source	11 Jun 2009
LaTeXmathML[?]	Javascript Library	30-October-2007	LaTeX	XML + MathML	Open Source	30 Oct 2007
Maxtract[?][?]	Standalone Executable	v.1752	PDF	LaTeX/ Annotated PDF	Free to download	15 Nov 2012
InftyReader[?][?]	Standalone Executable (On Windows)	2.9.6.2	PDF, TIFF, BMP, GIF, PNG	LaTeX, XML + MathML	Commercial	22 Dec 2013

Table 3.1: Comparison of available Mathematical extraction tools

For extraction from LaTeXfiles we concluded that the best option is LaTeXML. The tool is still in current development and has lot of support from the academic community. The performance on a small dataset also showed that the results are consistently better than from the other tools. A more thorough test[?] also confirms that LaTeXML provides the best results. For PDF files, the trial version of InftyReader worked very well giving better results and processing successfully more files than Maxtract. As said previously, a big portion of the documents stored in CDS, come from pre-printing services like ArXiV were fortunately the source code of an article is available and because of this reason, our base set for this work will consist on extracting equations from LaTeXfiles. For licensing reasons, for this project we will not integrate our system with InftyReader for the moment and will proceed with LaTeXML as our extraction tool.

3.3 Computer Algebra Systems

Computer Algebra Systems (CASs) are software packages that allow performing symbolic transformations on mathematical objects. They allow to manipulate, simplify and analyse mathematical equations and offer interesting functionalities for a mathematical based search engine. The most important functionalities that we are looking for in such system are simplification and normalization of mathematical expressions and pattern matching. As discussed previously, the language for representing expressions will be MathML, so we also require that such system is able to import this format. Finally, we are also interested in integrating these functionalities in a bigger system, so integration trough APIs was also considered.

CAS	**Imports MathML**	**Expression normalization**	**Pattern Matching**	**License Type**	**Supported APIs**
Maple	Yes	Yes	Yes	Commercial	C, Java, VB
MatLab (Symbolic Math Toolbox)	Experimental	Yes	Yes	Commercial	C/C++, Fortran
Mathematica	Yes	Yes	Yes	Commercial	C, Java, .NET
Maxima	Experimental	Yes	Yes	Open Source	C++, Java (External Project)
SymPy	No	Yes	Yes	Open Source	Python

Table 3.2: Comparison of main CAS

Table 3.2 presents a comparison of the desired features in the main CASs. The systems that better fit our requirements are Maple and Mathematica. For licensing reasons, we will continue our project using Mathematica.

Chapter 4

Related Work

4.1 Status of MIR

4.1.1 TREC

4.1.2 TREC

4.1.3 CLEF

4.1.4 NTCIR

4.2 Mathematical based search projects

4.2.1 MIaS

MIaS[**?**] (Math Indexer and Searcher), currently, is one of the flagship projects in the indexing and searching of mathematical content. As most of the other projects, it processes documents in MathML format. It allows the user to include in the queries mathematical expressions and textual content. During indexing time, equations are transformed following a set of heuristics that include:

- Ordering of elements: Taking into account commutativity of certain operators (Addition, multiplication), elements are ordered such that $3 + a$ would be converted into $a + 3$ (Since the <mi> tag comes first than the <mn> tag)

- Unification of variables: This process takes into the account the structure of the equations in despite of the naming of the variables. Expressions like $a + b^a$ and $x + y^x$ would be converted into an expressions of the form $id_1 + id_2^{id_1}$ which would match.

- Unification of constants: This step consists of replacing all occurrences of constants (<mn> tags) by a const symbol.

The extracted tokens from a given equation consist of all its valid sub-expressions. The original expression is given a weight value of 1, and following sub-expressions are given smaller weights depending on how general or specific they are. The system, also as most of the current projects, is developed by using the Apache Lucene framework. The system adapts the default scoring equation such that the given weight is taken into account.

4.2.2 EgoMath

EgoMath[?] and its new version EgoMath^2[?], is a system oriented to index the mathematical content from the Wikipedia.org database. The processing steps before indexing are similar to the ones in MIaS (Rearranging of symbols, unification of constants). The extraction process started from a complete dump of the Wikipedia database and filtering only the math articles, which are identified by the "<math>" tag. This consists on around thirty thousand articles, from which 240.000 equations were identified. The processing step includes translating the equation from LaTeX into MathML format and then indexing both representation. Some additional effort is done into splitting large mathematical blocks like tables into single mathematical expressions.

4.2.3 DLMFSearch

4.2.4 MathWebSearch

MathWebSearch[?] (Currently on its 0.5 version) is a open-source search engine for mathematical equations. While most of the documentation relates on the architecture of the system and how they address scalability; the indexing technique is also very interesting. The system implements and idea proposed by Peter Graf in [?] called substitution tree indexing. This idea can be viewed as a generalization of the variable and constant unification employed by MIaS

4.2.5 LaTEXSearch

LaTEXSearch `http://latexsearch.com/` is a system developed by the scientific publisher Springer that allows to search over a database of around eight million latex snippets extracted from their publications. The system, unfortunately, is proprietary and no details are provided about the internals of the system.

4.3 Other related projects and initiatives

4.3.1 arXMLiv

Chapter 5

MASE(CDS)

In this chapter we present our solution called MASE(CDS), Math Aware Search Engine (For CDS). We present the key points of our system, and how they are related to previous built systems.

5.1 Features Extraction

Our main goal is to develop a system that compliments the already available search capabilities in Invenio and in particular the CDS instance. Our system was developed on top of the the Apache Solr/Lucene framework. CDS already uses Solr as a search engine layer, so developing MASE using this technology supposed a reasonable choice. One of the main steps while building an IR tool, is understanding the data and what kind of features can be extracted from it: If the system is based on text, then tasks as stemming, handling of synonyms, misspelling and so on, such be taken into consideration; if the system being developed is based on images, then examples of features can be histograms of colors, geometrical features and so on. During the literature review, we discovered that most of the projects handled the notational aspects of the expressions, that is, the real naming of the terms in a given equation and each project developed its own set of heuristics to handle the structural component.

5.1.1 Notational Features

Notational features correspond to the elements that relate to the actual naming of the variables, constants and numeric quantities found in an equation. Here we extract tokens that represent single leaf nodes (E.g. <mi>, <mn>, <mo> tags), and simple constructs like subscript and superscript elements, fractions, roots. We also extract bi-grams and tri-grams as analogue to that is done for text indexing in current systems.

After examining a set of relevant documents from our dataset, we identified some heuristics that could be applied to improve the recall of our system:

Unicode Normalizing

The first issue we discovered is that because of the big wide of different characters to represent the possible mathematical symbols, there is need for an automatic way to match different possible variations of a single character. An example of this situation is specification of an natural number, sometimes denoted by the letter N. In this case, at least three different common representations were identified: $N = 1$ (Using character with unicode code point 0x004e), $\mathcal{N} = 1$ (0xd835 0xdc41 or in latex expressed as \mathcal{N}) and $\mathbb{N} = 1$ (0x2115 or in latex expressed as \mathbb{N}).
For addressing this situation, we rely on the Unicode equivalence framework. In the unicode specification, sometimes the same character have two or more different code-points because of backwards compatibility with other encoding standards or because the same character has different essential meanings. Another group of equal characters with different code-points are pre-combined characters such as the Latin letter Ñ which has Unicode point 0x00D1 and also is the sequence of code-points 0x004E (Latin letter N) and 0x0303 (the combining character tilde). Taking this sample into account, Unicode defines the character composition and decomposition operations as replacing the decomposed representation with the pre-composed one and vice-versa. Finally, Unicode defines two types of equivalences between characters: Canonical and compatible. Canonical equivalent characters are assumed to have the same appearance and meaning. Compatible ones are allowed to have slightly different graphical representations but the same meaning in some contexts. One example of compatible equivalent characters is the Roman number 12 Ⅻ with code-point 0x216b and the sequence of characters XII with codepoints 0x0058 0x0049 0x0049. Taking into account the composition or decomposition of characters and the canonical or compatible equivalences, Unicode specifies 4 different forms of a given character:

- NFD: Normalization Form Canonical Decomposition
- NFC: Normalization Form Canonical Composition
- NFKD: Normalization Form Compatibility Decomposition
- NFKC: Normalization Form Compatibility Composition

Table 5.1 presents a small list of troubling characters classes that were identified in our datasets.

Character Name	**Visual Rendering**	**NFC**	**NFD**	**NFKC**	**NFKD**
LATIN CAPITAL LETTER A	A	0x41	0x41	0x41	0x41
LATIN CAPITAL LETTER A WITH MACRON	Ā	0x100	0x41 0x304	0x100	0x41 0x304
LATIN CAPITAL LETTER A WITH RING ABOVE	Å	0xc5	0x41 0x30a	0xc5	0x41 0x30a
ANGSTROM SIGN (0x212b)	Å	0xc5	0x41 0x30a	0xc5	0x41 0x30a

LATIN CAPITAL LIGATURE IJ	Ĳ	0x132	0x132	0x49(I)
				0x4a(J)
					0x49
					0x4a

CYRILLIC CAPITAL LIGATURE A IE	Ӕ	0x4d4	0x4d4	0x4d4	0x4d4
LATIN CAPITAL LETTER AE	Æ	0xc6	0xc6	0xc6	0xc6
DOUBLE-STRUCK CAPITAL N	ℕ	0x2115	0x2115	0x2115	0x4e
LATIN CAPITAL LETTER N	N	0x4e	0x4e	0x4e	0x4e
MATHEMATICAL ITALIC CAPITAL N	𝒩	0xd835 0xdc41	0xd835 0xdc41	0x4e	0x4e
ROMAN NUMERAL TWELVE	Ⅻ	0x216b	0x216b	0x58 0x49 0x49	0x58 0x49 0x49

LATIN CAPITAL LETTER X - LATIN CAPITAL LETTER I - LATIN CAPITAL LETTER I	XII	0x58
		0x49
		0x49
			0x58
			0x49
			0x49
				0x58(X)
				0x49(I)
				0x49(I)
					0x58
					0x49
					0x49

VULGAR FRACTION ONE QUARTER	¼	0xbc	0xbc	0x31(1)
				0x2044(/)
				0x34(4)
					0x31
					0x2044
					0x34

DIGIT ONE - SOLIDUS - DIGIT 4	1/4	0x31 0x2f 0x34	0x31 0x2f 0x34	0x31 0x2f 0x34	0x31 0x2f 0x34
DIGIT ONE - SOLIDUS - DIGIT 4	1/4	0x31 0x2f 0x34	0x31 0x2f 0x34	0x31 0x2f 0x34	0x31 0x2f 0x34
DIGIT ONE - FRACTION SLASH - DIGIT 4	1⁄4	0x31 0x2044 0x34	0x31 0x2044 0x34	0x31 0x2044 0x34	0x31 0x2044 0x34
THERE EXISTS	∃	0x2203	0x2203	0x2203	0x2203
THERE DOES NOT EXIST	∄	0x2204	0x2203 0x338	0x2204	0x2203 0x338
LATIN CAPITAL LETTER OPEN E	Ɛ	0x190	0x190	0x190	0x190
EULER CONSTANT	ℇ	0x2107	0x2107	0x190	0x190
PLANCK CONSTANT	ℎ	0x210e	0x210e	0x68	0x68
LATIN SMALL LETTER H	h	0x68	0x68	0x68	0x68
PLANCK CONSTANT OVER TWO PI	ℏ	0x210f	0x210f	0x127	0x127
LATIN SMALL LETTER H WITH STROKE	ħ	0x127	0x127	0x127	0x127
GREEK CAPITAL LETTER OMEGA	Ω	0x3a9	0x3a9	0x3a9	0x3a9
OHM SIGN	Ω	0x2126	0x2126	0x3a9	0x3a9
INTEGRAL	∫	0x222b	0x222b	0x222b	0x222b
DOUBLE INTEGRAL	∬	0x222c	0x222c	0x222b 0x222b	0x222b 0x222b
CONTOUR INTEGRAL	∮	0x222e	0x222e	0x222e	0x222e

Table 5.1: Characters with similar glyphs and/or semantics

Since our goal is to match as much as possible similar characters, our system employs the NFKD representation of a character. For a given token, we test whether if it is already equal to its NFKD. If not, for all the characters that are not combining nor control characters, we add the token into the same position as the original one. For example, given the token <mi>Å</mi>, this will lead to the sequence <mi>Å</mi><mi>A</mi> and the token <mi>Ⅻ</mi> will produce <mi>Ⅻ</mi><mi>X</mi><mi>I</mi><mi>I</mi>.

Synonym Expansion

Even though the Unicode normalization step work for a big part of the cases presented in table 5.1, there are still some groups of characters which would not be matched (Even partially), such as some types of integrals. For addressing these groups, a precomputed table was created for some general groups of characters which have some similar meaning. For each token that belongs to some of this predefined groups, a second token is created identifying the group. For example the token <mo>∮</mo> is expanded into <mo>∮</mo><mo>INTEGRALS</mo>, similarly the token token <mo>∫</mo> is expanded into <mo>∫</mo><mo>INTEGRALS</mo> and then both will have a partial match in the INTEGRALS token. Unicode specification does not define a strict rule to group similar characters based on their semantics. For general operators, we used the grouping provided by Xah Lee in his website[?].

Numeric Approximation

Some equations relate specific numeric quantities. While most of the previously explorer systems, handled <mn> tags by only representing them as a constant identifier, this approach is not suitable. A lot of articles in CDS relate to specific experiments under certain conditions, and require a high level of precision in the measurements. However, different experiments of the same phenomenon, can result in different measurements and we wanted to include this variability in our system. To handle this, when a token with a certain numeric quantity is discovered, we apply iterative rounding and index all the different approximations in the same position as the original element. For example the token <mn>2.0135</mn> will produce the following sequence of tokens: <mn>2.0135</mn><mn>2.014</mn><mn>2.01</mn><mn>2.0</mn>.

5.1.2 Structural Features

Algebraic structures

The first set of structures consist on purely algebraic relationships between the expressions. Inspired by predefined integral tables for different kinds of expressions, we decided to extract from a given equation, whether it has the occurrence of these kind of patterns. An example of this type of structures would be: $X\_ * (X\_ + A\_)$ where $X\_$ and $A\_$ can be any type of expression. Sample equations matching this pattern can be $C(C + 1)$ or $\frac{(\sin\theta\cos\theta)^2}{2a}(\frac{(\sin\theta\cos\theta)^2}{2a} + \tan\frac{\theta+\phi}{2})$. This structural analysis of equations is more powerful than the unification of variables and constants proposed in previously discussed projects because these patterns, as shown in the second example, can match an arbitrary complex expression. The drawback of our solution is the need to manually select representative and useful patterns. This selection was done based on standard based tables of integrals similar to the ones that Computational Algebraic Systems employ to compute them.

Non-formal mathematical variations

This set of structures represent common operations that cat appear in a given expression, but its identification is based on notation factors. A simple example of this can be the matrix decomposition $A = VDV^{-1}$ where the actual name of the variables A and V is not meaningful but the real importance is the fact that a given variable appears alongside its inverse in a matrix multiplication form. We would like to assign a partial match to any other kind of this decomposition say $B = MRM^{-1}$. these types of structures live in-between the notational and structural categories but, as it will be discussed, we identify them using Mathematica and encompass them as structural features.

Specific domain structures

Additionally, we identified patterns that are relevant and frequent in specific domains of the high energy physics.

5.1.3 Development with Lucene/Solr

This section explains how the described features were implemented in the `Apache Solr/Lucene` framework.

5.1.4 Integration with Mathematica

This section describes some aspects that should be had to be considered while implementing the integration with the Mathematica CAS. Our system uses the provided `JLink` library to create a connection with the underlying process `MathKernel`. The main class for communicating java code and `Mathematica` is `KernelLink` and the main used method was `evaluateToOutputForm(String expr, int pageWidth)` where `expr` is the expression to be evaluated and `pageWidth` affects the formatting of the result.

Pattern Extraction

Recursive interpretation

Error handling

Sometimes, after certain time, the connection between `KernelLink` and `Mathematica` will be lost launch an exception and following invocations would fails as well. Also, re-instantiating a KernelLink will yield an Exception. For this scenario, killing the associated processes to Mathematica was necessary:

```
killall -s 9 Mathematica
killall -s 9 MathKernel
```

or

```
taskkill /im Mathematica.exe
taskkill /im MathKernel.exe
```

Depending on the OS the system is running on.

With this, the next creation of a new KernelLink works properly and the subsequent invocations work as expected.

Timeout Handling

Unfortunately, checking the quality of a LaTeX document is a difficult job and users have a lot of freedom in the way they write their document since the only visible result is the rendered version. One example we found during our work is the following TeX snippet:

```
"\ \varphi\ are\ shown,\ as\ well\ as\ the\ weight\ and\ tension"
```

A human can easily identify that this corresponds to text and that should be inserted into a proper ”mbox” or ”text” tag. In this case, the snippet produced the following MathML code

```
<math><mrow><mi> </mi><mo></mo><mrow><mi>a</mi><mo></mo><mi>r</mi>
<mo></mo><mi>e</mi></mrow><mo></mo><mrow><mi>s</mi><mo></mo>
<mi>h</mi><mo></mo><mi>o</mi><mo></mo><mi>w</mi><mo></mo><mi>n</mi>
</mrow><mo>,</mo><mrow><mi>a</mi><mo></mo><mi>s</mi></mrow><mo></mo>
<mrow><mi>w</mi><mo></mo><mi>e</mi><mo></mo><mi>l</mi><mo></mo>
<mi>l</mi></mrow><mo></mo><mrow><mi>a</mi><mo></mo><mi>s</mi>
</mrow><mo></mo><mrow><mi>t</mi><mo></mo><mi>h</mi><mo></mo>
<mi>e</mi></mrow><mo></mo><mrow><mi>w</mi><mo></mo><mi>e</mi><mo>
</mo><mi>i</mi><mo></mo><mi>g</mi><mo></mo><mi>h</mi><mo></mo>
<mi>t</mi></mrow><mo></mo><mrow><mi>a</mi><mo></mo><mi>n</mi><mo>
</mo><mi>d</mi></mrow><mo></mo><mrow><mi>t</mi><mo></mo><mi>e</mi>
<mo></mo><mi>n</mi><mo></mo><mi>s</mi><mo></mo><mi>i</mi>
<mo></mo><mi>o</mi><mo></mo><mi>n</mi></mrow></mrow></math>
```

Mathematica took around 10 minutes to interpret it. In the ideal case we would be able to identify the malformed expressions and pass to Mathematica only well-formed equations to be processed but as this simple example shows, that is a very challenging task and may require more work. As a remedial situation, we used a fixed timeout to stop the processing of a given expression.

5.1.5 Integration with CDS

Chapter 6

Evaluation

...

6.1 Dataset

Our dataset consists on 12234 latex documents fetched from ArXiV. This documents correspond to the latest records harvested for CDS. The documents were processed using LateXML as described in the extraction process. From the given dataset, the simpler equations were filtered out (The ones containing one or two elements).

6.2 Setup

The machine used for experiments has a Intel Core i7 processor running at 3.4Ghz with 4 cores. The available RAM is 8Gb, and the Java Virtual Machine used for running Solr was run with the $-Xms2G-Xmx5G$ parameters. The complete specifications of the used software is presented in table 6.1

Software	**Version**
Operative System	Linux Mint 16 Petra (Ubuntu based)
Linux Kernel	3.11.0-15
Java Virtual Machine	1.7.0-45

Apache Solr	4.6
Python	2.7.5+

Table 6.1: Software used during evaluations

6.3 Results quality

In this section we are interested in evaluating the quality of the results provided by our system. The type of queries that MASE was designed for, are typical equations that can be found in a scientific publication. Out main user case is that a user of CDS is interested in finding documents inside the collection whose equations are related to the one that is provided.

6.3.1 Equations

The evaluated queries were provided by physics experts working at CERN from their own research topics. The queries were provided in LaTeXformat as the users used them in their own publications. Table 6.2 presents the set of queries that were evaluated, and a simple explanation about it.

Id	**Description**	**MathML code**	**Visual Rendering**
Eq1	Time-independent Schrödinger equation	`<math><mfenced close=")" open="("> <mrow> <mo>-</mo> <mfrac> <msup> <mi> </mi> <mn>2</mn> </msup> <mrow> <mn>2</mn> <msub> <mi>m</mi> <mi>e</mi> </msub> </mrow> </mfrac> <msup> <mo> </mo> <mn>2</mn> </msup> <mo>+</mo> <mi>V</mi> <mfenced close=")" open="("> <mi>r</mi> </mfenced> </mrow> </mfenced> <mi> </mi> <mfenced close=")" open="("> <mi>r</mi> </mfenced> <mo>==</mo> <mi>E</mi> <mi> </mi> <mfenced close=")" open="("> <mi>r</mi> </mfenced> </math>`	$\left(-\frac{\hbar^2}{2m_e}\nabla^2 + V(\mathbf{r})\right)\psi(\mathbf{r}) = E\psi(\mathbf{r})$

Table 6.2: Software used during evaluations

6.4 Querying Performance

6.5 Indexing Performance

For this set of evaluations, we were only interested in observing how the performance of the indexing stage is affected by the different types of features that are employed. We selected the first 100 records from our dataset and measured the total time it required to index all the elements in Solr using a python script and the solrpy library to do the communication. The default similarity in Solr was used. We also measured the index size at the end of each experiment. Our four scenarios were organized as follows:

- N : Only notational features over the original expression were used
- N+S : Notational and structural features over the original expression
- N+S+NN : Notational and structural features over the original expression and notational features over the normalized string using Mathematica's Simplify function
- N+S+FN : Notational and structural features over the original expression and notational features over the normalized string using Mathematica's Full Simplify function

Table 6.3 summarizes the results of this set of experiments. The first result is the significant increase in processing time when using Mathematica. Going from the N scenario to N+S represent an increase of almost 15x in the indexing time. Adding standard normalization increases the indexing time by a factor near to 1.44x and full normalization by a factor of 1.55x.

With respect to storage size, we see a small footprint (Less than 5%)in the total size by going from N to N+S. Adding normalization represents an increase by a factor of 1.78x and the size of the fully normalized index represents a smaller increase (1.76x)

We can observe that there is a small trade-off between time and storage size depending on which normalization mode is employed.

Feature Set	Total time (sec)	Index Size (kb)	Avg. time per equation	Avg. size of equation (kb)
N	58	13926	0.00524	1.237
N+S	884	14438	0.0785	1.282
N+S+NN	1273	25780	0.113	2.345

N+S+FN	1377	25548	0.122	2.269

Table 6.3: Indexing performance

Appendix A

Configuration files

A.1 Solr Schema

The Solr schema file `schema.xml` indicates which are the fields that are going to be used in this instance, and what type of analyzer is going to be used for each one. One can also define field types in case different fields want to have the same behaviour but conceptually having different meanings.

The following snippet add the fields declarations:

```
 <!-- Arthur Oviedo AO-->
<!-- Fields declaration using the field types declared below-->
 <field name="math_notational_field"
  type="math_notational_type"/>
 <field name="math_structural_field"
  type="math_structural_type"/>
 <field name="math_normalized_notational_field"
  type="math_normalized_notational_type"/>
 <field name="math_full_normalized_notational_field"
  type="math_full_normalized_notational_type"/>
 <field name="filename"
  type="string" indexed="true" stored="true"/>
 <field name="number_occurences"
  type="int" indexed="true" stored="true"/>
```

And the following one, defines the types and the configurations (Including the java class implementing the analyzer) for each one:

```
<fieldType name="math_normalized_notational_type"
class="solr.TextField"
```

```
positionIncrementGap="1"
indexed="true"
stored="true"
termVectors="true"
termPositions="true" >
<analyzer class="cern.ch.mathexplorer.lucene.analysis.analyzers.
SolrNormalizerNotationalAnalyzer"/>
</fieldType>

<!-- Field type for MathMl info
(Tokens that are extracted from the MathML representation)
over the normalized/simplified string using Full simplification-->
<fieldType name="math_full_normalized_notational_type"
class="solr.TextField"
positionIncrementGap="1"
indexed="true"
stored="true"
termVectors="true"
termPositions="true" >
<analyzer class="cern.ch.mathexplorer.lucene.analysis.analyzers.
SolrFullNormalizerNotationalAnalyzer"/>
</fieldType>

<!-- Field type for  structural features extracted from the equation-->
<fieldType name="math_structural_type"
class="solr.TextField"
positionIncrementGap="1"
indexed="true"
stored="true"
termVectors="true"
termPositions="true" >
<analyzer class="cern.ch.mathexplorer.lucene.analysis.analyzers.
SolrStructuralAnalyzer"/>
</fieldType>
```

A.2 Solr configuration file

The solrconfig.xml file specifies additional configuration parameters of the instance, including what handlers to use for each different type of request.

One specific type of handler is queryParser, and for using our specific query parser we include the following line of code:

```
 <queryParser name="mathqueryparser"
class="cern.ch.mathexplorer.lucene.query.MathQueryParserPlugin" />
```