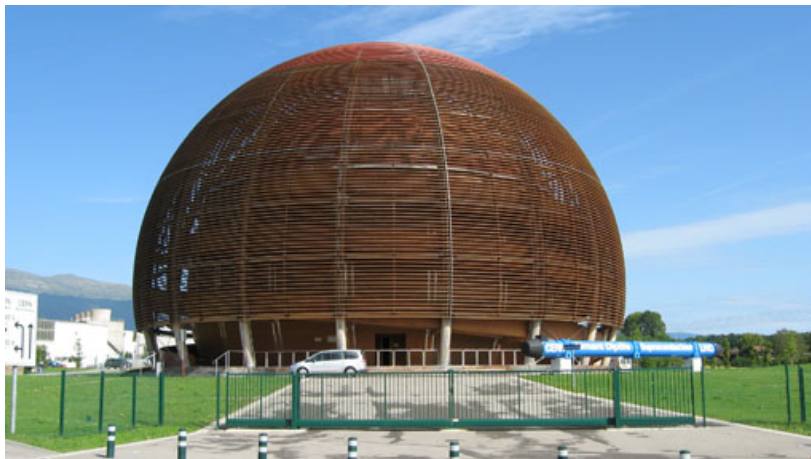


CERN Digital Library

Fast and scalable mathematical expressions
search engine

MASTER PROJECT



Autor:

Arthur OVIEDO

Supervisors:

CERN: Nikolaos KASIOUMIS

EPFL: Karl ABERER

Contents

1	Introduction	7
2	Theoretical Background	9
2.1	Boolean Model	9
2.2	Vector Space Model	9
2.3	tf-idf	9
2.4	Probabilistic Relevance Model	9
2.5	Okapi BM25	10
2.6	Tree Edit Distance	11
3	Technical Considerations	13
3.1	Digital Representations of Mathematics	14
3.1.1	MathML	14
3.1.2	OpenMath	14
3.1.3	LaTeX	14
3.2	Math Extraction Tools	14
3.3	From LaTeX to MathML	14
3.3.1	From PDF to MathML	14
3.3.2	Other projects	14
3.4	Computational Algebraic Systems	14
3.4.1	Mathematica	14
3.4.2	Mapple	14
3.4.3	Matlab	14
3.4.4	Octane	14
3.4.5	SAGE	14
4	Related Work	15
4.1	Status of MIR	15
4.1.1	TREC	15
4.1.2	TREC	15
4.1.3	CLEF	15

4.1.4	NTCIR	15
4.2	Mathematical based search projects	15
4.2.1	MIaS	15
4.2.2	EgoMath	16
4.2.3	DLMFSearch	16
4.2.4	MathWebSearch	16
4.2.5	LaTEXSearch	16
4.3	Other related projects and initiatives	16
4.3.1	arXMLiv	16
5	CERN Math Explorer	17
5.1	Information Retrieval Model	17
5.1.1	Notational Features	17
5.1.2	Structural Features	18
5.1.3	Development with Lucene/Solr	18
5.1.4	Integration with Mathematica	18
5.1.5	Integration with CDS	19
6	Evaluation	21
6.1	Dataset	21
6.2	Indexing Performance	21
	Bibliographie	23

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

This model 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.

2.2 Vector Space Model

2.3 tf-idf

2.4 Probabilistic Relevance Model

The probabilistic relevance model was developed in 1976 by Robertson and Jones[?]. The similarity of a document d_j to a query q is given by the probability of d_j 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:

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

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

2.5 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, \dots, q_n then the score for a document d is:

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

and

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

where $freq(q_i, D)$ 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:

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

and

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

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 $field_j$.

2.6 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 $\Omega = (\omega_1, \omega_2, \dots, \omega_n)$ we assign a cost function $cost(\Omega) = \sum_{i=1}^n cost(\omega_i)$. 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:

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

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[1] whose worst time case time bound is $O(|T_1|^2|T_2|^2)$, Klein's[2] 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 previous 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

3.3 From LaTeX to MathML

3.3.1 From PDF to MathML

Maxtract

InftyReader

OCR analysis

3.3.2 Other projects

3.4 Computational Algebraic Systems

In the previous section we discussed about algorithmical ways to transform, simplify and detect patterns in mathematical equations. Here, we explore what tools are available for performing this type of operations and how they are suitable for our system.

3.4.1 Mathematica

3.4.2 Mapple

3.4.3 Matlab

3.4.4 Octane

3.4.5 SAGE

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. Express-

sions 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 EgoMath²[?], 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

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

CERN Math Explorer

In this chapter we propose our solution for a scalable math based oriented search engine. We present the key points of our system, and how they are related to previous built systems.

5.1 Information Retrieval Model

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 using the Apache Solr/Lucene framework.

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 needed for an automatic way to match different possible variations of a single character. Some simple examples of this can be: A and its compliment \bar{A} in the case of set theory or ...

Character Rendering	Character Name	Latex Command
A	LATIN CAPITAL LETTER	A
Å	LATIN CAPITAL LETTER A WITH MACRON	Å
Å	LATIN CAPITAL LETTER A WITH RING	Å

- **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.
- **Grouping of similar operators:**

5.1.2 Structural Features

5.1.3 Development with Lucene/Solr

5.1.4 Integration with Mathematica

This section describes some aspects that should be considered while implementing the integration with the Mathematica CAS.

Recursive interpretation

Error handling

Sometimes after certain time Mathematica will crash, launch an exception and following invocations would fail 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

... 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 OS was Linux Mint 16 Petra, which is a fork based on Ubuntu, running on kernel 3.11.0-15.

6.1 Dataset

6.2 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. 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 ?? 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 10x in the indexing time.

Feature Set	Total time (sec)	Index Size (Mb)	Avg. time per equation	Avg. size of equation (Kb)
N	57	13.6	0.00507	1.237
N+S	574	14.1	0.05099	1.283
N+S+NN	1229	32.4	0.10918	2.947
N+S+FN	961	24.5	0.08537	2.229

Bibliography

- [1] Kaizhong Zhang and Dennis Shasha. Simple Fast Algorithms for the Editing Distance Between Trees and Related Problems. *Siam Journal on Computing*, 18:1245–1262, 1989.
- [2] Philip N. Klein. Computing the edit-distance between unrooted ordered trees. In *Proceedings of the 6th Annual European Symposium on Algorithms*, ESA '98, pages 91–102, London, UK, UK, 1998. Springer-Verlag.
- [3] Erik D. Demaine, Shay Mozes, Benjamin Rossman, and Oren Weimann. An Optimal Decomposition Algorithm for Tree Edit Distance. *ACM Transactions on Algorithms*, 6:146–157, 2007.
- [4] S. Henein. *Conception des guidages flexibles*. Presses Polytechniques et Universitaires Romandes, Lausanne, Switzerland, 2004.
- [5] J. Botsis. *Introduction la mcanique des solides et des structures*. Presses Polytechniques et Universitaires Romandes, Lausanne, Switzerland, 2004.
- [6] K E Petersen. Silicon as a mechanical material. *Proceedings of the IEEE*, 70(5):420–457, 1982.
- [7] L-S Fan, Y-C Tai, and R S Muller. IC-processed electrostatic micromotors. In *Tech. Dig. IEEE Int. Electron Devices Meeting (IEDM '88)*, pages 666–669, San Francisco, CA, USA, December 1988.
- [8] S Henein, P Spanoudakis, S Droz, L I Myklebust, and E Onillon. Flexure pivot for aerospace mechanisms. In *Proc. 10th European Space Mechanisms & Tribology Symposium (ESMATS) – European Space Agency Special Publication (ESA SP)*, number 524, pages 285–288, San Sebastian, Spain, 2003.
- [9] Sarajlic E. Yamahata C. Cordero M. Jalabert L. Iizuka T. Fujita H. Single mask 3-phase electrostatic rotary stepper micromotor. pages 1505–1508, 2009.

- [10] E. Sarajlic, C. Yamahata, M. Cordero, and H. Fujita. Three-phase electrostatic rotary stepper micromotor with a flexural pivot bearing, 2010. soon to appear.
- [11] E. Sarajlic, C. Yamahata, M. Cordero, and H. Fujita. An electrostatic 3-phase linear stepper motor fabricated by vertical trench isolation technology. *Journal of Micromechanics and Microengineering*, 19(7):074001 (7 pp.), July 2009.