An Architecture for Searching and Indexing Latex Equations in Scientific Literature

Ashish Lohia Kirti Sinha Soujanya Vadapalli Kamalakar Karlapalem
{ashish_lohia,kirtisinha,soujanya}@students.iiit.ac.in, kamal@iiit.ac.in
International Institute of Information Technology
Gachibowli, Hyderabad - INDIA.

ABSTRACT

Equations play a prominent role in developing theories and formulating problems. Most scientists find it easier to work with equations, rather than text documents or keywords. The problem of processing large number of equations, facilitation of search on equations and indexing equations is not trivial. In this paper, we develop a framework to manage, store and process equations. We define various equation similarity measures, propose various search and indexing techniques. A toolkit has been built to process equations and enable users to browse and search equations [9].

Keywords: information retrieval, equation search, similarity, indexing.

1. INTRODUCTION

Most of the data mining [5] work has concentrated on manipulating numerical/categorical data through techniques such as association rules, clustering and classification. Recently, there has been some work on extracting well-defined structures from free text or unstructured documents using HMM models [1]. But the role of text mining is much more higher than this. One of the major challenges in AI is comprehending text and few approaches towards this deal with text summarization, content management, and natural language processing [7]. Moreover, text databases are rapidly growing due to the increasing amount of information available in electronic forms (email, CD-ROMs and WWW)[5]. But with very large number of focused and subject-oriented documents such as arXiv dataset, though semi-structured, it should be possible to perform advanced data mining tasks. The existing services like search engines and indexing techniques on scientific literature employ popular text mining and text categorization techniques on text documents. But in case of scientific documents, equations form the core of content. It is well known that 'Mathematics is the language of Physics'. Equations in a document represent this language of Mathematics and there have been no attempts to exploit this language. For example, the possibility that the documents which contain the equation $e = mc^2$ are related to each other is quite high compared to the possibility of documents having the words 'energy', 'mass' and 'velocity' to be related to each other. This is so, because the language of English is more ambiguous than the language of Mathematics. Moreover, words have different meanings in different contexts, whereas an equation holds for a single concept. Though an equation could be used in other areas, its meaning of concept is still preserved. Similarly, algorithms/data structures form the core of computer science papers, chemical formulae form the core of chemistry literature and so on.

For a physicist, it is easier to remember the equation rather than the words to describe that equation. Some times, equations are remembered vaguely and with whatever the user remembers, getting all the equations that are similar is a non-trivial task. There are various aspects of an equation. An equation has a set of tokens, it follows a language and there is a structure present within it. Moreover, the tokens (operators and operands) have semantic meanings. This makes the task of finding similar equations from the traces of a users' remembrance very challenging. It is quite obvious that such a tool is a necessity which when coupled with the good text search engine, can help in rich semantic searching among the scientific documents. This motivates us to develop a framework for the equations present in physics documents. There has been no formal work on extracting and processing equations from these raw scientific text documents, to the best of our knowledge.

The main contributions of the paper are:

- Development of an equation extraction and storage system.
- Formalization of Equation Similarity and Equation Containment.
- Facilitating equation search and indexing.

ADVANCES IN DATA MANAGEMENT 2005
Jayant Haritsa, T.M. Vijayaraman (Editors)

The rest of the paper is organized as follows: Related work is discussed in section 2. Section 3 discusses the issues in building the system. Section 4 describes the equation processing techniques, equation annotation and some experimental results. Section 5 develops and evaluates equation similarity, section 6 illustrates the search utility for equations. Section 7 discusses the indexing of equations and Section 8 presents conclusions and future work.

2. RELATED WORK

There has been a similar attempt made in the direction of managing equations. The website eqndb.com [4] provides facilities for searching equations based on the name of the equation, names of the terms present in the equation (like force, mass) and displays the equations in the form of images. The equation database is manually built and it is understood that each equation is annotated with its respective name and a database of all the terms is maintained. These equations are not obtained automatically from the documents. Whereas in our approach, we aim to automate the process of equation extraction for semi-structured documents like latex.

Another related (but not similar) work on indexing and searching equations [3] proposes indexes and search techniques on MathML documents. This requires a set of wrapper modules that can convert equations found in text documents into MathML documents. Developing a wrapper module is a non-trivial task. The equation grammar is complicated and various authors have different styles of writing. For example, consider the following two equations:

$$F = G \times \frac{m_1 m_2}{d^2}$$

$$F = G \times \frac{mass_1 mass_2}{distance^2}$$

The above equations represent the gravitational force equation. In one of the equation, the tokens are single characters with subscripts, while in the other the tokens are a sequence of characters. To deal with such discrepancies, either there should be human-assistance or some meta-dictionary that stores all the possible strings representing a particular physical quantity, variable etc. The problem of converting these tagged-equation text into MathML documents is a tough one.

Since, currently most of the research literature is in the form of latex files (in the text format), we define the problem of extracting equations from these files (which are free-running text files), managing them, and supporting search and indexing facilities over the set of equations extracted. We are addressing the problem of building a human-assisted semi-automatic system that extracts and searches equations from the latex files, and then displays results in an appropriate fashion.

3. EQUATION EXTRACTION AND PROCESSING SYSTEM

The various issues in extracting and processing equations are:

Extraction (Data Cleaning): Equations are to be extracted from the repository of latex or xml documents. Only a few formats such as Latex or xml have explicit equations. On the contrary, it is difficult to extract equations from pdf, word files etc. This is a huge data cleaning task, wherein problems like missing labels, unmatched parenthesis, incomplete equations, and 'typedef' commands of latex make it very difficult to extract equations. Extraction is possible if we have latex or XML documents of the research papers i.e. text versions of the documents.

Equation Quality: Most of the times, we encounter trivial equations like $n = 1$, $x = 2$ and these equations are sensitive to the context in which they are used. These of equations do not have much relevance, but they occur in large numbers. There is a need to identify such equations based on the quality/relevance. The problem is to quantify the quality/relevance. One possible approach would be to generate an exhaustive grammar for these kinds of equations.

Indexing: The names of the equations and the related keywords of an equation help the process of indexing them by categorizing them based on the keywords. Equation classification/categorization is yet another problem that needs to be addressed here. In this paper, we propose a preliminary indexing structure on the equations based on the tokens present.

Equation Similarity: Equations have to be specified and transformed into a standard form. Searching mechanisms for equations are developed based on the structure, labels and mathematical expressions. A few similarity metrics are developed and used for the equation search.

Searching: Given a repository of "cleaned" equations in a standardized form, with a good storage and retrieval mechanism, a search engine can be developed. The search performed will extract all 'similar' equations. Equations that match the criteria are displayed along with the links to the documents in which they occur.

Experimental results: We addressed the problems mentioned above and have conducted experiments. The results of each of the problem is given within the corresponding section.

4. EQUATION PROCESSING TECHNIQUES

The architecture of an equation management system which automatically extracts and stores the equations, and provides features like searching and indexing is shown in figure 1.

In figure 1, the oval shaped components are the implemented modules. The rectangular components deal with some database or file, where the output of one/more modules is stored. The modules are coded in Perl language, known for its support for string/text operations. The programs are executed in

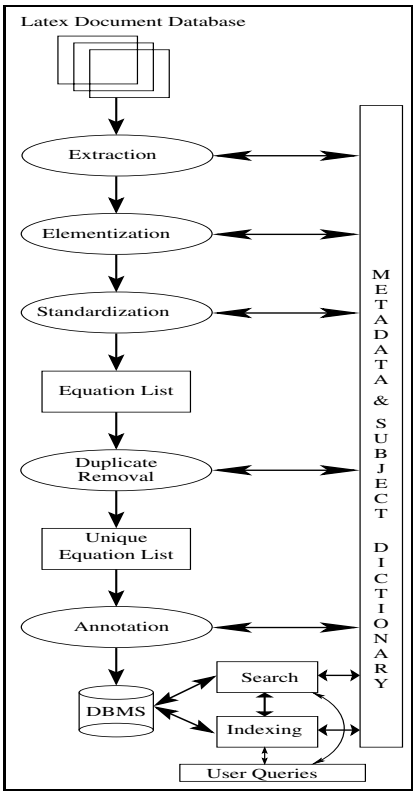

Figure 1: System for Equation Processing and Searching

Linux environment. A web-based search engine has been implemented, using Perl for CGI Programming. The output of the equation search engine is displayed in various formats: HTML, Latex and pdf. A complete working web-interface [9] for this system has been built. The techniques applied to process equations are described in the following subsections.

4.1 Dataset description

The arXiv dataset obtained from arxiv.org (kddcup 2003 dataset) is a repository of Physics papers in the topic of High Energy Particle Theory. There are 28,553 Latex documents, written over a period of 11 years (1992-2002). Each document is uniquely identified as $yymm < id >$ (yy-year, mm-month). Latex is a tag-based type-setting system. Tags are provided to mark the beginning and ending of various parts of the document and type-setting styles. These tags help in extracting different components of the documents. For example: an equation has \begin{equation}, \end{equation}, $ ... $ tags to mark the beginning and ending of the equation.

Latex files are highly susceptible to noisy, missing and inconsistent equation syntax due to various styles of writing. Equations are part of the research documents and are written as continuous text, but to use them effectively the equation needs to be structured. As expressed in [6], it can be seen that equation-data cleaning is a much more complicated and a necessary task.

An equation is a mathematical expression which follows rules of maths. Therefore, we can structure an equation using rules of grammar and standardize them by representing equations as prefix or postfix expression trees which include symbols like parenthesis. These trees capture the precedence order of the mathematical operators and implicit embedding of operators between the variables, which is essential to identify their structure. These trees/expressions can then be appropriately represented as an XML document using an XML Schema or DTD.

There has been considerable work that has gone into automatic text segmentation of postal address and bibliographic references [1]. This kind of text segmentation does not help in equation tokenization as we have no preset idea about the equation expression.

4.2 Issues in Cleaning Equations

Equation Identification: Extracting equations when they are specified using standard equation tags is easy to address. But when user-defined tags/macros surface in an equations, the processing is very complex and also a number of tedious and time consuming details need to be tackled. From the raw latex files, tags like "equation", "eqnarray" and "$$" format have to be checked in every line of the file. But in certain cases, $$ format is used to italicize text or to just write the variables. Therefore, equations are checked for the presence of some comparison operators (that is, $=$, $\neq$, $<$, $>$, $\leq$, $\geq$, $\equiv$, $\leftarrow$, $\rightarrow$, $\in$) to mark them as equations.

Identifying equations which have user-defined tags, requires standardization of equations. In latex, commands like '\newcommand', '\def', '\let' etc. are used to define macros/user-defined tags. A sample set of user definitions is given below (from paper "9202058"):

```
\newcommand{\EN}{\end{equation}}
\newcommand{\ENN}{\end{eqnarray}}
\newcommand{\ep}{\epsilon}
\newcommand{\LM}{\Lambda}
\newcommand{\EQ}{\begin{equation}}
\newcommand{\EQN}{\begin{eqnarray}}
\newcommand{\mat}[2]{\left(\begin{array}{#1}#2
                    \end{array}\right)}
```

Consider the above typedefs and note that the user-defined tags EN, ENN, ep, EQ, EQN just need to be substituted with their definition wherever they occur. But notice the '\mat' tag wherein, there are some arguments that are passed to the macro. In such cases, the arguments passed should be identified and replaced accordingly. Often, there is no standard format followed by the authors. The arguments are sometimes enclosed in {,} brackets and sometimes they are not. Identifying the parameters for these macros is not trivial. To address this issue, a thorough checking of how the command 'latex' (in Linux) compiles these kind of latex

commands is done. All such macro definitions are stored in a hash, where the key of each hash is the new user-defined tag. The value of each key is the macro definition of the corresponding new user tag. A hash file is built by scanning the whole document once and the occurrences of the new user-tags are substituted with their complete definition. This constitutes our **Equation Standardization** module. Given below is an equation from the document "9202058" with user-defined tags.

```
\EQ
\LM_{1}=\mat{c|c}{\ep_{1} & 0 \\ \hline
                        0 & 0  } ,
\quad
\LM_{2}=\ep \mat{c|c}{1_{N} & 0 \\ \hline
                         0  & {N\over 2}1_{2}} ,
\quad
\LM_{3}=\mat{c|c}{0  & \xi \\ \hline
                \eta & \ep_{2}  },
\label{eq:para}
\EN
and $\ep_{1}\in sl(N)$,
```

After standardization, the above equation is transformed into:

```
\begin{equation}
\Lambda_{1}=\left(\begin{array}{c|c}\epsilon_{1}
 & 0 \\ \hline 0 & 0 \end{array}\right),
\quad
\Lambda_{2}=\epsilon \left(\begin{array}{c|c}
 1_{N} & 0 \\ \hline 0 & {N\over 2}1_{2}
 \end{array}\right),
\quad
\Lambda_{3}=\left(\begin{array}{c|c}0 & \xi \\
 \hline \eta & \epsilon_{2} \end{array}\right),
\label{eq:para}
\end{equation}
and $\epsilon_{1}\in sl(N)$,
```

Equation Elementization: Identifying variables, constants, operators and latex labels within the equations is the next step towards structuring the equations. Some of the tokens are latex tags (which start with a \). An exhaustive list of all the latex tags is built. The tokens are checked from this list. When tokenizing, we even consider implicit operands like $\times$ (multiplication operator). For example, $3g$ is actually $3 \times g$. If a token is not present in the list, the equation is marked as 'erroneous' and is eliminated.

Incomplete file problem: A small percentage of latex files were incomplete. Within these files, there have been some inclusions of other files using the latex command 'input'. The files mentioned in \input{file.tex} were not available. These files usually contain the user-definition of macros thus making it difficult to standardize equations. One possible way to overcome the problem of missing macro-definitions is to predict what the new labels would mean, by learning from the rest of the documents. We have eliminated all such files, which have incomplete macro definitions.

Duplicate Equation Removal: After completion of the previous three steps, a complete list of equations has been obtained. This list of equations could have repetitions. The reason being that an equation could have occurred in more than one document. It is better not to store those duplicated equations in the database. A **duplication removal module** has been built which removes duplicate equations. The module takes a standardized equation and removes any tag that is used for graphical representation (i.e., extra spaces, vertical spaces etc.) For this task, a "near"-complete list of all graphical tags present in Latex has been compiled and using this list those tags are removed from the equations. After doing so, the equation is again tokenized and the tokens are matched in the same order as they occur, with the tokens of another equation. Consider two equations E_1 and E_2 which need to be checked if they are the same or not. Each equation is represented as an ordered sequence of tokens, ordered based on the occurrence, $E_1 = [tk_{11}\ tk_{12}\ tk_{1i} \ldots tk_{1n_1}]$ and $E_2 = [tk_{21}\ tk_{22}\ tk_{2i} \ldots tk_{2n_2}]$. A counter *error* is maintained. For every token tk_{1i} in E_1, the position j of first occurrence in E_2 (in the unchecked sequence) is noted. The *error* is set to $|i - j|$. So, if the tokens tk_{1i} and tk_{2j} are the same for $i = j$ (i.e., tokens at the same position in both equations), then the *error* is not incremented. But if there is a difference in position, the difference of i and j is added to *error*. At the end of checking, if *error* is 0, then the two equations match perfectly. Therefore only one copy of the equation is maintained. For each equation, a list of documents in which this equation occurs, is maintained.

Storing Equations: Each equation along with the associated list of documents is stored in mysql database in the form of $< \ eqid,\ eqn,\ < list\ of\ docs >>$. By storing the document list, we obtain the chronological order in which the equation has appeared in various documents, as the documents have time tags.

4.3 Experimental Results

The dataset on which the experiments were conducted is the arxiv dataset. The description of the dataset is given at the begining of this section.

Equation Extraction: Out of the 28,553 documents, 28,402 latex documents were complete(without any missing or erroneous syntax) and we could extract equations from them. 153 documents were eliminated due to the Incomplete file problem, mentioned in the section 4.2. The total number of equations extracted from these files are 3,794,389. To measure the accuracy of our equation extraction, two persons went through the pdf versions of a few latex files and counted the number of equations. The number of equations that they have observed is 719. The equation extraction module detected 727 equations. The number of equations extracted is more than the actual number of equations, because the module identifies a few kinds of variables as equations. In the latex file, some variables were represented as $ \tilde x_i $. In such equations, there was a comparison operator present (\tilde) and were thus identified as equation. In file "9209051", the number of extracted equations

FileName	Actual No.	Extracted No.
9201072	41	40
9204035	350	358
9205043	23	23
9207029	119	127
9208009	114	114
9209051	72	61

Table 1: Equation Extraction Accuracy

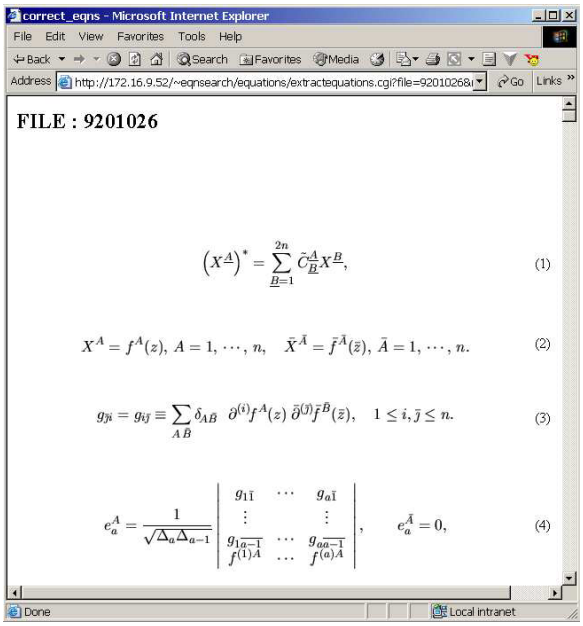

Figure 2: Screenshot of equation extraction (SMiLES System)

is less than the number of actual equations due the usage of a non-standard latex tag by the author to define equations. The file-wise statistics are given in table 1.

Duplicate Equation Removal: Due to lack of time, we could not perform the duplicate removal module on the complete equation list(3,794,389). We took a sample of around 100 documents chosen randomly, extracted equations. The number of equations extracted is 14402, of which the number of unique equations extracted is 11,671. The percentage of equations repeating is 18.9%. We performed the same on another sample of documents, of which the number of equations extracted is 11,365, the number of unique equations identified is 7,202 and the reduction rate is 36.63%.

5. EQUATION SIMILARITY

A search module should give "expected" accurate results with a reasonable response time. With the equations and their related information in place, we now formally define the concepts of similarity and containment of equations.

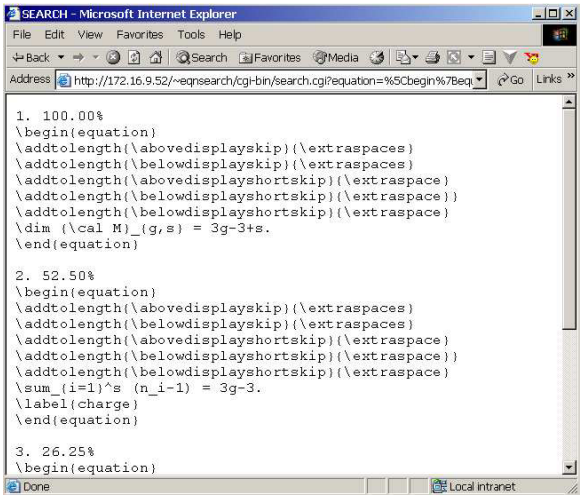

Figure 3: Screenshot of search result (SMiLES System)

The mechanism for identifying equations that are "very similar" to or "nearly contained" in an equation forms the core of the search module. The work is similar to that of [2], where "resemblance" and "containment" of web documents are handled. In [2], each document A is viewed as a set of unique *shingles* represented as $S(A)$. The resemblance r of two documents A and B is calculated as the number of common shingles between the two documents divided by the total number of unique shingles. The containment of A in B is defined as the number of common shingles divided by the number of shingles in document A.

In case of equations, there are different types of similarity measures. Equations could be compared based on their structure or on the variables/constants that are contained in the equation. Before discussing the similarity measures, the notation used in defining similarity is as follows:

E_i, the i-th equation in the database, is represented as a set of tokens $\{tk_{ij}\}$. A token is defined as the basic unit of an equation. The basic units of an equation are operators, constants and variables. Tokens could be a single character (like f,m) or a character string (like \lambda, \mu).

Identifying isomorphic equations: Consider the equation: $(a*b)+(c*d)$. This equation has seven more variants(due to change in order) but the equation is still the same, semantically. The variants are: $(c*d)+(a*b)$, $(b*a)+(d*c)$, $(d*c)+(b*a)$, $(b*a)+(c*d)$, $(a*b)+(d*c)$, $(d*c)+(a*b)$ and $(c*d)+(b*a)$. The prefix trees for all the above equations are different. Two trees are shown in figure 4. The non-leaf nodes are numbered and their numbers shown to their right in the figure.

Given a set of such equations, the problem is to find "similar" equations when there are such interchanges in subexpressions of the equation. This can be solved by arranging the nodes having commutative operators in the prefix

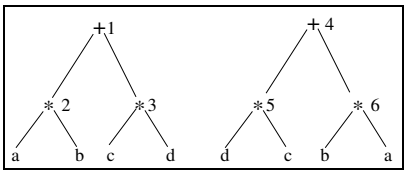

Figure 4: Expression trees for two equations

trees lexicographically. A prefix expression of any equation is represented as an n-ary prefix tree. Each node of n-ary tree is labeled with the subexpression formed with itself and its children nodes. The leaf nodes will be labeled with the variable/constant names. A sample prefix tree is shown in figure 4. Two child nodes are ordered according to the lexicographical order of their label names (i.e., subexpressions). For example, let us consider the second tree in the figure 4. The ordering of the whole tree is a bottom-up approach. Consider the leaf nodes d, c, b, a. At node 5, lexically $d > c$, thus the nodes are interchanged. Similarly at node 6, b and a are interchanged. Now at nodes 5 and 6 the subexpressions are $c * d$ and $a * b$. Thus, at node 4, the child nodes 5 and 6 are interchanged (lexicographically, $a * b < c * d$). The final tree is now transformed to the first tree. Although the two prefix trees are different, we obtained a unique tree for both such isomorphic variants. Note that this holds for commutative operators only. We keep the order as it is for non-commutative operators.

5.1 Equation Similarity Measures

Equation similarity measures form the basis of searching. The user submits a query equation Q to find out all the similar equations. Depending on the similarity measure used, the results will vary. The query Q is compared with all the equations in the database based on the metric chosen.

- **Exact Similarity**: Given a query equation Q and an equation E from database, the task is to find out if both the equations are exactly similar or not. Formally, let n_Q and n_E be the number of tokens in the equations Q and E respectively. Let the sequence of the tokens be represented as $T_Q = [T_{Qk}]$, for $k = 1 \ldots n_Q$, similarly for E. Let $n = \max(n_Q, n_E)$. The exact similarity measure between the equations Q and E, $s\_exact(Q, E)$ is defined as:

$$s\_exact(Q,E) = \frac{\sum_{k=1}^{n} \mid T_{Qk} \cap T_{Ek} \mid}{\sum_{k=1}^{n} \mid \{T_{Qk}\} \cup \{T_{Ek}\} \mid} \qquad (1)$$

  The s_exact is a number between 0 and 1. $s\_exact(Q, E) = 1$, when the equations match. The numerator is the comparison of the tokens at the corresponding positions in both the equations. At each position, only one token is present, thus $\{T_{Qk}\}$ is a single member set. The intersection between $\{T_{Qk}\}$ and $\{T_{Ek}\}$ gives ϕ if they do not match, else gives the same element. When both the equations match, the numerator and the denominator will be the same.The denominator checks if both the equations have the same number of tokens. If they have the same tokens, the union of both the equations will be the same as the equations' token set.

  The above metric is similar to that of Ordered Similarity (OS) measures based on the rank of presentation in the documents proposed in [8]. The OS measure captures the decreasing similarity measure with relative difference in the order of two sequences.

  A typical string matching technique is sufficient, where each equation is now treated as a single string and compared. But it is rare that two equations are written exactly the same way at various occurrences of the equations. The equations when displayed in pdf are the same, but their latex definitions turn out to be different because of graphic-specific tags. Additional tags are used to improve visual display of equation like \vspace, \addtodisplayspace etc. Moreover, apart from these graphical tags, two different tags mean the same symbol like \leq and \le ($\leq$). The equations thus need to be transformed, by removing all graphical tags and synonymous tags.

- **Partial Similarity**: Yet another type of similarity which is of interest is partial similarity. We would like to obtain equations which are "roughly" similar to the query equation. In this measure, the order in which the tokens occur is no longer considered, but only the common existence of tokens is checked. The definition of the measure is:

$$s\_part(Q,E) = \frac{\mid \{T_Q\} \cap \{T_E\} \mid}{\mid \{T_Q\} \cup \{T_E\} \mid} \qquad (2)$$

  The s_part varies from 0 to 1, and is 1, when both the equations have the same tokens.

- **Expression Similarity**: Apart from the token-matching variations described earlier, it would be semantically meaningful if the similarity is defined on the subexpressions of the equations. An equation E has a sequence of tokens represented as $[Tk_E]$ and the length of the sequence being n. A subexpression is defined as a subsequence of tokens of length three in the form of [tk_i tk_{i+1} tk_{i+2}], for i=1 to $n-2$. For example, consider the equation $a = b + c * d$, the complete set of all the subexpressions is $\{a = b, = b+, b+c, +c*, c*d\}$. The set of subexpressions of an equation E is represented as SE_E. Expression similarity of equations is defined as:

$$s\_expr(Q,E) = \frac{|SE_Q \cap SE_E|}{|SE_Q \cup SE_E|} \qquad (3)$$

A more robust technique for expression similarity is based on the pre(post)fix order of the equation expression trees. For an equation, the prefix is computed and is stored in the form of abstract syntax trees.

- **Variable Name Similarity**: Given a set of equations and a set of variables $V = \{v_i\}$, the task is to find those equations in which these variables are present. This similarity is defined as:

$$s\_var(E, V) = \frac{|Tk_E \cap V|}{|V|} \quad (4)$$

5.2 An example

Let us consider the following equations in the equation database:

$$F = ma \quad (5)$$

$$W = mg \quad (6)$$

$$P = mgh \quad (7)$$

$$K = \frac{1}{2}mv^2 \quad (8)$$

$$P = mv \quad (9)$$

$$S = ut + \frac{1}{2}at^2 \quad (10)$$

Now we test the above similarity metrics for the results. In the table 2, for every similarity metric and given query, the corresponding similarity values are calculated.

Sim. Metric	(5)	(6)	(7)	(8)	(9)	(10)	Query
s_exact	3/7	3/7	4/6	1/15	1	2/18	$P = mv$
s_part	1/3	1/5	2/11	3/13	3/9	6/12	$v = u + at$
s_expr	1/7	1/3	1	0	1/3	0	$P = mgh$
s_var	1/2	1	1	1/2	1/2	0	$\{m, g\}$

Table 2: Equation Similarity Measures

In the table 2, no equation has a value of 0 for *s_exact* measure with the query $P = mv$, because every equation has the symbol = in the second position. Note that equation $P = mv$ has *s_exact* value of 2/3 because p,=,$\times$ and m match. It is the same with the partial similarity too. In the expression similarity, the measure is 0, since there are no common expressions(subsequence of 3). In the variable similarity, the measure of equation 10 is 0 since neither m nor g are present. Please note that in the calculation of the above measures, implicit operators like $\times$ are also considered as tokens.

Structural similarity: The structure of the equation is represented as an abstract syntax tree. Under this similarity measure, two equations are termed 'similar' when the trees match in terms of number of child nodes etc. but the names of the variables or the labels of the leaf nodes are not taken into account. Trees of two equations could match structurally even if the token labels do not match. If this technique is applied to the above equations, given $F = ma$ as the query equation, the result would be equations 5, 6 and 9.

5.3 Issues

- Tokenizing operators and constants is comparatively easy compared to variable names. The variable names can be just anything, either a single character or a word. For operators and constants, Latex has tags to represent them. But consider the two equations $F = ma$, $Force = mass \times acc$. Both the equations mean the same, but the variable names differ. There is a need for a subject dictionary to be built which could help in the extraction of the variable names.

- For equations involving complex functions, integration and differentiation, it will be difficult to represent them in post-fix or pre-fix expressions. For such operations on many variables, the operator precedences need to be mentioned. The nodes in the prefix-trees for such operators might have variable number of child nodes.

6. EQUATION SEARCH

There are two kinds of search queries that a user can give:

1. *Tokens/labels as the query*: The user would just give the terms that need to be present in the equation. For these kind of queries, the equations are checked if they contain the terms and the results are ordered in descending number of matches. If the user wants to find out all the equations that contain the term acceleration due to gravity g, a regular lexical token matching is sufficient to obtain equations which have similar labels. The result in the above example case would be equations 6 and 7. The underlying assumption is that the alphabets/symbols associated with universal constants remain the same in whichever equation they are used. The variable names could change but not the universal constants. Hence a subject dictionary needs to be built which stores information about the universal constants, their labels along with regular/common labels used to represent regular terms in the subject.

   Token based search: In this search technique, the set of tokens from the search equation along with the number of times it occurs in that equation are extracted and stored in a hash. Then the search equation is compared with all the equations present in the database. The similarity measure based on the frequency of matching token is computed for every equation it (the search equation) is compared with. The similarity measure is determined by the summation of the frequency for each matched token. Let the search

equation be S and the equation being compared be E, then,

$$s\_token(S,E) = \sum_{t \in tokens(S)} match(t,E,S)$$

if $frequency(t,E) < frequency(t,S)$ then,

$$match(t,E,S) = frequency(t,E)/frequency(t,S)$$

otherwise, match is calculated as,

$$match(t,E,S) = frequency(t,S)/frequency(t,E)$$

When the frequency of a token in an equation is more than the frequency in the search equation indicates some dissimilarity between equations, thus the measure should be less in such cases. The maximum similarity value of the above measure is $|tokens(S)|$, when the equation to be compared has the same number of tokens with the same frequencies as in the search equation. The *s_token* similarity measure is slight modification of the measure *s_part*, where in *s_token*, the frequency of each token is taken into consideration. After obtaining the similarity measures for all the equations, the results are displayed in descending order of similarity values.

2. *Search by samples*: The user gives an equation as the sample equation, and expects similar equations to be displayed. The similarity technique discussed in the above point are of boolean type, i.e., (1) whether an equation has the same structure as the other one or not, and (2) whether the equation has the same labels contained within it. It is the case of exact matching of tokens/structure.

   An enhancement to the above kind of search would be obtaining a measure, which is a real number. There could be equations which would be a sub-expression of a larger equation. One of the methods that we implemented is 'subexpression matching', wherein we obtain all the possible subexpressions in the sample equation. A subexpression is defined as the expression that contains an operand, an operator and an operand, i.e., $x+3$, $y*z$ etc. This differs slightly from the expression similarity that we defined earlier. For the equation $a = b + c * d$, according to the definition expression in the section 5, there are 5 expressions. But here as we restricted the expression that it should have an operand, operator and operand, the subexpressions here are $\{a = b, b + c, c * d\}$. The expressions $\{= b+\}$ etc. violate the restriction.

   Consider the equation:

$$\dim \mathcal{M}_{g,s} = 3g - 3 + s. \tag{11}$$

   The sub-expressions in this equation are: $3g-3$, $3+s$. Note that $3g+s$ will not be considered a subexpression, because only the adjacent operands and operators are considered. These subexpressions are now matched with all the other equations in the database. The results are as follows in sorted order:

$$\dim \mathcal{M}_{g,s} = 3g - 3 + s.$$

$$\sum_{i=1}^{s}(n_i - 1) = 3g - 3.$$

$$l(\mathcal{O}_1 \ldots \mathcal{O}_n\rangle_g \equiv$$

$$\int_{\mathcal{M}_g} l \int \mathcal{O}_1^{(2)} \ldots \int \mathcal{O}_n^{(2)} \prod_{a=1}^{3g-3} \int \mu_a G \int \overline{\mu}_a \overline{G}\rangle_g.$$

The next improvement is to increase the length of the subexpression. Increasing the length makes the search more tight. This is because the basic unit of search is the subexpression and when the subexpression is large there will be very few results compared to a subexpression of smaller length.
Another similarity technique that could be employed, is to use the concept of 'Approximate Tree Matching'. The work done in [10] addresses the problem of matching trees approximately, based on the similarity of the tokens and edit distances. The distance between two such structures will be the dissimilarity between the two equations.

7. EQUATION INDEXING

The search techniques mentioned in the previous section, scan the whole equation database to search the equations similar to the search equation. This naive scanning of the equation database makes it infeasible to search the whole database of equations. We propose a simple indexing technique based on the tokens present in the equation. A token as defined in the previous sections is either an operator, constant or variable (single characters).

In token based index, we first obtain all the unique tokens occuring in all the equations. The number comes to around 111 as it includes standard latex tokens and small and capital letter alphabets . For each token, we obtain the list of equation ids in which this token occurs. Along with the equation id we store the frequency of the token in that equation. This index is analogous to the inverted index used in text retrieval. Having built this index, we now modify the search techniques based on this index.

For each equation, we obtain the list of tokens that occur in the equation.

In the token based search mentioned in section 6, the user either gives an equation (the tokens appearing in the equation are taken as the set of search tokens) or a few tokens as the input query and equations that are similar based on the tokens are to be obtained.

For improving the performance of the token based search (section 6), we use the token-based index. When a user gives a query , i.e. an equation , we extract the tokens from it along with the number of times they appear. Then for each token, we obtain the list of equation ids in which this token occurs and obtaining the frequency of the token in these equations is a one-step procedure, and compute the similar-

ity measure as explained in the Naive token based search. In the following subsection, we show the improvement in the search time by using this token-based index.

7.1 Experimental results

We have experimented and compared the performance of naive token-based search with the index-based search technique. We have chosen a few equations randomly with varying number of tokens in each equation. We gave these equations as input to both the search techniques. We found that the index-based search definitely out-performs the naive technique. This is so because, in the index-based technique, whatever tokens occur in the equations, we retrieve only those equations that have atleast one of these tokens. This reduces the search space thus enhancing the search performance. The results are given in the figure 5.

In the subexpression matching technique, we observed that the search was taking a very long time. Most of the time was utilized in obtaining the subexpressions. We made a slight modification to the search technique by storing all the subexpressions of an equation. This has made the search much faster than the naive subexpression matching. The graph in figure 6 depicts the same

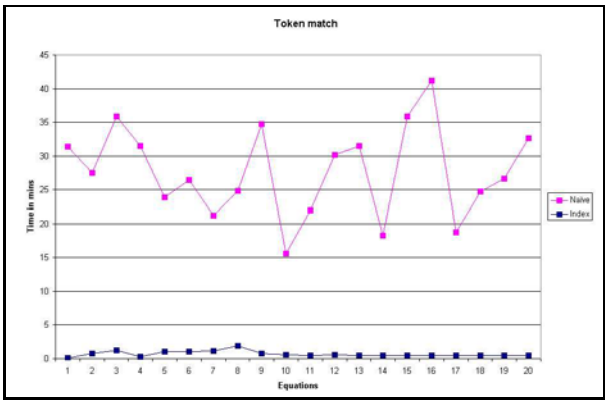

Figure 5: Comparison between the naive and index based Token Search

The time taken by various equations is slightly varying (as in figure 5 and 6) because of various factors. The equations we used to test are of different sizes (the number of tokens ranging from 5 to 25).Moreover, it even depends on the other load on the system due to other processes.

8. CONCLUSIONS

Equations are critical mathematical expressions that lay foundation to the development of science and make way for its transition to technology. Researchers in science and mathematics need to understand, communicate and contrast equations on a day to day basis.

In this paper, we develop a framework for extracting, cleaning, standardizing, loading and processing a set of equations from Latex papers. This is a non-trivial task which requires crucial solutions to each of the above tasks in processing equations. A system for searching and browsing equations has been built. Some of the issues tackled are: duplication equation removal and equation similarity measures . There is lot of scope for future work in terms of indexing equations, approximate/fuzzy search for equations and finer annotation of equations using machine learning techniques.

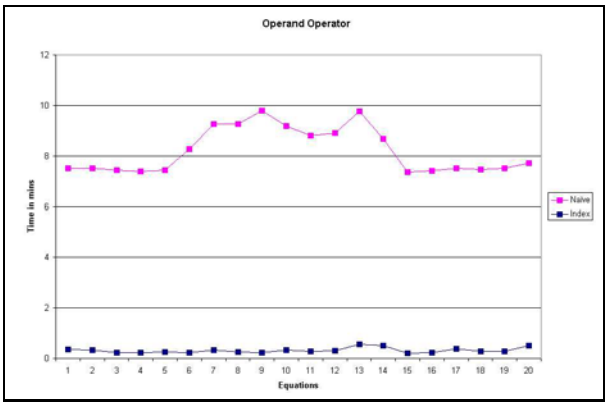

Figure 6: Comparison between the naive and index based Operand Operator search

9. REFERENCES

[1] V R Borkar, K Deshmukh, and S Sarawagi, *Automatic segmentation of text into structured records*, SIGMOD Conference, 2001.

[2] Andrei Z Broder, Steven C Glassman, Mark S Manasse, and Geoffrey Zweig, *Syntactic clustering of the web*, 6th International World Wide Web Conference on WWW, 1997.

[3] Stephane Dalmas and Marc Gatano, *Indexing mathematics with searchfor*, MathML International Conference, 2000.

[4] Eqndb, http://www.eqndb.com.

[5] Jiawei Han and Micheline Kamber, *Data mining, concepts and techniques*, Morgan Haufmann, 2001.

[6] R Kimball, *Dealing with dirty data*, (September, 1996).

[7] I Mani and M T Maybury, *Advances in automatic text summarization*, MIT Press, 1999.

[8] Christine Michel, *Ordered similarity measures taking into account the rank of documents*, Information Processing and Management, 2000.

[9] SMILES, http://cde.iiit.net/smiles/.

[10] Jason Tsong Li Wang, Kaizhong Zhang, Karpjoo Jeong, and Dennis Shasha, *A system for approximate tree matching*, IEEE Transactions on Knowledge and Data Engineering, Vol 6. No. 4, August, 1994.