Computación y Sistemas
ISSN: 1405-5546
computacion-y-sistemas@cic.ipn.mx
Instituto Politécnico Nacional
México

Pathak, Amarnath; Pakray, Partha; Sarkar, Sandip; Das, Dipankar; Gelbukh, Alexander
MathIRs: Retrieval System for Scientific Documents
Computación y Sistemas, vol. 21, núm. 2, 2017, pp. 253-265
Instituto Politécnico Nacional
Distrito Federal, México

ISSN 2007-9737

MathIRs: Retrieval System for Scientific Documents

Amarnath Pathak[1], Partha Pakray[1], Sandip Sarkar[2], Dipankar Das[3], Alexander Gelbukh[4]

[1] National Institute of Technology Mizoram, Aizawl, India

[2] Hijli College, Kharagpur, India

[3] Jadavpur University, Kolkata, India

[4] CIC, Instituto Politécnico Nacional, Mexico City, Mexico

{amar4gate, parthapakray, sandipsarkar.ju, dipankar.dipnil2005}@gmail.com, www.cic.ipn.mx/∼gelbukh

Abstract. Effective retrieval of mathematical contents from vast corpus of scientific documents demands enhancement in the conventional indexing and searching mechanisms. Indexing mechanism and the choice of semantic similarity measures guide the results of Math Information Retrieval system (MathIRs) to perfection. Tokenization and formula unification are among the distinguishing features of indexing mechanism, used in MathIRs, which facilitate sub-formula and similarity search. Besides, the scientific documents and the user queries in MathIRs will contain math as well as text contents and to match these contents we require three important modules: Text-Text Similarity (TS), Math-Math Similarity (MS) and Text-Math Similarity (TMS). In this paper we have proposed MathIRs comprising these important modules and a substitution tree based mechanism for indexing mathematical expressions. We have also presented experimental results for similarity search and argued that proposal of MathIRs will ease the task of scientific document retrieval.

Keywords. Natural language processing, information retrieval, MathIRs, indexing.

1 Introduction

Tremendous increase in scientific documents repositories and enormous amount of queries intended for retrieving such documents, necessitate the requirement of developing specialized tools and techniques which could handle such documents.

Scientific documents, unlike normal text documents, contain mathematical expressions and formulas which posses different form and meaning in different contexts. Several distinct appearing math expressions may actually turn out to be semantically similar and equally probable is the other situation where a given expression may resolve to several different interpretations in different contexts. For instance, the mathematical expression "h(x)" may get interpreted as function 'h' having 'x' as argument or a variable 'h' multiplied to another variable 'x'. Adding further, "$(x + y)^{1/2}$" and "$\sqrt{x+y}$ " look syntactically different but semantically they are the same. Process of indexing takes care of such visual and semantic ambiguities, present in mathematical expressions, by using a universal canonical representation for all the expressions followed by tokenization and storing of tokens in the index database to facilitate their retrieval on users' demand. In fact, the effective search for an expression inside a document and the accuracy of retrieval is determined by proficiency of the approach

adopted for indexing and semantic similarity measures used to compare query expression with the mathematical expressions inside documents. Thus, the task of mathematical information retrieval boils down to designing an effective indexing technique and similarity matching technique which could supplement and escalate the searching process.

Five major steps in indexing mathematical documents have been identified as: Preprocessing, Canonicalization, Tokenization, Structural Unification and Representation of Math for Indexing [23, 14]. However, these steps have undergone several changes and some intermediate steps have also been introduced as the indexing process has evolved with time.

Measuring semantic similarity between sentences or short texts is equally important for MathIRs and finds its application in many other natural language processing tasks such as machine translation, opinion mining, text summarization, and plagiarism detection. Need for semantic similarity detection at a very early stage is justified by the fact that the underlying documents may contain synonym, hyponym or hypernym of the query term rather than the exact query term. Comparatively less number of relevant documents will be retrieved, if semantic similarity between query expression and the indexed terms of the document is not detected by the searcher module.

Taking into account above facts and requirements, we have proposed an enhanced modular architecture for MathIRs which contains separate modules for semantic similarity detection between (text/math) contents in the query and (text/math) contents inside documents. We have also proposed an enhanced substitution tree based indexing mechanism which is presumed to overcome shortcomings of a similar indexing technique [18].

Rest of the paper is organized as follows: Section 2 describes past works on math information retrieval in general and indexing in particular. Section 3 briefs about semantic textual similarity. Section 4 details proposed system architecture for MathIRs and Section 5 describes features for similarity modules used in MathIRs. Section 6 contains experimental results for similarity modules and a comparison between winner score and our system's best score. Section 7 contains detailed discussion on working principle of indexing mechanism used in MathIRs and dataset and query set description for MathIRs. Section 8 concludes the paper and points directions for future research.

2 Related Work

Indexing of mathematical expressions, contained inside scientific documents, was first attempted using a substitution tree based method – node of the tree referring to a substitution and the leaves of the substitution tree referring to actual terms that have been indexed [3]. Moving along a path in the tree and applying substitution yields the indexed term. Such an indexing method reduces memory requirement, owing to the fact that we only need to store the substitutions and not the actual terms. It also promises paced indexing of terms and with such an indexing method in hand, the task of searching boils down to performing tree traversal in depth first fashion.

A later work identified several areas in which normalization is applicable for MathML[1] documents [8]. Documents converted to MathML usually contained MathML and XML errors, important of which include malformed tags, improper number of attribute values and child counts. An error correcting parser has been designed to address these issues which produces well formed XML documents by removing unrecognized elements, illegal attribute values and inserts missing entities, wherever required. Ambiguity in MathML documents has been further diminished by choosing single canonical representation for mathematical expressions and decimal numbers having multiple popular representation conventions.

An enhancement to the full text search engines to facilitate search for mathematical content, has been proposed in [9]. A normal text search engine uses a recognizer which recognizes text and parses it to sentences and words. Likewise, use of formula recognizer has been proposed which could recognize mathematical formulas

[1] https://www.w3.org/Math/

and convert it to MathML form, Presentation or Content[2]. Mathematical formulae are stored in postfix notation so as to avoid use of parenthesis and to facilitate similarity detection. Mathematical expression "(a+b)-(c+d)", for example, is first converted to a postfix notation "ab+cd+-" followed by storing of tokens "ab+", "cd+" and "–" in the index database. Use of postfix notation for indexing offers flexibility of searching similar expressions.

Design principle of Math Indexer and Searcher (MIaS) system [5] get motivation from the fact that conventional search engines are incapable and inefficient at handling math information retrieval from Digital Mathematics Library. To build an MIaS, different mathematical representations such as TEX/LATEX, MathML, OpenMath[3] and OmDoc[4] have been explored and their advantages and disadvantages have been discussed. Moreover, existing Math search engines have been compared on the grounds of indexing core, internal representation of document, used converters, query languages and query. Unfortunately, none of the existing systems could offer search for exact mathematical expression, equivalent expressions with different notation, sub-expressions and mixed mathematical contents.

Math Indexer and Searcher (MIaS) system was later implemented to facilitate indexing and searching of mathematical documents [23]. MIaS facilitates searching of not only the complete expression but also the sub-expressions and similar expressions by exploiting techniques of tokenization and structural unification respectively. Tokenization, a process of plucking out sub-expressions from a given input expression, is accomplished by traversal of the expression tree encoded using presentation MathML. Tokenization is followed by unification to generate generalized expressions for tokens and the same is achieved using ordering, variable unification and constant unification, whereby all the variables and constants present in the expression are substituted by single unified variable and constant.

Indexing process has been further delved into and the processes of ordering, variable unification and constant unification have been exemplified [24]. Ordering introduces an alphabetical order between arguments of an operator, whereas unification avoids risk treating two or more similar expressions, which differ only in terms of variable names or constant values, as different. Expressions, "$a+b^a$" and "$x+y^x$", are semantically same but differ in terms of variable names. Proposed system substitutes variable names with "ids", thus normalizing both these expressions to expression, "$id1 + id2^{id1}$" and causing the two expressions to match.

Canonicalization is act of transforming semantically similar MathML expressions into one common form [1]. Different ways of canonicalizing the MathML expressions have been discussed and detailed, salient ones include:

(i) removing unnecessary elements and attributes which only contribute to the appearance and not to the semantics,

(ii) minimizing the number of <mrow> elements,

(iii) subscript and superscript handling, and

(iv) avoiding use of entity "&#x2061" for function name.

Pre-processing, canonicalization and representation of math for indexing are salient issues related to indexing process of Math Indexer and Searcher (MIaS) [13]. Indexing process involves comparison of expressions to be indexed with the pre-indexed tokens and assigning weights to the token based on percentage similarity. The weights associated with the tokens later help in ranking the documents.

Design of MIaS has been further extended by incorporating modules for structural and operator unification [14] – two important features which were absent in the previous architectures of MIaS. An open source tool named MathML Unificator, is used to generate series of structurally unified variants of the input expression. Original expression and the structurally unified variants are added to the index database afterwards. However, appropriate care is taken not to overfill the index database with all possible variants of the input expression.

[2] https://www.w3.org/TR/WD-math-980106/chapter2.html
[3] http://www.openmath.org/overview/technical.html
[4] http://www.omdoc.org/

3 Semantic Textual Similarity

Retrieval of specific information requires an intelligent retrieval mechanism which is suitable in some specific domain. Many researchers have used semantic based approaches in information retrieval system to enhance performance of their system. Word based similarity is the primary similarity measure for measuring similarity between sentences. Similarity between two texts can be measured using four distinct similarity measurement approaches: lexical similarity, syntactic similarity, WordNet similarity and distributed semantic [6, 10]. Lexical similarity measure is concerned with words and is normally suitable for measuring semantic similarity of all languages [16, 17], whereas syntactic similarity measure is predominantly concerned with similarity of sentence structure. WordNet is an external resource which has been used to find similarity between sentences. On the other hand, distributed semantic similarity is primarily concerned with minimizing the difference between syntactic and semantic similarity. Machine learning tool (METEOR) and Levenshtein ratio find their application in measuring semantic similarity between sentences [15].

4 System Architecture

Our system architecture, shown in Figure 1, is enriched by three modules: Indexing, Math Processing and Similarity.

4.1 Indexing Module

Apache Lucene[5] is to be used for the purpose of indexing which is a free and open-source information retrieval software library, supported by the Apache Software Foundation and released under the Apache Software License. It performs operations such as indexing, reverse indexing and analysis on the documents fed to it, thus making the documents searchable and easy to retrieve.
Three core components of Lucene comprise Document Analyzer, Tokenizer and IndexWriter. Document analyzer analyzes the documents to recognize their content. Tokenizer separates the content into several small components called tokens which are written to the index database using IndexWriter. Writing to index is based upon positional information of the tokens in the document, a technique known as full inverted indexing. Lucene based search engines have been enabled to recommend similar documents based on current search interest of the user. In the recent release of Apache Lucene, few changes have been made with respect to fuzzy querying mechanism and query parser. Apache Lucene 6.4.0 is the latest version which is characterized by enhanced features of spellchecking, hit highlighting, advanced analysis and tokenization capabilities.

4.2 Math Processing Module

Mathematical documents originating from different sources attain heterogeneous forms such as .pdf, .tex etc. and hence it becomes necessary to pre-process the documents, converting them to one common MathML form. Automated tools such as Infty Reader[6], MaxTract[7], Tralics[8] and LatexML[9] are used to accomplish this task.

MathML, a form of XML concerned with encoding syntax and semantics of mathematical expressions, has got two major forms: presentation and content[10]. Presentation MathML markup comprises 30 elements accepting around 50 attributes. Most of these elements are concerned with syntax or layout of representation and can be categorized into three broad categories:

(i) Script elements: <msub>, <munder> and <mmultiscripts>,

(ii) Layout elements: <mrow>, <mstyle> and <mfrac>,

(iii) Table elements.

[5] https://lucene.apache.org/

[6] http://www.inftyreader.org/

[7] https://www.cs.bham.ac.uk/research/groupings/reasoning/sdag/maxtract.php

[8] http://www-sop.inria.fr/marelle/tralics/

[9] http://dlmf.nist.gov/LaTeXML/

[10] https://www.w3.org/TR/WD-math-980106/chapter2.html

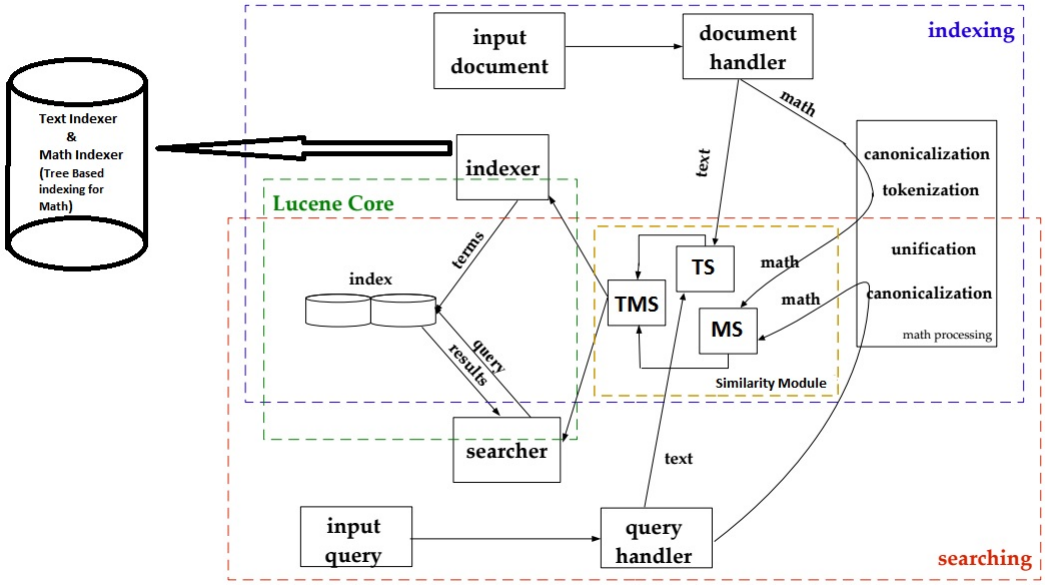

Fig. 1. System Architecture

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

Fig. 2. $(a+b)^2$ represented using Presentation MathML

```
<apply>
  <eq/>
  <ci>y</ci>
  <apply>
    <power/>
    <apply>
      <plus/>
      <ci>a</ci>
      <ci>b</ci>
    </apply>
    <cn>2</cn>
  </apply>
</apply>
```

Fig. 3. $(a+b)^2$ represented using Content MathML

Mathematical expression, "$(a+b)^2$" expressed using presentation MathML is shown in Figure 2.

Content markup comprises about 120 elements which accept around one dozen attributes and majority of them are used to represent operators, relations and named functions. <apply> is among most important content MathML elements, primarily used to apply functions to expressions and perform arithmetic and logical operations on collection of arguments. Mathematical equation, $y = (a+b)^2$, expressed using content MathML is shown in Figure 3.

Pre-processing is followed by canonicalization in which MathML expressions are normalized to single canonical form. Different ways of canonicalization include [1] :

(i) **Removing unnecessary elements and attributes:** Many elements such as <mspace>, <mpadded>, <mphantom> and <malignmark> contribute only to the appearance of documents and not to the semantics, thus being the preferred candidates for removal.

(ii) **Minimizing the number of <mrow> elements:** <mrow> element is used for grouping

other elements. Number of sub-expressions present in an expression can be found by counting the number of <mrow> elements. However, if a parent expression contains only one child expression then use of <mrow> element is redundant and the same should be omitted and avoided.

(iii) **subscript and superscript handling:** Use of <msubsup> element should be discouraged and <msub> and <msup> elements should be used instead. If both the functionalities are required then <msub> should be placed inside <msup>.

(iv) **Applying Functions:** Function name should be placed in the <mo> element and for the arguments we can use <mfenced> elements or parenthesis or both. Other ways of representing functions such as use of Entity '&#x2061' and use of function name inside <mi> element should be discouraged for the sake of avoiding ambiguity.

Most often, a user might be interested in search for a sub-expression rather than the complete expression. Tokenizer facilitates such functionality by plucking out sub-expressions from the input math expression and indexing them separately at a later stage. Consider a user querying for the expression "$\sqrt{(x+y)}$ " and a document 'd' containing Equation 1:

$$y = \frac{\sqrt{(x+y)}}{\sqrt{y}}. \tag{1}$$

MathIRs will not retrieve document 'd' for the given query expression, if Equation 1 has not been tokenized into tokens "$\sqrt{(x+y)}$ " and "$\sqrt{y}$ ".
Further, a user might probably be interested in similarity search rather than complete expression search or sub-expression search. This may correspond to a situation where expressions look syntactically different but semantically they are the same. Unifactor copes up with such an issue by generating all meaningful structural variants of expressions and sub-expressions and indexing them separately by assigning appropriately lowered weights to the structural variants [14]. Consider, for example, an input expression of the form "$x^2 + \frac{\sqrt{x}}{z}$". Different structural variants of this expression will be:

(i) $\odot^2 + \frac{\sqrt{x}}{z}$,

(ii) $\odot^2 + \frac{\odot}{z}$.

These variants, when indexed, are helpful in searching an expression which is similar to original expression but not exactly same. With this added functionality, if a user searches for the expression "$x^2 + \frac{\sqrt{x}}{z}$" then the document containing expression "$x^2 + \frac{\sqrt{x}}{z}$" is likely to get retrieved.

Final indexing of tokenized and structurally unified math formulas and sub formulas is proposed to be done using a substitution tree based indexing mechanism. An index tree is going to be used to systematically store math expressions. When a new expression is encountered, the existing tree is searched for the expression and if absent, extra nodes are appended in the tree to accommodate the expression. However, the tree need not be modified if parent expression of the given expression has already been indexed. Such an indexing mechanism is presumed to minimize our memory requirement by eliminating the need to store all the structurally unified variants of an expression.

4.3 Importance of Similarity Modules

Similarity modules constitute essential components of proposed architecture because user query may contain text and math contents and to retrieve documents it becomes necessary to compare (text/math) contents in the query with (text/math) contents of the underlying documents. These mandatory comparisons, if not performed, will result in retrieval of relatively less number of relevant documents.

4.4 Similarity Modules

The idea of text-text, math-math and text-math similarity matching is driven by the system architecture of MIaS [11], containing separate modules for Text-Text Entailment (TE), Math-Math Entailment (ME) and Text-Math Entailment (TME). A text or math query serves as 'hypothesis' and the content present in scientific documents serves as 'text'. An ME module, for example, will retrieve a document if some part of its mathematical content entails the mathematical query hypothesis. On the same grounds, we have proposed an architecture comprising TS, MS and TMS modules which perform text-text, math-math and text-math similarity matching for scientific document retrieval.

Our first module is TS module which defines the degree of similarity between text query and the text in document. The similarity between text and text can be computed using four approaches: lexical, syntactic, wordnet and distributed semantic. In knowledge-based approach the query is represented using synonym, hypernyms and hyponyms. Detailed description about the feature set and the experiment is provided in section 5 and section 6 respectively.

Our second module is MS module which finds the similarity between math query and the the document containing mathematical expressions. MS module helps in comparing canonicalized query expressions with canonicalized and structurally unified math expressions present in the document. A query expression might not match the exact math expression inside a document but may match one of its structurally unified versions which eventually results in retrieval of the underlying document. For example, the query expression "$a + \frac{x}{y}$" will match one of the structurally unified versions of "$a + \frac{\sqrt{x}}{y}$" and our MS module will retrieve the corresponding document.

Our third and final module is TMS module which finds similarity between text query and the mathematical expressions inside the documents. Text-Math similarity matching can be made feasible by assigning valid names to math expressions at the time of indexing. Text query will be matched to indexed names of mathematical expressions and if any suitable match is found, the corresponding document gets retrieved.

5 Features for Similarity Module

5.1 Cosine Similarity

Cosine similarity is a well known similarity measurement feature for finding similarity between two sentences. For our purpose we can represent each sentence using vectors. Similarity score of two sentences can be computed using dot product of corresponding vectors divided by the product of length of vectors.

Cosine similarity for two sentences, represented using vectors S_1 and S_2, can be computed using Equation 2:

$$S = \frac{S_1.S_2}{||S_1||.||S_2||}. \tag{2}$$

5.2 Levenshtein Ratio

Levenshtein distance, also known as edit distance, can be computed using three basic operations: insertion, deletion and substitution. To improve system performance, we can use normalization process to convert Levenshtein distance into Levenshtein ratio. Suppose 'a' and 'b' are two strings, then the Levenshtein ratio can be computed using the Equation 3:

$$\mathrm{EditRatio}(a,b) = 1 - \frac{\mathrm{EditDistance}(a,b)}{|a| + |b|}. \tag{3}$$

5.3 METEOR

METEOR is normally used for machine translation and it tries to include grammatical and semantic knowledge. METEOR score depends on outcome of types of matching: exact matching, stem matching, synonym matching and paraphrase matching [4]. Exact matching counts the similar words present in the hypothesis and referent part.

Stem matching matches words after they have been stemmed to their root words. Words such as organize, organizes, and organizing stem to the same root word 'organize' and hence they are said to match. Third module defines matching in terms of synonymy, meaning that two or more given

words are said to match if they are synonyms. The last kind of matching is based on the paraphrase matching between the hypothesis and reference translation.

5.4 Word2vec

In distributed semantic similarity approach, each word is represented using vector [12]. The main advantage of distributed semantic similarity is its ability to capture semantic representation of words after analyzing large corpora of text.

Distributed semantic similarity is based on the hypothesis: the meaning of a word is represented using the surrounding of that word [7]. Word2vec is a well-known model to produce word embedding and can be implemented using Gensim framework[11].

5.5 Jaccard Similarity

Suppose S and T are two sets containing words from two sentences. Jacardian similarity of the two sets is expressed using Equation 4:

$$JaccardSimilarity = \frac{|S \bigcap T|}{|S \bigcup T|} = \frac{|S \bigcap T|}{|S| + |T| - |S \bigcap T|}. \quad (4)$$

5.6 WordNet

WordNet is a lexical resource composed of synsets and semantic relations. Synset is a set of synonyms representing distinct concepts. Synsets are linked with basic semantic relations like hypernymy, hyponymy, meronymy, holonymy, troponymy, etc. and lexical relations like antonymy, gradation etc.

[11] https://radimrehurek.com/gensim/.

5.7 TakeLab

We have implemented TakeLab feature which was made available[12]. TakeLab system made use of various semantic features based on lexical, syntactic and external resources. The TakeLab 'simple' system obtained 3rd place in overall Pearson correlation and 1st for normalized Pearson in STS-12.

The source code was used to generate all its features namely *n-gram overlap, WordNet-augmented word overlap, vector space sentence similarity, normalized difference, shallow NE similarity, numbers overlap*, and *stock index features*.

TakeLab system predicts semantic similarity of two sentences using SVM regression approach which required full LSA vector space models provided by the TakeLab team. The word counts required for computing Information Content were obtained from Google Books Ngrams.

6 Text Similarity Module: Comparison between Winner Score and Baseline Score

Text based similarity module constitutes an important component of any Information Retrieval (IR) system. Sometimes relevant documents can't be retrieved, the reason being the absence of knowledge based similarity or lexical similarity between the text in the query and the text in document. The choice of an efficient and effective similarity measure for measuring text based similarity can boost the performance of our proposed system.

Our text similarity modules have been evaluated using SemEval 2016 dataset[13]. To find the semantic similarity, we have used two different approaches: lexical and distributed semantic similarity. Final semantic score has been computed after training using feed-forward neural

[12] http://takelab.fer.hr/sts.

[13] http://alt.qcri.org/semeval2016/task1/index.php?id=data-and -tools.

network and layer-recurrent network which are available in Matlab toolkit [14].

Five types of SemEval-2016 test datasets were used in monolingual sub-task: News, Headlines, Plagiarism, Post-editing, Answer-Answer and Question-Question.

The comparison between winner score[15] and our system's best score is given in Table 1. Besides, Table 1 shows that our system gives better result than TakeLab system which was the winner system in 2012. For plagiarism and question-question datasets, our system gave better results using distributional approach whereas best scores in answer-answer and headline datasets were obtained using lexical approach which is based on unigram matching ratio, Levenshtein ratio and METEOR.

7 Proposed Method for MathIRs

7.1 Motivation

For MathIRs, we have proposed substitution tree based indexing method which is an enhancement of the previous works in this regard [2, 18]. Substitution tree indexing was introduced by Graf where each node of the index tree represents predicates. Non-leaf nodes represent generalized substitution variables whereas leaf nodes represent specific ones. Depth first traversal of tree and applying substitutions in parallel yield specific predicates. Underlying idea has been further extended and the substitution trees have been used to represent mathematical expressions [18]. Working principle is exactly the same except for the fact that nodes in substitution trees now represent an expression in place of predicate. Each node of substitution tree is a Symbol Layout Tree (SLT) corresponding to some mathematical expression and edges of the SLT labeled as: NEXT, ABOVE and BELOW, indicating the terms next, super-scripted and sub-scripted to the given term, respectively.

[14] http://nl.mathworks.com/help/nnet/ref/feedforwardnet.html, http://nl.mathworks.com/help/nnet/ref/trainrp.html, http://nl.mathworks.com/help/nnet/ug/design-layerrecurrent-neural-networks.html.

[15] http://alt.qcri.org/semeval2016/task1/index.php?id=results.

Further simplification involves representing the nodes using substitutions to be applied while traversing the tree from root to leaf. Existing substitution tree indexing method offers minimum memory requirement benefits as it only stores the substitution at each node and not the complete terms. Memory is an important and deciding parameter while trying to build index databases and substitution tree indexing being best at minimizing memory requirements, motivates us to further investigate its structure. Besides, the proposed system in [18] suffers from obvious disadvantages of indexing only Latex formulas and representing the expressions purely based upon their syntax. Aim of our proposed system is to overcome such shortcomings and to further delve into structure of substitution trees for exploring other possible advantages.

7.2 Working Principle

The proposed system is characterized by underlying principles of SLT, substitution [18] and an improved indexing mechanism. SLT is 5 tuple (t, A, B, {x1,x2,...xn},N) where 't' refers to term, 'A' refers to above expression (or super-scripted expression to t), 'B' refers to below expression or (sub-scripted expression), "x_1,x_2,....x_n" refer to arguments and 'N' refers to next term. Any of these five tuples, if absent, are represented by 'ϕ'. Shorthand notation, (t) is used for SLTs containing only term and (t,N) for SLTs containing term plus next term, to avoid unnecessary ϕ entries. For example, "$x^2 + y_1$" represented using SLT tuples is shown in Figure 4.

$$(x, (2), \phi, \phi, (+, (y, \phi, (1), \phi, \phi)))$$

Fig. 4. SLT tuples for, $x^2 + y_1$

However, for the purpose of indexing mathematical expressions, SLTs are replaced by their substitution counterparts. In subsubsection 7.2.3, we have explained the difference between proposed indexing method and the indexing method used in [18], through an illustrative example.

Table 1. Performance comparison with other Systems on Monolingual Dataset

Corpus	Winner Score (Samsung Poland NLP Team)	TakeLab Score	Our System		Baseline Score
			Best Score	Approach	
answer-answer	0.69235	0.43301	**0.57454**	Lexical	0.48023
headlines	0.82749	0.55870	**0.72567**	Lexical	0.70749
plagiarism	0.84138	0.75086	**0.79877**	Distributional	0.76752
postediting	0.83516	0.64140	**0.81060**	Cosine	0.77196
question-question	0.68705	0.48409	**0.63612**	Distributional	0.43751

7.2.1 Dataset Description for MathIRs

We evaluated our system on the dataset from NTCIR-12 MathIR task[16]. Two corpora were used for the evaluation of our system: arXiv corpus and Wikipedia Corpus. arXiv corpus has been written by technical users assuming some level of mathematical understanding from users whereas Wikipedia corpus contain mathematical formulas written for normal users.

(i) **arXiv corpus:** arXiv corpus contains 105120 scientific articles converted from LaTeX to HTML+MathML format. Technical document from several arXiv categories[17] such as math, cs, physics:math-ph,stat, physics:hep-th are contained in arXiv corpus. Each document is divided into paragraphs resulting in 8,301,578 search units equating to 60 million math formulae.

(ii) **Wikipedia corpus:** Wikipedia corpus contains 319,689 articles from English Wikipedia converted into simpler XHTML format with images removed. These articles are not split into smaller documents with 10 % of the sampled article containing <math> tag and 90 % of the article containing complete English without <math> tag. There are around 590,000 formulas in this corpus encoded using presentation and content MathML. This corpus having uncompressed documents is 5.15 GB in size.

[16]http://ntcir-math.nii.ac.jp/.

[17]https://arxiv.org/list/math-ph/1101.

7.2.2 Query Set Description for MathIRs

arXiv corpus query set comprises 29 queries, 5 of them having only formula query whereas rest 24 having formula query plus keyword(s) and Wikipedia corpus query set comprises 30 queries – 3 of them having only formula whereas rest 27 having formula plus keyword(s)[18]. Queries are characterized and identified by unique IDs assigned to each of them. MathIRs is evaluated by comparing results of retrieval with Gold Dataset and checking values of the evaluation parameters obtained as result of comparison. Mean Average Precision (map), P_5 (Precision@5), P_10 (Precision@10) and bpref are some of the commonly used evaluation parameters.

7.2.3 Proposed Indexing Method

As per the previous indexing method [18], the expression to be indexed is first normalized by introducing an absolute ordering of variables present in the expression – for example, replacing 'x' by '$*_1$' , 'y' by '$*_2$' and so on. The expression to be inserted, is compared with the root and if match is successful then the indexing process is recursively called upon the children. If at any node, mismatch between the existing substitution at a node and the syntax of expression is found, a new node having a new substitution strategy is created to accommodate the expression. 'ϕ' symbol is used as root if no generalization is possible for all the expressions. Consider , for example, that expressions shown in Figure 7 are to be indexed and they appear in the documents in the same order as they have been written.

[18]http://ntcir-math.nii.ac.jp/

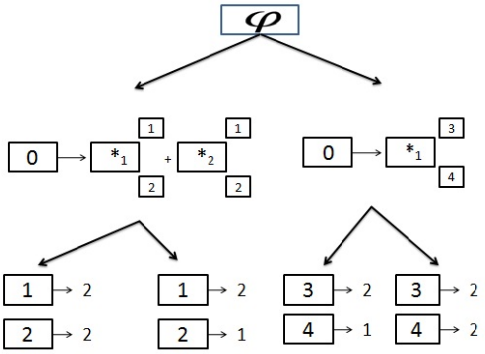

Fig. 5. Substitution Tree for expressions created using previous indexing method

In normalized forms of these expressions, 'x' is replaced by '$*_1$' and 'y' is replaced by '$*_2$'. Complete substitution tree for these expressions is shown in Figure 5.

The obvious disadvantage worth noticing about this indexing method is its specificity. The method treats two expressions, "$x_1^2 + y_1^2$" and "$x_2^2 + y_2^2$" as different although semantically they are the same – both representing sum of squares of two variables. Such an extreme specificity might not be desired in MathIRs. The user query "$x_1^2 + y_1^2$" might probably benefit from all the documents containing expressions similar to "$x_1^2 + y_1^2$". Such a shortcoming is probably an outcome of the syntax based approach of indexing which overlooks semantic similarity of two expressions, while indexing. Moreover, it only indexes math formulas written using LatexML, a serious bug which needs to be fixed keeping in mind enormously growing corpus of MathML documents.

As explained in subsection 4.2, our proposed substitution tree based indexing method comes into role after the math expressions have been tokenized and structurally unified. In our proposed indexing method, we index one of the structurally unified versions of an expression rather than indexing its normalized version. However, utmost care should be taken in selecting the structurally unified version of an expression otherwise we might end up generating an index tree which will yield absurd results or huge number of unnecessary results when searched for a query expression. Structurally unified version of expression should be such that it preserves syntax of the original expression. For example, if we have to index the expression: "$a + \frac{\sqrt{(b)}}{c}$", it will be a wise decision to select "$\odot + \frac{\sqrt{(\odot)}}{\odot}$" over "$\odot + \frac{\odot}{\odot}$", because the former one retains the syntax of original expression while the latter one completely deforms it.

For indexing the expressions: "x_1^2", "x_2^2", "$x_1^2 + y_1^2$" and "$x_2^2 + y_2^2$", we will structurally unify them to obtain "$\odot^2$", "$\odot^2$", "$\odot^2 + \odot^2$" and "$\odot^2 + \odot^2$", respectively. Substitution tree for these unified expressions is shown in Figure 6.

Substitution tree shown in Figure 6, has comparatively less number of branches, causing minimization of memory requirement and offering more generality. It also ensures generation of relatively more number of relevant results when searched for a query expression. Moreover, the proposed indexing method is able to index MathML formulas after they have been tokenized and structurally unified – an important feature which was absent in the previous indexing method [18].

8 Conclusion and Future Works

In this paper we have proposed system architecture for MathIRs which comprises separate

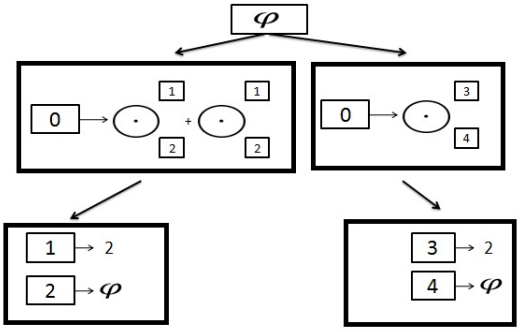

Fig. 6. Substitution Tree for expressions created using proposed indexing method

$$x_1^2,\ x_2^2,\ x_1^2 + y_1^2 \text{ and } x_2^2 + y_2^2$$

Fig. 7. Mathematical Expressions

modules for text-text, math-math and text-math similarity matching. An existing substitution tree based indexing [18] for mathematical expressions has been modified to make the indexing process more effective and generalized.

Unlike the previous approach [18], our proposed indexing method treats two semantically equivalent expressions as similar and the method is presumed to minimize memory requirement and increase number of relevant results. However, further modification in structure of substitution tree and careful selection of structural variant of an expression for indexing, can further improve effectiveness of MathIRs.

In addition, we plan to consider applying similarity measures based on the soft cosine measure [21] and syntactic n-grams [22, 19, 20].

Acknowledgement

The work presented here falls under the Research Project Grant No. YSS/2015/000988 and supported by the Department of Science & Technology (DST) and Science and Engineering Research Board (SERB), Govt. of India. The authors would like to acknowledge the Department of Computer Science & Engineering, National Institute of Technology Mizoram, India for providing infrastructural facilities and support. The fifth author acknowledges the suuport of Mexican Government though Instituto Politécnico Nacional SIP grant 20172008.

References

1. **Formánek, D., Líška, M., Růžička, M., & Sojka, P. (2012).** Normalization of digital mathematics library content. *Proceedings of the Conference on Intelligent Computer Mathematics (CICM)*, Bremen, Germany, pp. 91–103.
2. **Graf, P. (1995).** Substitution tree indexing. *Proceedings of the International Conference on Rewriting Techniques and Applications*, Springer, Kaiserslautern, Germany, pp. 117–131.
3. **Kohlhase, M. & Sucan, I. (2006).** A search engine for mathematical formulae. *Proceedings of the International Conference on Artificial Intelligence and Symbolic Computation*, Springer, Beijing, China, pp. 241–253.
4. **Lavie, A. & Agarwal, A. (2007).** Meteor: An automatic metric for mt evaluation with high levels of correlation with human judgments. *Proceedings of the Second Workshop on Statistical Machine Translation*, StatMT '07, Association for Computational Linguistics, Stroudsburg, PA, USA, pp. 228–231.
5. **Líška, M. (2010).** Mathematical indexing and querying. Online: https://mir.fi.muni.cz / presentations / liska-04-05-2011.pdf.

6. **Lynum, A., Pakray, P., Gambäck, B., & Jimenez, S. (2014).** NTNU: measuring semantic similarity with sublexical feature representations and soft cardinality. *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval)*, volume 14, Dublin, Ireland, pp. 448–453.
7. **Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013).** Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, pp. 3111–3119.
8. **Miner, R. & Munavalli, R. (2007).** An approach to mathematical search through query formulation and data normalization. In *Towards Mechanized Mathematical Assistants*. Springer, pp. 342–355.
9. **Mišutka, J. & Galamboš, L. (2008).** Extending full text search engine for mathematical content. *Towards Digital Mathematics Library*, pp. 55–67.
10. **Pakray, P., Bandyopadhyay, S., & Gelbukh, A. (2011).** Textual entailment using lexical and syntactic similarity. *International Journal of Artificial Intelligence and Applications*, Vol. 2, No. 1, pp. 43–58.
11. **Pakray, P. & Sojka, P. (2014).** An architecture for scientific document retrieval using textual and math entailment modules. *Recent Advances in Slavonic Natural Language Processing*, pp. 107–117.
12. **Pennington, J., Socher, R., & Manning, C. D. (2014).** Glove: Global vectors for word representation. *EMNLP*, volume 14, pp. 1532–1543.
13. **Růžička, M., Sojka, P., & Líška, M. (2014).** Math indexer and searcher under the hood: History and development of a winning strategy. *Proceedings of the 11th NTCIR Conference on Evaluation of Information Access Technologies*, Tokyo, Japan, pp. 127–134.
14. **Růžička, M., Sojka, P., & Lıška, M. (2016).** Math indexer and searcher under the hood: Fine-tuning query expansion and unification strategies. *Proceedings of the 12th NTCIR Conference on Evaluation of Information Access Technologies*, Tokyo Japan, pp. 7–10.
15. **Sarkar, S., Das, D., Pakray, P., & Gelbukh, A. (2016).** JUNITMZ at SemEval-2016 task 1: Identifying semantic similarity using Levenshtein ratio. *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval)*, Association for Computational Linguistics, San Diego, California, pp. 702–705.
16. **Sarkar, S., Pakray, P., Das, D., & Gelbukh, A. (2016).** Regression based approaches for detecting and measuring textual similarity. *Mining Intelligence and Knowledge Exploration: 4th International Conference (MIKE)*, Springer International Publishing, Mexico City, pp. 144–152.
17. **Sarkar, S., Saha, S., Bentham, J., Pakray, P., Das, D., & Gelbukh, A. (2016).** NLP-NITMZ@DPIL-FIRE2016: language independent paraphrases detection. *Shared task on detecting paraphrases in Indian languages (DPIL), Forum for Information Retrieval Evaluation (FIRE)*, Kolkata, India, pp. 256–259.
18. **Schellenberg, M. (2011).** *Layout-based substitution tree indexing and retrieval for mathematical expressions*.
19. **Sidorov, G. (2013).** *Non-linear construction of n-grams in computational linguistics: syntactic, filtered, and generalized n-grams [in Spanish]*.
20. **Sidorov, G. (2014).** Should syntactic n-grams contain names of syntactic relations? *International Journal of Computational Linguistics and Applications*, Vol. 5, No. 1, pp. 139–158.
21. **Sidorov, G., Gelbukh, A. F., Gómez-Adorno, H., & Pinto, D. (2014).** Soft similarity and soft cosine measure: Similarity of features in vector space model. *Computación y Sistemas*, Vol. 18, No. 3, pp. 491–504.
22. **Sidorov, G., Velasquez, F., Stamatatos, E., Gelbukh, A., & Chanona-Hernández, L. (2014).** Syntactic n-grams as machine learning features for natural language processing. *Expert Systems with Applications*, Vol. 41, No. 3, pp. 853–860.
23. **Sojka, P. & Líška, M. (2011).** The art of mathematics retrieval. *Proceedings of the 11th ACM symposium on Document engineering*, Association for Computing Machinery, Mountain View, California, pp. 57–60.
24. **Sojka, P. & Líška, M. (2011).** Indexing and searching mathematics in digital libraries. *Proceedings of the International Conference on Intelligent Computer Mathematics*, Springer, Bertinoro, Italy, pp. 228–243.

Article received on 08/03/2016; accepted on 23/05/2017.
Corresponding author is Partha Pakray.