OPMES: A similarity search engine for mathematical content

Wei Zhong and Hui Fang

University of Delaware, Newark, DE, USA
{zhongwei, hfang}@udel.edu

Abstract. This paper presents details about a new mathematical search engine, i.e., *OPMES*. This search engine leverages operator trees in both representation and relevance modeling of the mathematical content. More specifically, *OPMES* represents mathematical expressions using operator trees, and then indexes each expression based on all the leaf-root paths of the generated operator tree. Such data structures enable *OPMES* to implement an efficient two-stage query processing technique. The system first identifies structurally relevant expressions based on the matching of the leaf-root paths, and then further ranks them based on their symbolic similarity to the query.

1 Introduction

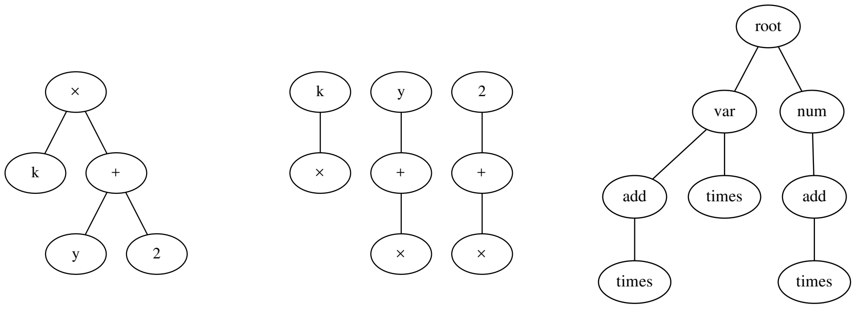
Mathematical content is widely used in technical documents such as the publications and course materials from STEM fields. To better utilize such a valuable digitalized mathematical asset, it is important to offer search users the ability to find similar mathematical expressions. For example, some students may want to collect additional information about the formula that they have learned in the class, and others may want to find an existing proof for an equation. Unfortunately, major search engines do not support the similarity-based search for mathematical content.

The goal of this paper is to present our efforts on developing *OPMES* (Operator-tree Pruning based Math Expression Search), a similarity-based search engine for mathematical content. Given a query written as a mathematical expression, the system will return a ranked list of relevant math expressions from the underlying math collections.

Compared with existing mathematical search systems, such as MIaS [1], Tangent[2], and Zentralblatt math from Math Web Search [3] (MWS), the developed *OPMES* is unique in that operator trees [1] are leveraged in all the system components to enable efficient and effective search.

More specifically, *OPMES* parses an math expression into an operator tree, and then extracts leaf-root paths from the operator tree to represent structural

[1] `https://mir.fi.muni.cz/mias/`
[2] `http://saskatoon.cs.rit.edu/tangent/random`
[3] `http://search.mathweb.org`



(1) Operator tree representation of math expression $k(y+2)$.

(2) Leaf-root paths generated from the operator tree.

(3) Tree index (after tokenizing)

Fig. 1. Operator tree index example

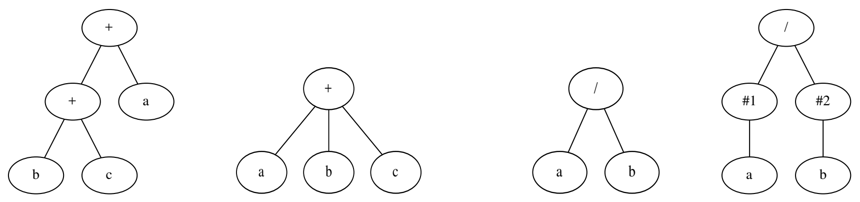
information. An example of a math expression operator tree representation and its extracted leaf-root paths is illustrated in (1) and (2) of Figure 1. The intuition is that no matter how operands are ordered, an operator tree uniquely determines the leaf-root paths decomposed from the tree. This property implies the advantage to use leaf-root paths from operator tree as indices or keys to retrieve math expressions (as previously proposed by [2–4]) in which a large portion of commutative operators is present. Built on top of this idea, *OPMES* further constructs a global tree index from all indexed math expressions, by continuously inserting tokenized leaf-root paths into this persistent tree index, as shown by figure 1 (3). Using this index, *OPMES* is able to search for structurally relevant expressions efficiently through a pruning method. Then, *OPMES* evaluates symbolic similarity between query and document expressions to finally rank search results based on symbol set similarity. For example, $E = mc^2$ is considered more meaningful when exact symbols are used rather than just being structurally identical with $y = ax^2$. We also need to rank documents higher if they are α-*equivalent* to query, since changes of symbols in an expression preserve more syntactic similarity when these changes are made by substitution, e.g. for query $x(1+x)$, expression $a(1+a)$ are considered more relevant than $a(1+b)$.

The demo page of the *OPMS* is avaialble at `http://tkhost.github.io/opmes`, and the source code can be downloaded at `https://github.com/t-k-/OPMES`. Students or scholars who have the need to search mathematical Q&A website or math-content articles are potential users of our search engine.

2 System Description

We now provide the details for three major components in our system: parser, indexer and searcher.

OPMES parser is responsible to extract math mode LaTeX markups from HTML files. A LALR (look-ahead LR) parser implemented by Bison/Flex is



(1) Commutative operator tree transformation for $a + b + c$.

(2) Non-commutative operator tree transformation for $\frac{a}{b}$.

Fig. 2. Operator tree transformation example

used to transform math mode LaTeX markup into in-memory operator tree from bottom up. In the case of constructing an operator tree with a commutative nodes, if a commutative node has a father operator who is also commutative, the node will pass its children to its father and delete itself (see (1) of figure 2) so that we make sure any sub-expression also represents a subtree in operator tree. On the other hand, when an operator tree of non-commutative node is constructed, it will insert different pseudo nodes (in our implementation we use the rank of the corresponding child) on top of its children (see (2) of figure 2) to distinguish their operands order in generated leaf-root paths.

The indexer then uses this operator tree to extract leaf-root paths. Note that a subtree of an operator tree (corresponding to a subexpression) would have its leaf-root paths being prefix of some leaf-root paths from this operator tree. To show this, use the same example in figure 1, in which $k(y + 2)$ has a leaf-root path set $A = \{k\times, y + \times, 2 + \times\}$, while the subexpression $y + 2$ has a leaf-root path set $B = \{y+, 2+\}$ where each element is prefix of A's subset $\{y + \times, 2 + \times\}$ respectively. OPMES indexer therefore tokenizes and inserts leaf-root paths into a "prefix tree" index as shown in figure 1 (3), to speed the lookup for another similar leaf-root path. The indexer also attaches (appends) an expression ID (or exprID) to the bottom node (of the newly inserted leaf-root path) every time a tokenized leaf-root path is inserted. In the case of figure 1 (3), all the three $\boxed{\text{times}}$ nodes are linked to a separate "posting list", the new exprID is appended after each. For implementation simplicity, we use file system directories to realize the index tree, the path name of each directory corresponds to a tokenized leaf-root path, and the posting list file of each node is stored at corresponding directory which represents the node it belongs to (every node has only one posting file).

OPMES searcher takes a query, parses it into operator tree, and decomposes the operator tree into leaf-root paths. Query processing step is divided into two stages. The first stage is to search for structurally relevant expressions. Instead of searching each leaf-root path one by one, the searcher searches simultaneously along the way of all leaf-root paths in the index tree, and merges the exprIDs from posting lists in all the corresponding directories of the query paths. Moreover, if all query leaf-root paths can go one deeper level in the index tree, and the deeper level nodes have a common node (with the same tokenized name), the searcher will simultaneously go into the common node from all query paths and merge the posting lists under that common directory. This process is repeated recursively to prune indexes (directories) that are not common at the deeper level. The second step is to rank all the structurally relevant expressions identified in the first step based on their symbolic similarities with the query. The scoring algorithm Mark and Cross, which addresses both symbol set equivalence and α-equivalence, is fully explained in the first author's master thesis [5].

3 Demonstration Plan

In our demo, we first illustrate some key ideas mentioned above. We will choose a simple query, show its operator tree representation in ASCII graph as well as its leaf-root paths (through the output of parser). Then we demonstrate the structure of our index tree, and walk through the steps and directories where the searcher goes and finds relevant expressions for input query. Users are invited to enter queries and experience our search engine based on a collection (with over 8 million math expressions scrawled from Math Stack Exchange website) that contains most frequently used and elementary math expressions.

4 Conclusion and Future Work

The paper describes a new similarity-based mathematical search engine, i.e., *OPMES*. Operator trees are used in almost all the system components to facilitate the representation, query processing and relevance modeling of the mathematical content.

As for the future work, we plan to enhance the system with MathML parser and wildcard support. Moreover, we also plan to integrate text search ability into our math-only search method. Finally, equivalent math-expression transformations (such as $a + \frac{1}{b} = \frac{ab+1}{b}$) can also be introduced into pre-query process to further improve the usefulness of math similarity search engine.

References

1. Richard Zanibbi and Dorothea Blostein. Recognition and retrieval of mathematical expressions. *International Journal on Document Analysis and Recognition (IJDAR)*, 15(4):331–357, 2012.
2. Hiroshi Ichikawa, Taiichi Hashimoto, Takenobu Tokunaga, and Hozumi Tanaka. New methods of retrieve sentences based on syntactic similarity. *IPSJ SIG Technical Reports*, pages 39–46, 2005.
3. Yoshinori Hijikata, Hideki Hashimoto, and Shogo Nishida. An investigation of index formats for the search of mathml objects. In *Web Intelligence/IAT Workshops*, pages 244–248. IEEE, 2007.
4. Yokoi Keisuke and Aizawa Akiko. An approach to similarity search for mathematical expressions using mathml. *Towards a Digital Mathematics Library. Grand Bend, Ontario, Canada*, pages 27–35, 2009.
5. Wei Zhong. A Novel Similarity-Search Method for Mathematical Content in LaTeX Markup and Its Implementation. `http://tkhost.github.io/opmes/thesis-ref.pdf`, 2015.