Partial-match Retrieval with Structure-reflected Indices at the NTCIR-10 Math Task

Hiroya Hagino
Keio University
hagino@nak.ics.keio.ac.jp

Hiroaki Saito
Keio University
hxs@ics.keio.ac.jp

ABSTRACT

To attain fast and accurate response in math formulae search, an index should be prepared which holds structure information of math expressions; a different indexing for full text search. Although some previous research has been done by this approach, the size of indices tends to become huge on memory. This paper proposes a partial match retrieval system for math formulae with two kinds of indices. The first one is an inverted index constructed from paths to the root node from each node seeing formula as an expression tree. The other index is a table which stores the parent node and the text string for each node in the expression trees.

A hundred thousand documents in the NTCIR-10 Math Task (formula search) containing 36 million math formulae were used for evaluation. The number of nodes was about 291 million and the number of path kinds in the inverted index was about 9 million. Experimental results showed that the search time grows linearly to the number of retrieved documents. Concretely, the search time ranges from 10 milliseconds to 1.2 seconds; the simpler formulae tend to need more search time.

Team Name

NAK

Subtasks

Math IR

Keywords

math formulae search, inverted index

1. INTRODUCTION

We built a math formulae search system, in which the query is written in MathML and the target documents are described in the XMTML format including HTML and MathML, along the lines of the formula search scenario in the NTCIR-10 math retrieval task. Our system consists of two components; the index builder and the query handler. The former builds indices from documents in advance and the latter receives a query and searches for appropriate documents using the built indices.

Section 2 explains the data structure our index builder adopts. Section 3 shows how the query handler performs the task.

2. DATA STRUCTURE OF INDICES

We build two indices; an inverted index and a node information table.

2.1 Inverted index for paths

We transform a DOM tree of Content-Markup to an expression tree and build an inverted index in which an index word represents the path from all the nodes to the root. The root, which exists only one, is a child node of node `<math>`.

2.1.1 Transformation from DOM tree to expression tree

The following shows how the transformation to an expression tree is performed.

1. If `<csymbol>` node exists as in Figure 1, transform to a single node whose name is concatenation of string '`symbol_`' and the text of `<csymbol>` node.

```
<csymbol>subscript</csymbol>
                ↓
      <symbol_subscript />
```

Figure 1: Substitution of `<csymbol>`

2. If `<apply>` node exists as in Figure 2, substitute the `<apply>` node for its left-most child node and delete that node.

```
<apply>
  <plus />
  <ci>x</ci>
  <ci>y</ci>
</apply>

      ↓

<plus>
  <ci>x</ci>
  <ci>y</ci>
</plus>
```

Figure 2: Substitution of `<apply>`

2.1.2 Inverted index

An inverted index is built, where the index word is the path from each node in the expression tree, which is created by the method described in Section 2.1.1, to the root node, and its value is the node list.

As an example, let's build an inverted index for $\sqrt{x^2+y^2}$.

1. Attach the node number as in Figure 3. Each node number must be unique in the entire documents.

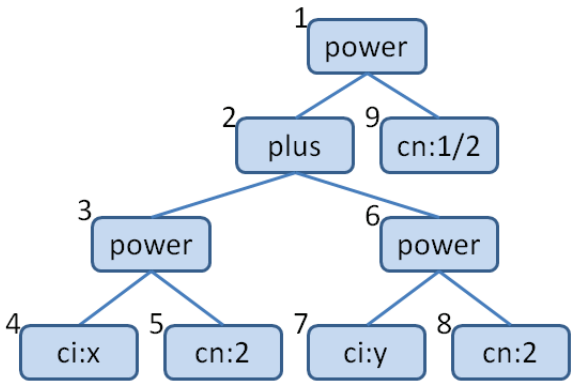

Figure 3: Expression tree of $\sqrt{x^2+y^2}$

2. Build an inverted index for all the paths from each node to the root. Each node is encoded to a 1-byte code in order to reduce the index size. When the encoding rules are set as in Table 1, for instance, an inverted index like Table 2 is built, where 'index word (after encoded)' and 'node list' are paired. The inverted index is sorted alphabetically by the encoded index words.

Table 1: Tag code table

code	node
0x01	plus
0x02	times
0x03	power
0x04	ci
0x05	cn
0x06	eq

Table 2: Inverted index for paths

Index word (pre-encoded)	Index word (after encoded)	node list
plus power	0x01 0x03	2
power	0x03	1
power plus power	0x03 0x01 0x03	3,6
ci power plus power	0x04 0x03 0x01 0x03	4,7
cn power	0x05 0x03	9
cn power plus power	0x05 0x03 0x01 0x03	5,8

2.2 Node information table

The node information table stores the number of the parent node and the node text of each node, where the node text specifies the text of `<ci>` nodes and `<cn>` nodes. The node text is encoded in integers. If the encoding rules are set as in Table 3, the node information table for the expression tree in Figure 3 looks like Table 4.

Table 3: Code table for node texts

Code	Text
1	"x"
2	"y"
3	"1"
4	"2"
5	"1/2"
6	"z"

Table 4: Node information table

node#	parent-node#	pre-encoded text	encoded text
1	0		
2	1		
3	2		
4	3	"x"	1
5	3	"2"	4
6	2		
7	6	"y"	2
8	6	"2"	4
9	1	"1/2"	5

3. SEARCH ALGORITHM

Search proceeds in the following way.

1. As preprocessing, the system transforms a query to its expression tree and extracts its query path and branch information.

2. Search for the query path in the prefix-search fashion.

3. The system checks the variable names and tree structure for each suffix and judges on its validity.

3.1 Preprocessing of queries

To handle queries efficiently, a query is transformed to its expression tree in the same way as in Section 2.1.1. After that, the path from the leaf to the root and how the branch occurs are extracted. We call the path from the leaf to the root a 'query path.' The number of query paths equals to the number of leaf nodes in the expression tree.

3.1.1 Transformation from query to expression tree

`<csymbol>` nodes and `<apply>` nodes are transformed in the same way as in Section 2.1.1.

Table 5: Query information of $x^n + y^n = z^n$

node number	query path (encoded)	node text (encoded)	number of steps for merge	merged node
1	0x04 0x03 0x01 0x06	1		
2	0x04 0x03 0x01 0x06	−1	1	1
3	0x04 0x03 0x01 0x06	2	2	1
4	0x04 0x03 0x01 0x06	−1	1	3
5	0x04 0x03 0x06	6	2	1
6	0x04 0x03 0x06	−1	1	5

3.1.2 *Extraction of paths and branches*

All the leaf nodes are enumerated from left to right and the query paths are extracted from each. The query paths from only leaf nodes suffice here because the child node contains the information of its parent node, while all the nodes were targeted to the root in Section 2.1.1 for search efficiency.

Unlike regular MathML, queries to our system might contain `<qvar>` nodes which represent query variables, but `<qvar>` nodes denote variables when searching, thus they are treated as `<ci>` nodes, which represent variables in MathML. The node text is encoded to the same code when the `<qvar>` text is identical. We used negative integers for this purpose. Node texts for the other nodes are encoded according to the code table in the same way as in Section 2.2.

We also focus on the node which merges its left node at the first time. We record how far the node is from the leaf node as step number. We also record the leaf node if we traverse from the focused node to the leaf node via left-most nodes. We can ignore the query path from the left-most leaf, because such path does not merge into brother nodes.

Query $x^n + y^n = z^n$ is tranformed to the expression tree in Figure 4, for instance, and query path, node text, and branch information are collected as in Table 5, if the tag code in Figure 1 and the node text coding in Table 3 are used.

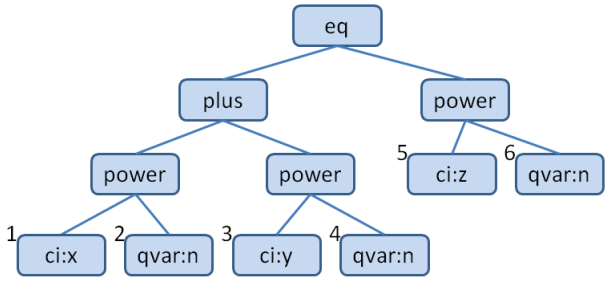

Figure 4: Expression tree of query $x^n + y^n = z^n$

3.2 Search

Using the inverted index, search is performed against all the query paths in the prefix-search fashion. If any match can be found, the query expression tree can be included in the documents as a partial tree. Because the inverted index is sorted alphabetically, binary search can be used.

Here we focus on the index word which hits in searching. If we call the unmatched part 'suffix', search result which has different suffix does not match as a tree, because they have a different root node. That means that we can avoid identical judgement between nodes with different suffixes. Thus we can accelerate search by looking at the node lists of the index words whose path have the same suffix.

3.3 Structure identification of math expressions

The following identification judgement is performed by backtrack against the math formulae with a suffix concatenated. When all the query paths are matched, it is regarded as a successful match and the document ID and the formulae ID are added to the output list.

- The system judges the variable name correspondence by comparing the node text of a query with the node text in the node information table.
- The system judges the tree structure identification by comparing the query branch information and the parent node numbers in the node information table.

4. RESULTS

4.1 Index building

Table 6 shows our indices built for the NTCIR Math Task.

Table 6: Our indices built for Math Task

# of documents	100,000
# of math formulae	36,627,568
# of nodes	291,823,116
# of tags	241
# of paths in inverted index	8,947,979
# of node text	510,285
size of indices	17.6GB
time for building indices	2 hours 11 minutes

4.2 Search time

Search time grew almost linearly to the number of retrieved documents. Concretely, the search time ranges from 10 milliseconds to 1.2 seconds; the simpler formulae tend to need more search time.

4.3 Accuracy

Table 7 shows our result for each task.

Table 7: Our result

Query	Hits	Relevant		Partially Relevant	
		Nums	Precision	Nums	Precision
1	0	0	N/A	0	N/A
2	15	0	0.000	1	0.067
3	12	9	0.750	12	1.000
4	35	7	0.200	25	0.714
5	34	21	0.618	30	0.882
6	0	0	N/A	0	N/A
7	37	5	0.135	15	0.405
8	592	26	0.044	29	0.049
9	1	0	0.000	1	1.000
10	0	0	N/A	0	N/A
11	0	0	N/A	0	N/A
12	0	0	N/A	0	N/A
13	0	0	N/A	0	N/A
14	0	0	N/A	0	N/A
15	0	0	N/A	0	N/A
16	0	0	N/A	0	N/A
18	401	35	0.087	43	0.107
19	0	0	N/A	0	N/A
20	0	0	N/A	0	N/A
21	0	0	N/A	0	N/A
22	0	0	N/A	0	N/A
Total	1127	103	0.091	156	0.138

5. CONCLUSION AND FUTURE WORKS

We proposed how to build new indices and how to use them in order to search for math expressions against documents including math formulae written in MathML. Although we take into consideration the structure of math expressions, the result was not impressive with a low precision around 10 - 14 %. We need to investigate why false positives or false negatives occured.

6. REFERENCES

[1] Michael Kohlhase and Ioan A. Sucan. A search engine for mathematical formulae. In *Proceedings of Artificial Intelligence and Symbolic Computation, Vol. 4120*, pages 241–253, 2006.

[2] Michael Kohlhase, Bogdan A. Matican, and Corneliu-Claudiu Prodescu. Mathwebsearch 0.5: Scaling an open formula search engine. In *Proceedings of Artificial Intelligence and Symbolic Computation, Vol. 7362*, pages 342–357, 2012.

[3] 橋本英樹，土方嘉徳，西田正吾. MathML を対象とした数式検索のためのインデックスに関する調査. **情報処理学会研究報告，データベース・システム研究会報告**, *Vol. 2007, No. 54*, pages 55–59, 2007. (In Japanese).

ACKNOWLEDGMENTS

We would like to thank the organizers at NTCIR for the resources of the Math IR task.