The MCAT Math Retrieval System for NTCIR-11 Math Track

Giovanni Yoko Kristianto
The University of Tokyo
Tokyo, Japan
giovanni@nii.ac.jp

Goran Topić
National Institute of Informatics
Tokyo, Japan
goran_topic@nii.ac.jp

Florence Ho
Institut National Polytechnique Toulouse
Toulouse, France
florence.ho@etu.enseeiht.fr

Akiko Aizawa
National Institute of Informatics
The University of Tokyo
Tokyo, Japan
aizawa@nii.ac.jp

ABSTRACT

This paper describes the participation of our MCAT search system in the NTCIR-11 Math-2 Task. The purpose of this task is to search mathematical expressions using hybrid queries containing both formulae and keywords. We introduce an encoding technique to capture the structure and content of the mathematical expressions. Each expression is accompanied by two types of automatically extracted textual information, namely words in context window and descriptions. In addition, we examine the improvement in ranking obtained by utilizing dependency graph of mathematical expressions and post-retrieval reranking method. The results show that the use of description and dependency graph together delivers better ranking performances than the use of context window. Furthermore, using both the description and context window together delivers even better results. The evaluation results also indicate that our reranking method is effective for improving the ranking performances.

Keywords

Mathematical Formula Search, MathML Indexing, Context, Description, Dependency Graph, Reranking

Team Name

MCAT

1. INTRODUCTION

The NTCIR-11 Math-2 Task aims to explore search methods for retrieving mathematical formulae in scientific documents. Given a query which contains a target formula expressed in MathML and several related keywords, each participating system is expected to return a ranked list of the relevant retrieval units containing a formula matching the query.

Our system in the previous NTCIR Math Pilot Task enabled searching for mathematical expressions using queries that contain both mathematical formulae and keywords. To enable full-text search, this system associated each indexed mathematical expression with words found in the fixed context window. However, this type of textual information is not guaranteed to be capable of representing the mathematical expressions precisely. In the NTCIR-11 Math-2 Task, we associate each mathematical expression with descriptions, which is automatically extracted using a machine learning method, in addition of words found in context window. However, from the description extraction results, we found that several mathematical expressions are not associated with descriptions. We then introduce a method based on dependency graph of mathematical expressions to compensate this lack-of-descriptions issue. Furthermore, we apply a post-retrieval reranking method and investigate its impact over the ranking performances.

This rest of this paper is organized as follows. Section 2 provides a brief overview of the background and related work on math retrieval systems. Section 3 explains the indexing procedure, textual information, dependency graph of mathematical expressions, and reranking method. Next, Section 4 presents the results of our submitted results. Finally, our conclusions are given in Section 5.

2. RELATED WORK

There are six mathematical search systems that are published in NTCIR-10 Math Pilot Task [1]. The UC Berkeley team [12] combined a standard keyword content information retrieval method with bitmap indexing of math operators identified as separate XML tags in the MathML structure. MathWebSearch search engine [10, 8, 9] was built around the substitution-tree indexing technique [5]. The Masaryk University MIRMU team [14, 13, 20, 19] focused on the similarity search based on enhanced full text search. The NAK team [6] proposed how to build new indices and how to use them in order to search for mathematical expressions. The FSE team [18] proposed to employ a distributed data processing system, i.e. Stratosphere [3], that accesses data in a non-index format. The MCAT team [21] adopted Apache Solr [2] and introduced three fields dedicated to encoding the structure and content of a mathematical expression. There were also additional fields to hold any natural language text associated with a given expression. The result from this Math Pilot Task showed that MCAT system [21] was the top performer in both formula and fulltext search among the competing systems. In addition, FSE and MCAT teams enabled full-text search feature in their retrieval methods.

3. OUR APPROACH

3.1 Indexing Mathematical Expressions

The Problem

MathML defines two different layers: Presentation MathML and Content MathML. The former expresses the layout of symbols used to display a mathematical formula, while the latter encodes its semantics, with no regard to notation. The reason for this split is the fact that mathematical notation is quite inconsistent, and symbol set limited: a notation is commonly reused, and there often exist several different ways of writing down the same core meaning. For example, the derivation of function $y = f(x)$ can be represented as $\frac{d}{dx}f(x)$, $\frac{df(x)}{dx}$, $\frac{d}{dx}y$, $\frac{dy}{dx}$, $f'(x)$, $D_x y$, $\dot{y}$ and more. Conversely, the notation $y(x+1)$ could represent both an application of function y, as well as a multiplication of the value y to $x+1$; and the meaning of i in the most common interpretation of a_i is very different from that in $3i+5$. Speaking of the imaginary constant, in fact, there are at least three ways in common use to represent it: some journals and writers use the italic i, some the regular i, while Unicode reserves for it the double-struck i at code point 0x214F (`&ImaginaryI;` entity in HTML). Thus, understanding a mathematical equation necessarily involves context: if it is clear from preceding text or formulae that y is a function, it is safe to conclude that $y(x+1)$ refers to application of this function to $x+1$, and not multiplication.

This semantic ambiguity of the Presentation MathML is a large problem for a mathematics search engine. Ideally, one would translate all queries into the content mark-up: we assume the users will care more about the meaning of their query rather than a specific notational form. However, currently there is no method capable of accurately disambiguating the presentation form. While there is ongoing research into this very issue, it is still not at the stage where it could reliably handle any but the simplest mathematical expressions. This ambiguity problem is especially acute when looking at a search query, which will not have the benefit of context.

The inability to reliably extract the meaning of an expression affected our method significantly. The first realization was that we will have to create an index based on the Presentation MathML, which can be obtained, rather than on much more appropriate Content MathML, which can't be. The second realization was that searching for an exact match is unfeasible. If a query looks for $\sum_{i=0}^{n} a_i$, exact matching will not find $\sum_{i=0}^{n} a_i$ — even though both are readily understood to be the same thing by a human reader. Similarly, a query for the relationship between velocity and acceleration $v = at$ would not find $v = gt$, a special case where acceleration of gravity is written as g instead of the usual a. Ideally, $v = gt$ would still be found, but would be ranked lower than the perfect match, $v = at$.

Therefore, the method needed to be flexible in order to account for the notational differences, in some ways similar to full-text search, yet still encode the structural information necessary to distinguish $x\frac{y}{z}$ from $\frac{x}{y}z$. We also wanted to be able to search by context, a very important requirement in the current task, which would require a full-text search engine. The challenge then is how to incorporate mathematical search with full-text search.

Our Method

The full-text search requirement was solved easily, by adopting Apache Solr [2], the most popular open-source full-text search engine. Starting from this choice, we used the method described below to index and search the mathematical formulae, and thus provide a flexible, soft-matching, unified interface for different query types (both formula search and full-text search).

In our Solr schema, the multi-valued `description_XX` fields hold any natural language text associated with a given expression, where `XX` is a language identifier. There are three fields dedicated to encoding the structure and content of a mathematical expression — `opaths`, `upaths` and `sisters` — which are described below in detail. Finally, there are also various other fields serving as keys: the primary key field `gmid`, the foreign key `gpid` identifying the document where the expression is found, and the key `gumid` which identifies different appearances of the same expression within a single paper as a single class.

Each expression, both at index time and at query time, is transformed into a sequence of keywords across several fields. First, the XML expression tree is cleaned up a little, in an attempt at basic normalization: it is run through SnuggleTeX [15] up-conversion semantic enrichment module, and then unnecessary `mrow` and `mfenced` nodes are removed (since they provide no semantic information other than hierarchy). Next, vertical paths are gathered into the `opaths` (ordered paths) field in such a way that ordering is preserved. In some cases (such as looking for $b+c$ and trying to match $a+b+c$), ordered paths will not be effective, so we introduce the `upaths` (unordered paths) field, with exactly the same information as in `opaths` but with ordering information removed. This vertical path encoding is performed not only for the expression's tree, but for each of its subtrees as well, to achieve a hit on $a(b+c)$ for the query $b+c$. The third and final field carrying the expression structure, `sisters`, lists the sister nodes in each subtree.

```
<math>
    <mrow>
        <msubsup>
            <mo>&Sigma;</mo>
            <mrow>
                <mi>i</mi>
                <mo>=</mo>
                <mn>0</mn>
            </mrow>
            <mi>n</mi>
        </msubsup>
    </mrow>
    <msub>
        <mi>a</mi>
        <mi>i</mi>
    </msub>
</math>
```

Figure 1: MathML example: $\sum_{i=0}^{n} a_i$

Figure 1 presents a simple mathematical formula, $\sum_{i=0}^{n} a_i$, written in MathML. Figure 2 shows an example of its representation our index. The first `opaths` encodes the whole MathML tree. The root of the whole tree, an `<mrow>`, is not shown — as explained above, `mrow` has no semantic

opaths:
`1#msubsup 1#1#mo#Σ 1#2#1#mi#i 1#2#2#mo#= 1#2#3#mn#1 1#3#mi#n 2#msub 2#1#mi#a 2#2#mi#i`

opaths:
`msubsup 1#mo#Σ 2#1#mi#i 2#2#mo#= 2#3#mn#1 3#mi#n`

opaths: `1#mi#i 2#mo#= 3#mn#1`

opaths: `msub 1#mi#a 2#mi#i`

upaths:
`#msubsup ##mo#Σ ###mi#i ###mo#= ###mn#1 ##mi#n #msub ##mi#a ##mi#i`

upaths: `msubsup #mo#Σ ##mi#i ##mo#= ##mn#1 #mi#n`

upaths: `#mi#i #mo#= #mn#1`

upaths: `msub #mi#a #mi#i`

sisters: `mi#i mo#= mn#1`

sisters: `mo#Σ mi#n`

sisters: `mi#a mi#i`

sisters: `msubsup msub`

Figure 2: Encoding example: $\sum_{i=0}^{n} a_i$

value, and so do not explicitly list it in the index. The root has two children: `<msubsup>` and `<msub>` elements, which is reflected in `opaths:1#msubsup 2#msub`: the first top-level child being `<msubsup>`, and the second `<msub>`. Grandchildren are given similarly, their position at each level being specified by consecutive numbers, divided by separators. If the element is one of the leaf elements (`<mo>`, `<mn>` or `<mi>`), it also has an additional separator and the representation of its value. Thus, `opaths:1#2#3#mn#1` says that the third child of the second child of the first top-level element is a number with a value of 1 (i.e. `<mn>1</mn>`).

The second row does the same for the sub-expression $\sum_{i=0}^{n}$; the third for the subscript $i = 0$; and the fourth for a_i. The sub-expressions n, i, 0 and a have no structure to encode, so they do not have their own `opaths`.

As said above, `upaths` is almost the same, but with ordering information removed. Accordingly, `upaths:###mn#1` just says that a child of a child of a top-level element is the number 1.

Because the ordering information is removed for all levels, not just the lowermost one, in a contrived example of $\frac{a+b}{c+d+e}$ and $\frac{c+a}{b+e+d}$, they will both match equally for the query $\frac{x}{e+c+y}$ (where the latter is a plausible match, with substitutions $x = a + b$, and $y = d$). This situation comes about because `opaths` are not applicable to any element (e is never first in its subtree, c is never second, and x and y do not match at all); while `upaths` won't differentiate between sub-elements of the numerator and the denumerator, and thus equally find both e and c. To solve this, we introduce the field `sisters`, whose role is to group sister elements together, and in this case boost the score of the first expression where both e and c are found in the same subtree, over the second expression where they aren't.

The `opaths`, `upaths`, and `sisters` fields, which capture the structure of each mathematical expression, are indexed as whitespace-tokenized fields, i.e. 'text_ws' in the default schema.xml[1]. On the other hand, descriptions are indexed as English text, i.e. 'text_en' in the default schema.xml[1].

Matching is then performed by a normal Solr disjunctive query (using default query parser). It considers the factors in the query as independent, then modifies the score using TF/IDF and length normalization. The result is a very flexible, heuristic system that ranks the search results by similarity to the query expression, and to a degree avoids the pitfalls of inconsistent representation. The downside is that it is *too* flexible: it is difficult to say where the relevant results stop and random matches begin; thus we predict higher recall, but lower precision rates than exact match systems.

3.2 Extracting Textual Information for Mathematical Expressions

This paper associates each mathematical expression with two types of textual information, namely words in context window and descriptions. To extract words in context window, we first tokenize sentences which contain any number of mathematical expressions. For each mathematical expression, we extract words in context window by taking ten words preceding and following the expression. On the other hand, descriptions are extracted automatically using a support vector machine (SVM) model that is trained using manually annotated corpus generate for NTCIR-10 Math Pilot Task [1]. The description extraction task is treated as a binary classification problem. We first pair each mathematical expression with each noun phrase that exists in the same sentence as the expression. Subsequently, for each pair, we extract several features based on sentence patterns, POS tags, parse trees, and predicate-argument structures. Next, using these features, we classify the pairs as correct (i.e. the noun phrase is the description of the expression) or incorrect. Further details of our description extraction task was described in another publication [11]. For the NTCIR-11 Math task, we modified our previous extraction model to use predicate-argument relations generated by natural language parser Enju [16] instead of Stanford dependencies generated by Stanford parser [7, 4], because the use of Enju allows short parsing time compared to the use of Stanford parser.

To measure the performance of our description extraction method, we conduct a description extraction experiment on 50 mathematical papers obtained from the NTCIR Math Understanding Subtask [1]. We set 40 papers to be the training set and 10 papers to be the test set. In addition, this experiment uses three metrics to measure the extraction performance: precision, recall, and F1-score. Furthermore, the extraction performance is measured using two different matching scenarios, namely strict matching and soft matching. An extracted description will pass the strict matching evaluation if its position, i.e. start index and length, is the same as that of a gold-standard description for the same target mathematical expression. On the other hand, an extracted description will pass the soft matching evaluation if its position contains, is contained in, or overlaps with the position of a gold-standard description for the same expression. Table 1 displays the results of the performance measurement.

Table 2 shows an example of words in context window and description. Before indexing the extracted textual informa-

[1] `http://svn.apache.org/viewvc/lucene/dev/trunk/solr/example/solr/collection1/conf/schema.xml?view=co&content-type=text/plain`

Table 1: Performance of the description extraction model

Matching Scenarios	P	R	F1
Strict	73.72	45.88	41.40
Soft	80.80	72.77	76.58

tion, several processes are applied to it, such as tokenising the string, eliminating stop words and any mathematical expression, and stemming the remaining words.

Table 2: An example of textual information extracted for the target expression $p(x|\theta)$ in a sentence "For several applications, it is often computationally convenience to work with the natural logarithm of the likelihood function $p(x|\theta)$, called the log-likelihood."

Type	Example of Textual Information
context	to work with the natural logarithm of the likelihood function called the log-likelihood
description	likelihood function

3.3 Developing Dependency Graph of Mathematical Expressions

While words in context window are extracted simply by taking words preceding and following the expression, descriptions are extracted using a machine learning model. From the extraction results, we found that several mathematical expressions are not associated with descriptions. To compensate this lack-of-descriptions issue, we propose a method based on dependency graph of mathematical expressions.

A dependency graph $G(V, E)$ of mathematical expressions is a directed graph that is built by examining tokens of each expression in a document. Each vertex in the graph represents a distinctive mathematical expressions found in a document. If a distinctive mathematical expression appears several times in a document, all of the extracted words in context window and descriptions are merged into their respective sets, namely the set of context window and the set of descriptions. A directed edge from vertex *mathexp*_1 to vertex *mathexp*_2 indicates that the mathematical expressions *mathexp*_1 contains expression *mathexp*_2. In this case, the vertices *mathexp*_1 and *mathexp*_2 are called parent and child, respectively. Figure 3 shows an example of dependency graph.

Having obtained a dependency graph, we can utilize the textual descriptions from each child expression to help represent the target expression. As a result, we can associate each mathematical expression not only with its own descriptions, but also with the descriptions from each of its children. Therefore, for each mathematical expression, we can store these extended descriptions instead of its own descriptions in the `description_en` field.

3.4 Reranking Retrieved Mathematical Expressions

The reranking system processes the math formula query and the ranking list of the initially retrieved math formulae, and then modify the ranking accordingly with new scores. The general concept we chose to apply was to measure the similarity between the query formula and each retrieved formula belonging to the input set. By "similarity", we mean either syntactically identical to the query or close to the

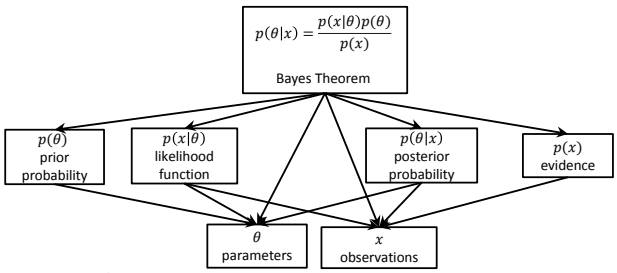

Figure 3: An example of dependency graph of seven mathematical expressions. Each mathematical expression is accompanied by its corresponding description.

query in terms of mathematical meaning, with the several equivalent notations in math formulae. Hence, we computed a similarity measure which is the combination of the two aspects of a math expression: the structure and the content. Our approach measures the content and the structure similarities between the query and each result separately, and then those two measures are combined to form the new score which is then a similarity score. We defined the similarity score between the query and a retrieved formula as follows:

$$formSim(f_x, f_q) = (contSim(f_x, f_q) \cdot \lambda) + (structSim(f_x, f_q) \cdot (1 - \lambda)) \quad (1)$$

where λ is the weighting parameter ranging from 0 to 1 applied to the content and the structure score, f_q is the query, and f_x is a retrieved formula to be compared to the query.

Both the content and the structure information are represented using the vector space model. A similarity vector is then generated by computing the similarity between each retrieved formula in the input dataset and the query using Equation 1. The value of the weight λ is determined to be 0.3. It is essential that this weight stays under 0.5 because we want the structure of a formula to be prominent over the content. Eventually, the formulae are ranked in ascending order of their similarity scores obtained.

(a) **Structure of a math formula.** The math expressions written in MathML have tree-based structure which is broken down into a collection of distinct paths. These paths are used to measure the structural distance between expressions written in MathML. The structure of a math formula, f_i, is modeled as a vector $(p_{i1}; p_{i2}; \cdots; p_{iN})$, where each element of the vector represents the frequency of a path that appears in the formula and N being the total number of different paths among all the results and the query included. We build a list of all the distinct paths encountered through the entire input dataset and the query. A structure similarity matrix of the MathML files is then generated. Eventually, the distance between the query and each retrieved formula is computed using Euclidean distance.

(b) **Content of a math formula.** With the use of the org.w3c.dom package [17], we extract the content of each MathML file and build a set of distinct terms. We distinguish three types of contents using the MathML markup and for which we considered to be of various importance: operators, numeric literal, and identifiers.

Hence, numeric literals and operators are subjects to bear a strong importance in the content to be considered, whereas identifiers which can be expressed in many ways would be weighted less. We included a few exceptions such as defined functions names like *sin* or *cos*, or symbolic constants such as π, which we placed as exceptions to be considered equal level as the numeric literal and operators. The content of a MathML file is modeled as a vector, where each element of the vector represents the frequency of a term which appears in the expression. Like in structure modeling, a content similarity matrix is generated with the different terms collected in all the files, including the query.

4. EXPERIMENTS

4.1 Experiment Setup

In the NTCIR-11 Math task, we submitted results from four runs, as follows.

1. **MCAT_nodep-context** is obtained by indexing both mathematical expressions and their associated context window.

2. **MCAT_dep-descriptions** is obtained by indexing both expressions and their associated descriptions. This run utilizes dependency graph approach to compensate for the any lack of extracted descriptions. Once a dependency graph of mathematical expressions is built for each document, we use the textual descriptions from each child expression to help represent the target expression.

3. **MCAT_all** is obtained by indexing expressions and both their associated context window (from MCAT_nodep-context) and descriptions (from MCAT_dep-descriptions).

4. **MCAT_dep-rerank** is obtained by applying post-retrieval reranking method to the result of MCAT_dep-descriptions run.

The ranking results from these runs are evaluated using three metrics, namely precision-at-5 (P@5), precision-at-10 (P@10), and mean average precision (MAP). The evaluation results allow us to investigate the effectiveness of descriptions compared to the context window in a mathematical search system. Furthermore, in one of the runs, we examine the impact of applying a post-retrieval reranking method, by weighting the original scores by the reranking scores. The found expressions are grouped by the containing paragraph. Scores for paragraphs are generated by adding together the scores of all expressions found in them.

4.2 Experimental Result

The results of the four submitted runs are shown in Table 3. This table shows that utilizing descriptions and dependency graph together (MCAT_dep-descriptions) delivers better results than utilizing context window (MCAT_nodep-context). When only highly relevant expressions are accepted, the MCAT_dep-descriptions run gives higher P@5, P@10, and MAP by 17.07%, 20.83%, and 25.83%, respectively, relative to the MCAT_nodep-context run. In the partial relevancy setting, the MCAT_dep-descriptions run yields a lower accuracy gain, that is by 2.97%, 8.33%, and 10.82%. The experimental results also indicate that utilizing both types of textual information (MCAT_all) gives even better results. This MCAT_all run outperforms not only the MCAT_nodep-context, but MCAT_dep-descriptions as well. Relative to the performance of MCAT_nodep-context, MCAT_all delivers higher P@5, P@10, and MAP by 29.27%, 29.17% and 39.42%, respectively, in the high relevancy setting and by 10.89%, 16.67% and 19.33% in the partial relevancy setting. In addition, the performance improvements obtained by MCAT_dep-descriptions and MCAT_all are also statistically significant, except for the P@5 improvement gained by MCAT_dep-descriptions run in partial relevancy setting.

Table 3: Summary of the results in the four submitted runs. The values within parentheses represent the p-value of the corresponding metrics. The p-values from MCAT_dep-descriptions and MCAT_all runs are relative to the MCAT_nodep-context run; the p-value from MCAT_dep-rerank run is relative to the MCAT_dep-descriptions run.

Run Name	P@5	P@10	MAP
High Relevancy			
MCAT_nodep-context	.1640	.0960	.0515
MCAT_dep-descriptions	.1920 (3.2×10^{-2})	.1160 (4.1×10^{-3})	.0648 (1.4×10^{-2})
MCAT_all	.2120 (1.8×10^{-4})	.1240 (5.6×10^{-4})	.0718 (1.1×10^{-3})
MCAT_dep-rerank	.2080 (1.4×10^{-1})	.1300 (3.2×10^{-2})	.0742 (6.5×10^{-3})
Partial Relevancy			
MCAT_nodep-context	.4040	.2400	.0776
MCAT_dep-descriptions	.4160 (2.3×10^{-1})	.2600 (2.2×10^{-2})	.0860 (4.6×10^{-2})
MCAT_all	.4480 (5.8×10^{-3})	.2800 (1.5×10^{-4})	.0926 (3.0×10^{-3})
MCAT_dep-rerank	.4640 (1.7×10^{-2})	.2800 (4.5×10^{-2})	.0933 (4.7×10^{-2})

In addition, we also gained some improvement from using a reranking method. When comparing the results with the use of reranking (MCAT_dep-rerank) and the use of descriptions and dependency graph without reranking (MCAT_dep-descriptions), we observe that when using the reranking method, the P@5, P@10, and MAP values are superior by 8.3%, 12.1% and 14.5% respectively in high relevancy. Improvement can also be observed for partial relevancy, with 11.5% , 7.7% and 8.5% of increase in the scores. Overall, the use of the reranking method maximized the scores obtained, with an increase of 26.8%, 35.4% and 44.1% in the P@5, P@10, and MAP scores respectively, in comparison to the MCAT_nodep-context run.

For an example where a dependency graph improved the search result, let us consider the query MATH-2, which consists of a query formula $x^2 - x - 1 = 0$ and two keywords: "golden ratio" and "Fibonacci". From the relevancy assessment, we found that two mathematical expressions, i.e. $\alpha^k = \alpha^{k-1} + \alpha^{k-2} + \alpha^{k-3}$ and $u_{ij} = \sum_{k=1}^{j} -1^{i-k}\binom{i-1}{k-1}\binom{n-i}{j-k}a^{2k-i-1}$, are relevant to the query MATH-2. Thus, given the query MATH-2, we would want these two expressions ranked high. However, these expressions, especially $u_{ij} = \sum_{k=1}^{j} -1^{i-k}\binom{i-1}{k-1}\binom{n-i}{j-k}a^{2k-i-1}$, do not resemble the query formula much. Furthermore, there

is no descriptions associated with them; thus there is no match between terms from descriptions and from query text. Therefore, it is difficult for these two expressions to have high rankings.

The use of dependency graphs enables these two expressions to have higher rankings than when no dependency graph is used. The dependency graphs for these mathematical expressions are displayed in Figure 4. When we use dependency graphs, we can expand the textual description of expression $\alpha^k = \alpha^{k-1} + \alpha^{k-2} + \alpha^{k-3}$ to contain the descriptions of α, which are "the golden ratio" and "the roots of the quadratic $x^2 - x - 1 = 0$ characterizing Fibonacci sequences (is the golden ratio)$\cdots$". In addition, we can also expand the description of $u_{ij} = \sum_{k=1}^{j} -1^{i-k}\binom{i-1}{k-1}\binom{n-i}{j-k}a^{2k-i-1}$ to contain the descriptions of a, which are "the golden ratio" and "a root", and of n, which is "the upper parameter". As a result, several terms from the descriptions are a match for the terms from the query text. For $\alpha^k = \alpha^{k-1}+\alpha^{k-2}+\alpha^{k-3}$, the terms "golden", "ratio", and "Fibonacci" match with the query text. For $u_{ij} = \sum_{k=1}^{j} -1^{i-k}\binom{i-1}{k-1}\binom{n-i}{j-k}a^{2k-i-1}$, the terms "golden" and "ratio" match with the query text. Therefore, we can now expect these two expressions to have higher rankings than when no dependency graph is used. As a matter of fact, our submission MCAT_dep-descriptions reported that these two expressions are ranked 2nd and 5th, respectively.

4.3 Discussion

The experiment results show that the performance of our system seems weaker than at NTCIR-10. We suggest that this happens because the task specification and the selection of topics have changed significantly between NTCIR-10 and NTCIR-11. In NTCIR-10, we used the flexibility of our system to our advantage, that is we could find relevant sub-formulae where exact-match systems might not have. Moreover, allowing anything to match query variables did not penalize our system too much. However, in NTCIR-11, query variables get much bigger emphasis, most topics feature complete and very particular formulae, and sub-formulae matching is not nearly as useful as before.

Regarding the indexing time, our indexing process is done in several steps:

- Encoding formulae and formatting them and descriptions into XML fit for Solr import: around 40 processor-hours, which most of the time being spent in XML processing and I/O.
- Importing the data into Solr: around 8 hours.

In addition, the size of Solr core on our disk is 58Gb. On average, Tomcat that carries Solr is only at 300Mb virtual RAM use, and under 3Mb of real RAM. Futhermore, our process takes no significant RAM, since search is completely delegated to Solr (except for the transformation of the MathML DOM to our indexing schema, which uses negligible time and space for a single query). The query is an "or" query, which presumably requires Solr to use significant space and time; and the number of disjunctive terms increases directly with the number of nodes in the query tree.

Table 4 shows the total answering time from each submission and the average answering time per query. The results suggest that there is no significant differences of the average answering time per query among MCAT_nodep-context, MCAT_dep-descriptions, and MCAT_all subsmission. On the other hand, MCAT_dep-rerank submission requires much higher average answering time than the other three submissions. This happens due to the additional reranking process, which is applied after the regular retrieval process. For further details, Table 5 shows the average answering time of our system for each query.

Table 4: Answering time from each submission

Submission	Total Ans. Time (ms)	Avg Ans. Time per Query (ms)
MCAT_nodep-context	2,466,405	49,328
MCAT_dep-descriptions	2,420,947	48,419
MCAT_all	2,400,350	48,007
MCAT_dep-rerank	3,764,686	75,294

Table 5: Average answering time for each query over four submissions

Query	Ans. Time (ms)	Query	Ans. Time (ms)
MATH-1	46,228	MATH-26	92,751
MATH-2	50,102	MATH-27	75,350
MATH-3	31,150	MATH-28	11,545
MATH-4	33,119	MATH-29	74,737
MATH-5	135,237	MATH-30	11,820
MATH-6	53,463	MATH-31	44,126
MATH-7	11,414	MATH-32	7,847
MATH-8	30,851	MATH-33	41,204
MATH-9	74,629	MATH-34	51,734
MATH-10	61,109	MATH-35	36,862
MATH-11	71,594	MATH-36	83,757
MATH-12	15,589	MATH-37	21,945
MATH-13	19,620	MATH-38	32,854
MATH-14	37,804	MATH-39	11,902
MATH-15	18,785	MATH-40	49,461
MATH-16	32,330	MATH-41	49,999
MATH-17	44,152	MATH-42	32,078
MATH-18	14,520	MATH-43	47,382
MATH-19	42,116	MATH-44	36,570
MATH-20	35,914	MATH-45	33,314
MATH-21	82,508	MATH-46	295,836
MATH-22	18,004	MATH-47	95,970
MATH-23	9,514	MATH-48	173,235
MATH-24	41,364	MATH-49	160,937
MATH-25	119,890	MATH-50	58,879
Overall			55,262

5. CONCLUSION

We have presented the participation of our MCAT system in the NTCIR-11 Math-2 Task. We introduced an encoding technique for capturing the structure and content of the mathematical expressions and a modified TF/IDF score for ranking the retrieved expressions. In addition, to capture the meaning of each mathematical expression in natural language, we associated each expression with two types of textual information, namely words in context window and descriptions. Subsequently, dependency graph approach was

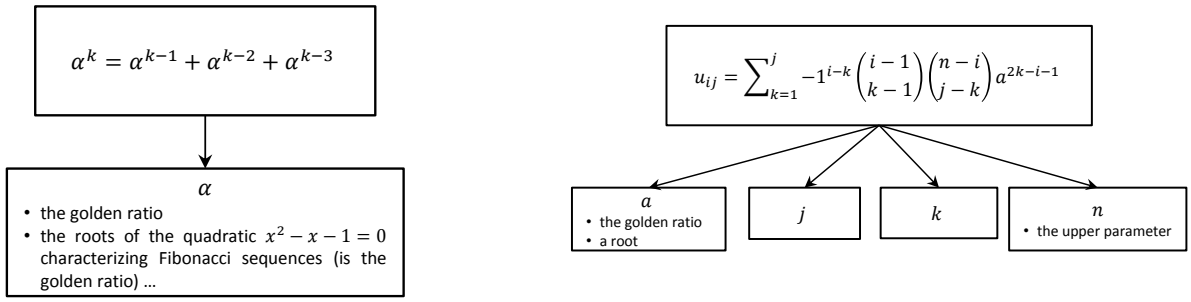

Figure 4: Two examples of dependency graphs that enrich the descriptions of target mathematical expressions. The left dependency graph enriches the descriptions of $\alpha^k = \alpha^{k-1} + \alpha^{k-2} + \alpha^{k-3}$. The right dependency graph enriches the descriptions of $u_{ij} = \sum_{k=1}^{j} -1^{i-k} \binom{i-1}{k-1} \binom{n-i}{j-k} a^{2k-i-1}$

utilized to tackle the encountered lack-of-descriptions issue. Moreover, we proposed a reranking method which is applied to the initially retrieved expressions. The experimental results showed that the use of descriptions and dependency graph together gave higher ranking performances than the use of context window. Furthermore, it is also shown that the use of descriptions and dependency graph together with the context window delivered even better results. The results also indicated that the reranking method effectively improved the ranking performances.

One of the future works is to set proper wights for context window and descriptions. We also consider to associate each mathematical expression with other types of textual information obtained in the same document as the expression, such as document title and keywords found in the abstract.

Acknowledgments

This work was supported by JSPS KAKENHI Grant Numbers 2430062, 25245084, and 14J09896.

6. REFERENCES

[1] A. Aizawa, M. Kohlhase, and I. Ounis. Ntcir-10 math pilot task overview. In *Proc. of the 10th NTCIR Conference*, 2013.

[2] Apache solr v4.1.

[3] D. Battre, S. Ewen, F. Hueske, O. Kao, V. Markl, and D. Warneke. Nephele/ pacts: A programming model and execution framework for web-scale analytical processing. In *Proc. of the 1st ACM Symposium on Cloud Computing*, pages 119–130, 2010.

[4] M.-C. de Marneffe, B. MacCartney, and C. Manning. Generating typed dependency parses from phrase structure parses. In *LREC*, 2006.

[5] P. Graf. *Term Indexing*. Number 1053 in LNCS. Springer Verlag, 1996.

[6] H. Hagino and H. Saito. Partial-match retrieval with structure-reflected indices at the ntcir-10 math task. In *Proc. of the 10th NTCIR Conference*, 2013.

[7] D. Klein and C. Manning. Accurate unlexicalized parsing. In *Proc. of the 41st Meeting of the Association for Computational Linguistics*, pages 423–430, 2003.

[8] M. Kohlhase, B. Matican, and C. Prodescu. Mathwebsearch 0.5 – scaling an open formula search engine. In *Intelligent Computer Mathematics. CICM*, pages 342–357, 2012.

[9] M. Kohlhase and C. Prodescru. Mathwebsearch at ntcir-10. In *Proc. of the 10th NTCIR Conference*, 2013.

[10] M. Kohlhase and I. A. Sucan. A search engine for mathematical formulae. In *Proc. of Artificial Intelligence and Symbolic Computation, number 4120 in LNAI*, pages 241–253. Springer, 2006.

[11] G. Kristianto, G. Topić, and A. Aizawa. Extracting textual descriptions of mathematical expressions in scientific papers. In *Proc. of the 3rd International Workshop on Mining Scientific Publications of the International Conference on Digital Libraries 2014*, London, England, 2014.

[12] R. Larson, C. Reynolds, and F. Gey. The abject failure of keyword ir for mathematics search. In *Proc. of the 10th NTCIR Conference*, 2013.

[13] M. Liška. *Evaluation of Mathematics Retrieval*. PhD thesis, Masaryk University, Brno, Faculty of Informatics, 2013.

[14] M. Liška, P. Sojka, and M. Ruzicka. Mirmu at the ntcir-10 math task: Similarity search for mathematics. In *Proc. of the 10th NTCIR Conference*, 2013.

[15] D. McKain. Snuggletex, 2011. http://www2.ph.ed.ac.uk/snuggletex/.

[16] Y. Miyao and J. Tsujii. Feature forest models for probabilistic hpsg parsing. *Computational Linguistics*, 34:35–80, 2008.

[17] org.w3c.dom (java platform se 7).

[18] M. Schubotz, M. Leich, and V. Markl. Querying large collections of mathematical publications. In *Proc. of the 10th NTCIR Conference*, 2013.

[19] P. Sojka. Exploiting semantic annotations in math information retrieval. In *Proc. of the 5th Workshop on Exploiting Semantic Annotations in Information Retrieval of The 21st ACM CIKM*, pages 15–16, 2012.

[20] P. Sojka and M. Liška. Indexing and searching mathematics in digital libraries - architecture, design and scalability issues. In *Proc. of Conferences on Intelligent Computer Mathematics 2011*, pages 228–243, 2011.

[21] G. Topic, G. Kristianto, M.-Q. Nghiem, and A. Aizawa. The mcat math retrieval system for ntcir-10 math track. In *Proc. of the 10th NTCIR Conference*, 2013.