MCAT Math Retrieval System for NTCIR-12 MathIR Task

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

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-12 MathIR Task. We introduce three granularity levels of textual information, new approach for generating dependency graph of math expressions, score normalization, cold-start weights, and unification. We find that these modules, except the cold-start weights, have a very good impact on the search performance of our system. The use of dependency graph significantly improves precision of our system, i.e., up to 24.52% and 104.20% relative improvements in the Main and Simto subtasks of the arXiv task, respectively. In addition, the implementation of unification delivers up to 2.90% and 57.14% precision improvements in the Main and Simto subtasks, respectively. Overall, our best submission achieves P@5 of 0.5448 in the Main subtask and 0.5500 in the Simto subtask. In the Wikipedia task, our system also performs well at the MathWikiFormula subtask. At the MathWiki subtask, however, due to a problem with handling queries formed as questions that contain many stop words, our system finishes second.

Keywords

Mathematical Formula Search, MathML Indexing, Path-based Encoding, Hash-based Encoding, Textual Information, Dependency Graph, Score Normalization, Cold-Start Weights, Score Boosting, Unification

Team Name

MCAT

Subtasks

MathIR arXiv Main Task (English), optional MathIR arXiv Formula Similarity Task (English), optional MathIR Wikipedia Task (English), optional MathIR Wikipedia Formula Browsing Task (English)

1. INTRODUCTION

The objective of NTCIR-12 MathIR Task [9] is to build MIR systems which enable users to search for a particular math concept using math formulae. Given a query which contains a target formula expressed in MathML and several related keywords, each participating system in this task is expected to return a ranked list of the relevant retrieval units containing formulae matching the query. Since this task has the same aim and similar searching scenarios as the previous NTCIR-11 Math-2 Task [1], our system [5] from the previous task is also applicable for the current task, but with several limitations as follows.

1. Our system employed only one index, where math expression is the retrieval unit. This did not work quite well, since the judgement process allowed paragraph, instead of only math expression, as a retrieval unit.

2. Our system stored multiple types (fields) of information for each math expression, but there was no score normalization applied for each information type. Therefore, we assumed that each type has the same importance.

3. Our technique to encode math expressions was too flexible. Given a math expression represented in the MathML format, this technique generated index terms based on paths from each element in the MathML to each of its descendants. As a consequence, two math expressions were considered matching, with a certain score, even when there was only a symbol appearing at the same location in the MathML tree representation of both expressions.

In the NTCIR-12 Math Task participation, we attempt to address the issues defined above. The contribution of this paper is as follows:

- Investigate the impact of indexing three levels of information (math, paragraph, and document) and combining them using weights obtained from multiple linear regression method to handle limitation 1 and 2.

- Investigate whether the effectiveness of text-enriching procedure using dependency graph, which was shown by our previous works [2, 5], are maintained when we have already included paragraph and document level textual information in our system.

- Investigate the effectiveness of unification as post-processing to handle limitation 3.

This rest of this paper is organized as follows. Section 2 describes the overview of our system, including brief explanation about encoding of math expression and extraction of textual explanation for each expression. Subsequently, it presents our approach in combining all the indexed information for the searching purpose. The description of unification method we applied as a post-processing tool ends this section. Next, Section 3 and 4 present the results of our submitted results. Finally, our conclusions are given in Section 5.

2. OUR APPROACH

Our search system is composed of three main building blocks as follows.

1. Pre-processing
   - encode math expressions
   - extract textual information
   - extract dependency graph of math expressions
2. Indexing math expressions
3. Searching and Ranking
   - combine all the indexed types of information
   - perform unification to the initial ranked list

Figure 1 shows the overview of our system. The retrieval unit returned by our system is in the math expression level. This output is suitable for Wikipedia task, but not for arXiv task, which expects that the output is in the paragraph level. In the arXiv task, to obtain a ranked list of paragraphs for each given query, we generate a score for each paragraph by taking the highest similarity score from all expressions found in them. This section focuses on describing the Searching and Ranking part of our system. We will only describe the pre-processing and indexing parts briefly, since the detailed explanation can be found in our previous works [2, 3, 5, 8].

2.1 Encoding Math Expressions

There are two techniques we employ to encode math expressions: path-based and hash-based techniques. Both techniques assume that math expressions are in the MathML format and treat each expression as a tree data structure.

Path-based Encoding Technique

Given a math expression, this encoding technique produces three types of information: `opaths` (ordered paths), which stores the vertical path of each node in the math expression and preserve the ordering information, `upaths` (unordered paths), which stores the same content as `opaths`, but without ordering information, and `sisters`, which stores the sibling nodes in each subtree. This encoding technique is used at both index-time and query-time. While `opaths` and `upaths` are generated for every subtree of the math expression being indexed, at query time, however, `opaths` and `upaths` always start from the root of the math expression. In addition, this encoding-technique handles each free variable `qvar` found in the queries by simply omitting any vertical paths, i.e. `opaths` and `upaths`, directing to it and excluding it from any `sisters`.

Hash-based Encoding Technique

In addition to the path-based technique, we apply hash-based technique as well to encode math expressions. There are three algorithms we use here, namely subtree hash, modular trick, and SIGURE hash, and their brief descriptions are as follows.

- *subtree hash* is designed to capture the implicit semantics represented by variable names. This algorithm is intuitively a depth-first pre-order traversal to construct hash codes, visiting each node oncee, and at each node having to combine (already computed) hash codes from children. The output from this algorithm is a feature set consisting of all the subtrees of the input tree.
- *modular trick* is designed to capture the semantics of structures or patterns in the formulae. For instance, the structure of $y = x^2$ indicates a quadratic equation, which in this example the left-hand side variable (y) is defined as the square of another variable (x). Based on this modular trick algorithm, both $y = x^2$ and $b = a^2$ have the same semantics. The output of this algorithm is a feature set consisting of all the substructures of a specified depth d rooted on any node of the tree.
- *SIGURE hash* is an algorithm to provide a metric which is invariant to variable names. In a math expression, alpha equivalent transformation corresponds to renaming of variables. For instance, this algorithm can capture the mathematician's intuition that $x = x$ has fundamentally a similar meaning to $y = y$, but not to $x = y$, for any value of x and y. The feature set obtained from this algorithm consists of subtrees of the input tree where all the variables are renamed according to their appearance order.

In the case of encountering leaves that are free variables `qvar`s, these algorithms generate hash values based on the `name` attribute found in the `qvar` element.

Detailed explanation and examples regarding these hash algorithms can be found in our previous paper [6].

2.2 Extracting Textual Information

Given a math expression, we extract textual information for each of the granularity levels as follows.

- Math-level
  - *words in context window* are obtained by taking ten words preceding and following each math expression,
  - *descriptions* are sets of terms that precisely denote the target math expression and are automatically extracted for each math expression using a support vector machine model,
  - *noun phrases* in the same sentence as the target math expression.
- Paragraph-level (for arXiv task only)
  - *all words* in the paragraph
- Document-level
  - *title* of the document,
  - *abstract* of the document,
  - *keywords* found in the document, which are extracted using RAKE [7],
  - *descriptions* from all math expressions in the document,
  - *noun phrases* in the document,
  - *all words* in the document.

Each extracted keyword is accompanied with a RAKE's score [7], which is in the interval of $[1.0, \infty)$. We later utilize the logarithm of this score, i.e. $1 + log(rakeScore)$, to boost the similarity score of each matching keyword.

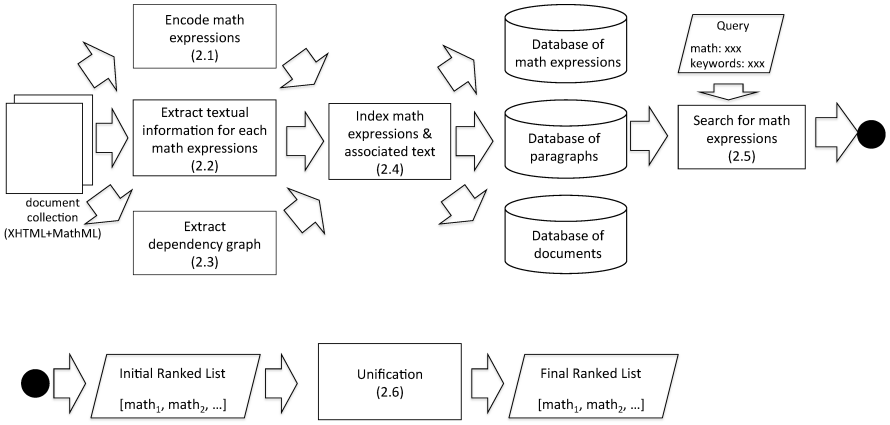

Figure 1: Overview of MCAT system

2.3 Extracting Dependency Graph of Math Expressions

In addition to the surrounding text (e.g. context window, descriptions), the meaning of each math expression can often be found in other regions of the document, especially if the expression appears several times within the document. Moreover, to understand a math expression, we often need to access the meaning of each constituent symbol. Having faced these challenges, we propose a method to capture the dependency relationships between math expressions within each document.

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 string representation of mathematical expressions *mathexp_1* contains expression *mathexp_2*. In this case, the vertices *mathexp_1* and *mathexp_2* are called parent and child, respectively.

In this MathIR task, prior to the string matching process, we apply several preprocessing steps:

- minimize the number of `mrow` elements within each math expression,
- remove a pair of outermost parentheses enclosing the math expression,
- remove all attachments, such as subscript, superscript, underscript, and overscript, from the expression, and
- if the math expression is an (in)equality, extract the object explained by the (in)equality, that is subexpression on the left side of the (in)equality symbol.

The purpose of these preprocessing steps is to obtain the relationships between math expressions when either:

- two math expressions have a similar meaning (i.e. the two expressions have the same base form), OR
- the meaning of a math expression helps determine the meaning of another expression (e.g. the former is a subexpression of the latter)

even though these expressions have different MathML representations.

Having obtained a dependency graph, we can use the math-level textual information (i.e. context window, descriptions, and noun phrases) from each child expression to also represent the target expression. As a result, we can associate each mathematical expression not only with its own text information, but also with the text information from each of its children.

2.4 Indexing Math Expressions

We adopt Apache Solr as the search platform of our system. There are three indexes used by our system, each of which contains math-, paragraph-, and document-level information. Each index contains multiple fields, which are shown in the Table 1.

The math index contains both encoded math expressions and textual information explaining the expressions. In Table 1, fields p_opaths_op to p_sister represent encoded math expressions using path-based encoding technique applied to MathML Presentation format, while c_opaths_op to c_upaths_arg represent the results from the same encoding technique applied to MathML Content. We separate vertical paths `opaths` and `upaths` directing to operators from ones directing to identifiers (`mi` and `ci`) and numbers (`mn` and `cn`). Therefore, in our database, there are two fields for each type of vertical path, namely one whose name ending in "_arg" (vertical paths to arguments, i.e., identifiers and numbers) and ones ending in "_op" (paths to operators).

Fields p_stree to p_sigure and c_stree to c_sigure are the result from hash-based encoding techniques applied to MathML Presentation and Content format. The rest of fields in the math index represent the textual information found surrounding the math expressions. The textual information contained by these textual fields, together with ones in paragraph and document indexes, are described in the Section 2.2.

2.5 Combining All Types of Information for Searching

During searching, the ranges of scores obtained for differ-

Table 1: List of fields in each index

Index Name	Field
Math	p_opaths_op p_opaths_arg p_upaths_op p_upaths_arg p_sisters c_opaths_op c_opaths_arg c_upaths_op c_upaths_arg p_stree p_mtrick p_sigure c_stree c_mtrick c_sigure contexts descriptions noun_phrases contexts_depgraph descriptions_depgraph noun_phrases_depgraph
Paragraph	all_words*
Document	title abstract keywords descriptions noun_phrases all_words

*Field is not used in Wikipedia task.

ent types (or fields) of information are varied. Therefore, we need to perform a score normalization step prior to obtaining final score. We use

$$finalScore = \frac{rawScore}{1 + rawScore} \quad (1)$$

as the normalization function.

In addition, we estimate cold-start weight for each field, since we consider that different fields may have different importance. To obtain this set of weights, we use multiple linear regression [4]. For the purpose of this regression task, we construct a training set using the relevance judgment result released by the NTCIR-11 Math-2 Task [1]. Procedure 1 specifies the detailed steps to construct the training set.

Procedure 1 (Training Set Construction).

1. Get a math expression f_i^* to represent each retrieval unit (paragraph) p_i in the training set.
   1.1. Set $f_i^* = \text{argmax}_{f_{ij} \in p_i} score(q, f_{ij})$ and $score(q, p_i) = score(q, f_i^*)$, where q is a query that contains math and keywords, and $score(q = query, f = math)$ is defined by Lucene's default TF-IDF similarity with the subscore from each field being normalized using Eq. 1.
2. Get features for each retrieval unit p_i in the training set.
   2.1. For each topic, compose a query q_{field} for each field we have in all indexes.
   2.2. Features: $s_{field} =$
   $$\begin{cases} score(q_{field}, f_i^*) \text{ for } field \text{ in math index} \\ score(q_{field}, p_i) \text{ for } field \text{ in paragraph index} \\ score(q_{field}, d_i) \text{ for } field \text{ in document index} \end{cases}$$
   where d_i is the document where p_i lies.
   2.3. Response: relevancy score (0 to 4) provided in judgment result.

For the searching procedure, both score normalization and cold-start weights are specified in the query using FunctionQuery provided by Solr. We obtain initial ranked lists of 2,000 units from math, paragraph, and document indexes. For the math index, we expand the ranked list in increments of 2,000 until the number of unique paragraphs referred to by the list is at least 2,000.

2.6 Unification

The motivation of applying unification as post-processing module in our search system is to put math expressions that can be instantiated from the query to the rank higher than the one that cannot. The unification module treats each symbol (i.e. operator, identifier, and number) found in math expressions as a constant. Only free variables, i.e. qvars, found in query math expressions are treated as variables. Given two terms, each of which can be either a constant or a variable, they unify if they are the same term or if they contain variables that can be uniformly instantiated with terms in such a way that the resulting terms are equal. For instance, query $p = mv$ (all terms are constants and there is no qvar) will only unify to math expressions that have the same presentation, i.e. $p = mv$. On the other hand, query $y = 5x + C$ (C is a `qvar`) will unify to $y = 5x + 9$, since variable C can be instantiated to constant 9. The unification by variable instantiations is allowed when the instantiations are compatible. For an example, query $X = X$ (X is a `qvar`) will not unify to $a = b$. This rule makes sense considering the query asks for an equality where the terms in the left- and right-hand side of the equality symbol are the same.

We implement this module by exploiting the unification feature from Prolog. To use this feature, we transform each math expression into its functor form (prefix notation). This functor form is obtained by utilizing the semantic provided by MathML Content representation of the math expression. For instance, given a math expression $a + b * c$, its functor form is: apply(plus, a, apply(times, b, c)).

We use this unification module to boost the similarity score from the math index. If a query and the math expression (or any subexpression of the expression) returned by our search system unify, we double the math-level similarity score of the expression while keep its paragraph- and document-level scores as they are.

3. ARXIV TASK

3.1 Experiment Setup

There are three common features we apply for obtaining the ranked list for arXiv task, which are as follows.

- utilize textual information obtained from exploiting dependency graph
- use cold-start weights obtained by multiple linear regression

- apply unification to boost the math-level score

Our system also applies the same set of features for Wikipedia task, which will be explained in the next section. The list of features we used for each submission in this arXiv task is depicted by Table 2.

The processing pipeline used for both subtasks in the arXiv task follows Fig. 1. For the Simto subtask, however, there are some additional details as follows.

- Prior to encoding the query math expressions, our system strips `simto` and `exact` tags, which are specially defined for Simto subtask. Hereinafter, the encoding process is the same as the Main subtask.
- The unification module utilize the simto region for boosting the similarity score.
  - If the query (with `simto` and `exact` are stripped) and the returned math expression (or any of its subexpressions) unify, the math-level similarity score is doubled. This is the common process that is also applied in the Main subtask. ELSE
  - Our system treats a simto region as a `qvar`, then replace this region with a single free variable. If the resulting query math expression and the (subexpression of) returned expression unify, our system multiplies the similarity score by 1.5.

In this subtask, the searching and ranking module of our system cannot utilize the regions of exact matching inside a similarity region yet.

3.2 Experiment Results

The performance of our system in arXiv task, which is measured using trec_eval, is shown in Table 3. For both Main and Simto subtasks, our system achieves the best precision among other teams. The discussion about this result will be conducted for each subtask.

Main Subtask

Among our four submissions, MCAT_allfields_nowgt_unif (a-nw-u) and MCAT_allfields_lr_unif (a-lr-u) obtain the best precision. In addition, our submission MCAT_nodep_lr_unif (nd-lr-u), which gives the lowest performance among our submissions, already performs slightly better than ICST, especially in high relevance measurements. However, it does not happen for partial relevance judgment up to P@10 measurement, where nd-lr-u is outperformed by ICST. Interestingly, at P@15 and P@20, this situation is reversed, where nd-lr-u now performs better. This indicates that this submission provides better coverage, i.e. recalling/retrieving relevant units, but has an issue in ranking the retrieved units.

We use the result from our four submissions to evaluate the effectiveness of three features we utilize: dependency graph, cold-start weights, and unification.

- *Dependency graph* has a major impact in our system. Compared to nd-lr-u, the submission a-lr-u, which exploits dependency graph, delivers significant precision improvement.
- *Cold-start weights* does not perform well enough. The submission a-lr-u, which uses cold-start weights, does not outperform a-nw-u consistently. It outperforms a-nw-u only in three occasions, namely P@5 (in partial relevance judgment) and P@10 (both in relevance and partial relevance), while the opposite happens in the other five occasions.
- *Unification* performs well enough. The submission a-lr-u outperforms a-lr in most of the measurements. At P@15 and P@20, unification performs well for partial relevance, but does not for relevance judgment. This suggests the that unification module obtains several partially relevant math expressions that were previously outside of the top-20 and boosts their scores. As a consequence, these expressions may now be ranked higher than several highly-relevant expressions that were previously already in the top-20. If these highly-relevant expressions (and all of its subexpressions) and the query do not unify, then they may be now out of the top-20.

Simto Subtask

Similar to the Main subtask, our submission nd-lr-u, which gives the lowest performance among our submissions, performs slightly better than MIRMU, especially in the high relevance measurements. However, unlike in the Main subtask, nd-lr-u does not perform well at P@20 of partial relevance judgment. In addition, while a-nw-u delivers the best performance in high relevance judgment, which also happens in the Main subtask, a-lr is the one who provides the best precision (except at P@5) in partial relevance judgment. We suggest that these two trends, which are not found in the Main subtask, may happen due to our attempt to relax the unification module during handling simto region (i.e. replacing each `simto` in a query with a `qvar` and multiplying the similarity score of the target math expression by 1.5 if the (subexpression of) expression and the query unify).

We evaluate the effectiveness of our three features in this subtask as follows.

- *Dependency graph* has a major impact in our system. The submission a-lr-u almost doubles the precision of nd-lr-u in partial relevance.
- *Cold-start weights* does not perform well. Submission a-lr-u is even consistently outperformed by a-nw-u.
- *Unification* surprisingly does not consistently perform well. Submission a-lr-u performs better than a-lr in the high relevance judgment, but not in the partial one (is good only up to P@5).

Discussion

Having examined the results from both Main and Simto subtasks, we found that the best P@5 obtained by our systems in both subtasks is similar, i.e. .2828 compared to .2750 in high relevance and .5586 to .5500 in partial relevance. However, in the Simto subtask, the drops of precision (from P@5 to P@10, P@10 to P@15, and P@15 to P@20) of submissions that are applying unification are steeper than ones in the Main subtask. We argue that this happens due to the same reason we have already mentioned above, i.e. out attempt to relax the unification module in Simto subtask. To tackle this problem in the future, we need to maintain some structure within simto region, instead of just replacing

Table 2: Features used in each submission

Run ID	dependency graph	cold-start weights	unification
arXiv task (Main and Simto subtasks) and Wikipedia task (MathWiki subtask)			
MCAT_nodep_lr_unif (nd-lr-u)		O	O
MCAT_allfields_nowgt_unif (a-nw-u)	O		O
MCAT_allfields_lr (a-lr)	O	O	
MCAT_allfields_lr_unif (a-lr-u)	O	O	O
Wikipedia task (MathWikiFormula subtask, no textual query)			
MCAT-browse_allfields_nowgt_unif (a-nw-u)			O
MCAT-browse_allfields_lr (a-lr)		O	
MCAT-browse_allfields_lr_unif (a-lr-u)		O	O

it with a `qvar`, when applying relaxed unification. In addition, we can later vary the multiplier, which is now fixed at 1.5, based on the depth of the structures of math expressions we maintain within the simto regions.

We also found out that the cold-start weights obtained using multiple linear regression do not work well. We suggest that this is caused by the negative weights we obtain for several fields in our system. Furthermore, these negative weights may be difficult to interpret, since for a given query, any match in a field is supposed to "increase", not decrease, the similarity score of a retrieval unit. To handle this problem in the future, we can consider to restrict the weights to positive numbers during training process.

The score normalization in Eq. 1 that we applied during searching, however, works well. This step allows our system to utilize many fields for searching without any concern regarding whether subscores from certain fields will improperly dominate the final score, especially when the number of terms generated for each of these certain fields is much larger than the number of terms generated for other fields. As a result, our system can finish at the top for both subtasks in the arXiv task.

4. WIKIPEDIA TASK

4.1 Experiment Setup

The list of features we used for each submission in the Wikipedia task is depicted by Table 2. For the MathWiki subtask, we employ the same set of features as in the arXiv task. Subsequently, we estimate the cold-start weights for all the fields specified in Table 1, except all_words field in the paragraph database. It is not a trivial task to correctly extract paragraphs from Wikipedia articles, thus we do not employ paragraph database for the Wikipedia task.

For the MathWikiFormula subtask, however, the dependency graph feature is not applicable as this subtask gives a set of topics that contain only math expressions, without any textual terms. Moreover, the absence of textual terms in this subtask leads to less fields specified in the practical (Solr) query. We re-estimate the cold-start weights for the fields that will only be used in this subtask. Apart from details we specified above for each subtask in the Wikipedia task, the rest of the processes are the same as in the Main subtask of the arXiv task.

4.2 Experiment Results

The performance of our system in Wikipedia task is shown in Table 3. Same as in the arXiv task, the performance is measured using trec_eval. The discussion about this result is provided as follows.

MathWiki Subtask

In the MathWiki subtask, unlike in the arXiv task, our system does not deliver the highest performance. However, several trends observed in this subtask are the same as trends found in the arXiv task - Main subtask, such as:

- *Cold-start weight* does not perform well enough.
- *Unification* performs really well. The submission a-lr-u significantly outperforms a-lr in all precision measurements.

An interesting trend that is observed only in the MathWiki subtask is that the dependency graph does not perform well enough, unlike in the arXiv task. We suggest that this happens because many topics in this subtask have extremely common textual terms (similar to stop words). From our manual observation, there are only 5 out of 30 topics that contain textual terms whose sparsity differs from stop words', i.e. topics #8, #9, #12, #24, and #29. Due to this issue, the use of dependency graph to enrich the textual information of the indexed math expression is not beneficial in this search scenario.

In addition, we observe that the ranked lists returned by our system for several topics contain some retrieval units that match only with the very common terms found in the textual query. This indicates that the tf-idf scoring we use cannot discriminate common terms well enough, and as a consequence, our system sometimes cannot handle queries that are formed as questions (containing many common terms) well enough. This condition is worsened by the fact that the use of dependency graph to enrich the indexed textual information will increase the number of indexed common words, thus will further harm the search results in this subtask. We suggest that this is also the reason why in this subtask our system does not perform as well as in the arXiv task.

MathWikiFormula Subtask

Our system delivers the best precision result in the MathWikiFormula subtask. The trends found in this subtask are the same as ones in the other subtasks, i.e. score normalization and unification perform well, but the cold-start weight does not. However, unlike in the MathWiki subtask, the topics in this subtask do not contain textual terms. Thus, our system does not encounter any issue with terms similar to stop words in this subtask.

Discussion

Having examined the results from both MathWiki and MathWikiFormula subtasks, we found that the score nor-

Table 3: Search performances

RunID	Relevant				Partially Relevant			
	P@5	P@10	P@15	P@20	P@5	P@10	P@15	P@20
arXiv task - Main subtask								
MCAT_nodep_lr_unif (nd-lr-u)	.2345	.1966	.1747	.1586	.4828	.4793	.4690	.4552
MCAT_allfields_nowgt_unif (a-nw-u)	.2828	.2379	.2184	.1948	.5448	.5345	.5149	.4897
MCAT_allfields_lr (a-lr)	.2552	.2379	.2092	.1828	.5586	.5379	.5034	.4690
MCAT_allfields_lr_unif (a-lr-u)	.2621	.2448	.2046	.1810	.5586	.5483	.5126	.4707
ICST	.2276	.1862	.1632	.1362	.5517	.4966	.4299	.4000
arXiv task - Simto subtask								
MCAT_nodep_lr_unif (nd-lr-u)	.2000	.1125	.0917	.0687	.3250	.2000	.1667	.1500
MCAT_allfields_nowgt_unif (a-nw-u)	.2750	.2125	.1667	.1375	.5500	.4250	.3667	.3250
MCAT_allfields_lr (a-lr)	.1750	.1375	.1417	.1313	.4250	.3875	.3833	.3687
MCAT_allfields_lr_unif (a-lr-u)	.2750	.1625	.1500	.1313	.5000	.3625	.3333	.3063
MIRMU	.0250	.0625	.0750	.0625	.3000	.2500	.2677	.2813
Wikipedia task - MathWiki subtask								
MCAT_nodep_lr_unif (nd-lr-u)	.2933	.2467	.2267	.1983	.6200	.5867	.5711	.5333
MCAT_allfields_nowgt_unif (a-nw-u)	.3600	.3233	.2689	.2433	.7667	.7167	.6867	.6533
MCAT_allfields_lr (a-lr)	.2467	.2233	.2044	.1817	.5533	.5133	.4956	.4650
MCAT_allfields_lr_unif (a-lr-u)	.2867	.2533	.2244	.1983	.6067	.5700	.5533	.5183
ICST	.4733	.3767	.2978	.2617	.8533	.7900	.7133	.6600
Wikipedia task - MathWikiFormula subtask								
MCAT-browse_allfields_nowgt_unif (a-nw-u)	.4900	.3900	.3317	.2825	.9100	.8400	.8067	.7687
MCAT-browse_allfields_lr (a-lr)	.4100	.3225	.2733	.2325	.7700	.7375	.7083	.6650
MCAT-browse_allfields_lr_unif (a-lr-u)	.4550	.3475	.2950	.2613	.7850	.7475	.7100	.6700
RITUW	.4400	.3225	.2700	.2300	.8400	.7650	.7317	.7063

malization and unification have a positive impact on our search system. However, the cold-start weights have a negative impact. These tendencies are similar to the ones exhibited in the arXiv task. In contrast, the results from the Wikipedia task raise our concern that our math search system may not be reliable enough to handle extremely common textual terms in the query. One of several possible approaches to solve this issue is either to properly configure Solr or to simply remove these common textual terms from the query prior the searching procedure. Another method is to map several common terms, such as "what", "why", "solve", and "define", to related terms for retrieval. For instance, term "what" might be mapped to "definition." We then use this mapping to encourage hits which both contain and define the math query more than those that only contain the query. Another possibility is to take into account the proximity between the retrieved math expression and the matching textual term in the original document during searching.

5. SPACE UTILIZATION AND PROCESSING TIME

The execution environment of our system is two Linux servers, each of which has 64 processors. We use one server to store math indexes, and another server to store paragraph and document indexes. For the arXiv task, the sizes of math, paragraph, and document indexes are 426.39GB, 898.18MB, and 1.26GB, respectively. For the Wikipedia task, the sizes of math and document indexes are 2.21GB and 365.94MB, respectively.

In addition, the processing time required by our system for each submission in the arXiv - Main subtask is shown in Figure 2. The first three plots depict the time needed to obtain initial ranked lists from document, paragraph, and math indexes, respectively. For each plot, the first four boxplots correspond to our four submissions. These four submissions utilize score normalization step and some of them use also cold-start weights. The fifth boxplot "all (default)" denotes the query time when none of these two features are implemented.

The order of the query time from the shortest to the longest is document, paragraph, and then math index. Two factors influencing the query time are: (i) the more retrieval units an index stores, the longer the query time is and (ii) the more query terms and fields we specify for searching in a certain index, the longer the query time is. Among our four submissions, the a-nw-u has the shortest query time. This happens because without the use of cold-start weights, the query (i.e. FunctionQuery) becomes simpler and less calculation is performed by Solr. This reasoning also explains why all our four submissions require query time that is significantly longer than the "all (default)" case. All our four submissions use FunctionQuery to apply score normalization to each field specified in the query. The more fields a query specifies, the more score normalization step Solr has to perform, thus the longer the query time is. This observation shows that the positive impact of score normalization procedure, i.e. ensures that none of the subscores from certain fields will improperly dominate the final score, comes at the cost of the query time.

The last plot in Fig. 2 displays the wall-clock time taken by our system to complete the unification procedure that is parallelized into 50 processes. All three submissions have the same median of unification time. However, the time distribution of submission a-nw-u is more skewed than the other two submissions. Since all these submissions contain the same number of math expressions in the initial ranked list,

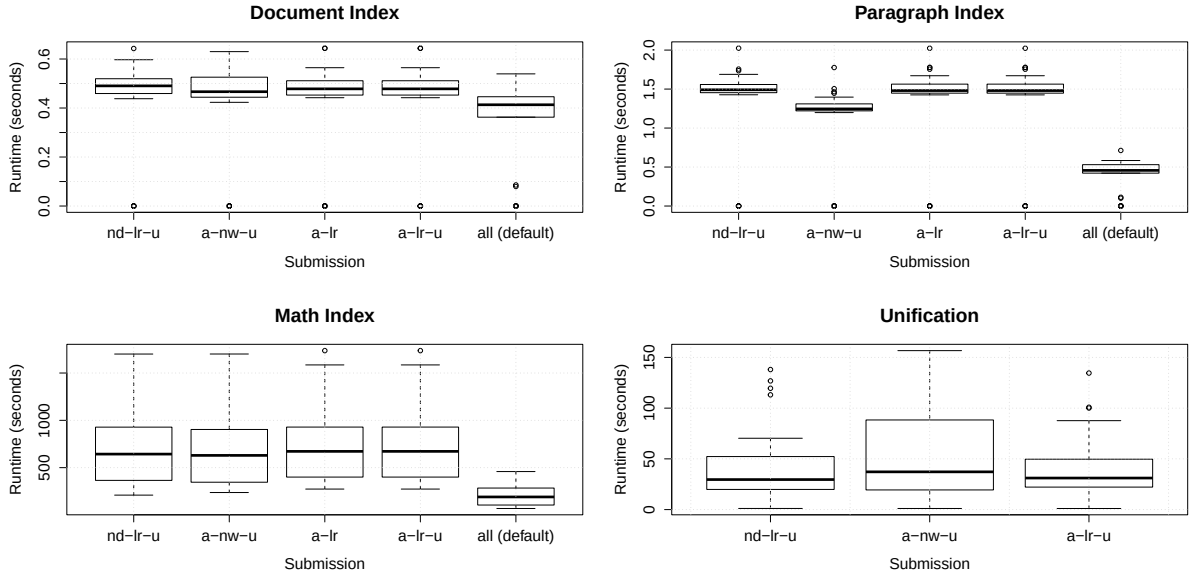

Figure 2: Query time of our system in the arXiv - Main subtask.

this difference in skewness indicates that the initial ranked list of a-nw-u contains more complex math expressions, i.e. have deeper levels of MathML representations and contain more sub-expressions, than the ranked lists from nd-lr-u and a-lr-u. As unification step is performed between math query and each sub-expression of the retrieved math expressions, the more sub-expressions a math expression contains, the longer the unification time is.

6. CONCLUSION

We have presented the participation of our MCAT system in the NTCIR-12 MathIR Task. We implemented several new building blocks in our search system for this task: hash-based encoding technique, several types of textual information to cover three levels of information granularity (math expression, paragraph, and document levels), new approach for generating dependency graph of math expressions, score normalization, cold-start weights, and unification. We then investigated the effectiveness of some of these approaches and found out that dependency graph, score normalization, and unification are integral parts of our search system. The cold start weights, however, do not have a good impact on the search performance due to the negative weights obtained by several database fields. We also find an issue regarding the inability of tf-idf scoring technique to handle extremely common textual terms in the Wikipedia task.

One of the future works is to set the unification module to vary the boosting score based on the depth location of the subexpression inside the retrieved math expression to which the query unifies. We also consider investigating the effectiveness of cold-start weights if their values are restricted to positive numbers.

Acknowledgments

This work was supported by JSPS KAKENHI Grant Numbers 25245084, 14J09896, and 15H02754.

7. REFERENCES

[1] A. Aizawa, M. Kohlhase, I. Ounis, and M. Schubotz. NTCIR-11 Math-2 task overview. In *Proc. of the 11th NTCIR Conference*, 2014.

[2] G. Y. Kristianto, G. Topić, and A. Aizawa. Exploiting textual descriptions and dependency graph for searching mathematical expressions in scientific papers. In *Proc. of the 9th ICDIM*, 2014.

[3] G. Y. 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.

[4] G. Y. Kristianto, G. Topić, and A. Aizawa. Combining effectively math expressions and textual keywords in math ir. In *Proc. of the 3rd International Workshop on Digitization and E-Inclusion in Mathematics and Science*, 2016.

[5] G. Y. Kristianto, G. Topić, F. Ho, and A. Aizawa. The MCAT math retrieval system for NTCIR-11 Math track. In *Proc. of the 11th NTCIR Conference*, 2014.

[6] S. Ohashi, G. Y. Kristianto, G. Topić, and A. Aizawa. Efficient algorithm for math formula semantic search. *IEICE TRANSACTIONS on Information and Systems*, E99-D(4):979–988, 2016.

[7] S. Rose, D. Engel, N. Cramer, and W. Cowley. Automatic keyword extraction from individual documents. In *Text Mining: Applications and Theory*. John Wiley & Sons, Ltd, Chichester, UK, 2010.

[8] G. Topić, G. Y. 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.

[9] R. Zanibbi, A. Aizawa, M. Kohlhase, I. Ounis, G. Topić, and K. Davila. NTCIR-12 mathir task overview. In *NTCIR*. National Institute of Informatics (NII), 2016.