Dowsing for Answers to Math Questions: Doing Better with Less

Andrew Kane, Yin Ki Ng and Frank Wm. Tompa[1]

[1]David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada, N2L 3G1

Abstract

We present our application of a new math-aware search engine to the 2022 ARQMath Lab. This extends the annual submissions from the University of Waterloo's "MathDowsers," for which we twice produced the best Task 1 participant run.

This year the major improvements result from greatly reducing the number of math tuples used to represent a math formula, which not only saves considerable space in the index and reduces query terms, but also improves retrieval effectiveness as measured by normalized discounted cumulative gain with unjudged documents removed (nDCG'). In addition, we have re-implemented math tuple generation in a stand alone code base and replaced Tangent-L, the previous Lucene-based search engine, with a new engine that simplifies experimentation.

Experimental results show that the new system has substantially better performance than its predecessor in terms of nDCG', MAP', and P'@10, both when trying to find answers to math questions (Task 1) and when trying to find similar formulas (Task 2). From this we conclude that traditional text retrieval systems can easily be enhanced to become effective math-aware search engines using the tools we have developed.

Keywords

Community Question Answering (CQA), Mathematical Information Retrieval (MathIR), Symbol Layout Tree (SLT), math tuples, Mathematics Stack Exchange (MSE), ARQMath Lab, formula matching

1. Introduction

The ARQMath Lab at CLEF 2022 [1] continues previous years' Labs [2, 3] to examine how best to search a community question answering (CQA) repository to find answers to questions involving math data. The Labs use a collection of questions and answers from Math Stack Exchange[2] (MSE) between 2010 and 2018 consisting of approximately 1.1 million question-posts and 1.4 million answer-posts. Task 1, the CQA task, asks participants to return potential answers to unseen mathematical questions among existing answer-posts in the collection. Task 2 is the formula retrieval task, where formulas in the context of specific unseen questions serve

CLEF 2022: Conference and Labs of the Evaluation Forum, September 5–8, 2022, Bologna, Italy

arkane@uwaterloo.ca (A. Kane); kiking0501@gmail.com (Y. K. Ng); fwtompa@uwaterloo.ca (F. Wm. Tompa)

https://www.linkedin.com/in/arkane/ (A. Kane); https://www.linkedin.com/in/kiking0501/ (Y. K. Ng); http://www.cs.uwaterloo.ca/~fwtompa/ (F. Wm. Tompa)

0000-0002-1907-9535 (F. Wm. Tompa)

[2]https://math.stackexchange.com

as queries for matching relevant formulas from question-posts and answer-posts in the same collection. Task 3, added this year, asks participants to formulate an answer to an unseen mathematical question using any resources and techniques available to them.

In the first year, the Waterloo team of MathDowsers participated in Task 1 only. We demonstrated how tagged mathematical questions can be automatically transformed into formal queries consisting of keywords and formulas and how the resulting formal queries can be effectively executed against a corpus [4]. In particular, the corpus of MSE question-answer pairs was indexed by a traditional math-aware search engine, namely Tangent-L [5] built on top of Lucene using a traditional text-based ranking algorithm (BM25+ [6]) with simple adaptations to handle math formulas within the engine. A key component of this approach is to represent formulas as a set of *math tuples* that reflect certain features found in the formulas and to treat the tuples as words in the engine.

In the second year, we participated again as the MathDowsers team for Task 1 and for Task 2, with the goal to continue exploring the potential of a traditional math-aware query system in tackling both tasks [7]. We demonstrated, among other things, that augmenting the set of math tuples with tuples that capture repeated symbols slightly increases retrieval effectiveness, that reducing noise in the data makes a substantial impact, and that searching a corpus of visually distinct formulas is effective for Task 2.

The success of Tangent-L in both ARQMath-1 and ARQMath-2 could result from several factors, but chief among them is (1) the use of a carefully constructed corpus of documents to be searched and (2) the extraction of features that serve well in representing math formulas. It is clear that anyone could adopt the former, and this year we show how anyone can incorporate the latter into their approach.

To this end, in this paper we present our re-implementation of the math feature extraction as a standalone code base. We also show that it can be applied universally by using that sub-system with a different, more conventional, text-based search engine. As it turns out, the first author had been developing such an engine in order to explore various aspects of search technology, and the ARQMath Lab provides an opportunity to test the new engine against a realistic application. We hope that with this presentation of our tools, others can use our constructed documents and math tuple generator as a basis on which to apply even smarter processing, regardless of the search engine they choose.

In addition to separating the math extraction code, we re-examine the mapping from formulas to sets of math tuples and show that capturing certain features only, and ignoring other features, improves effectiveness for both Task 1 and Task 2. We also show that including some extracted text from the LaTeX representation of formulas improves formula matching in Task 2. (We did not participate in Task 3.)

This paper first describes the three large processing steps used in our system:

1. document and query construction, described in Section 2;
2. math tuple generation, described in Section 3; and
3. indexing and querying with the core engine, described in Section 4.

Next, this year's experimental results are detailed in Section 5, and the conclusions follow thereafter.

2. Document and Query Construction

2.1. Document Corpus for Task 1

For ARQMath-3, we continue to use enhanced question-answer pairs as indexing units for the document corpus, because worse performance can be observed for the ARQMath-1 and ARQMath-2 benchmarks if the content of the associated question is dropped and only text from each answer is indexed [8]. As before, for every answer-post in the MSE database, we use a series of preprocessing steps[3] to create a document consisting of an "enhanced question" (including the Question Title, Question Body Text, Question Tags, Question Comments, Duplicate Post Titles, and Related Post Titles) together with an "enhanced answer" (including the Answer Body Text and the Answer Comments). Figure 1 illustrates the fields indexed as part of each question-answer pair.

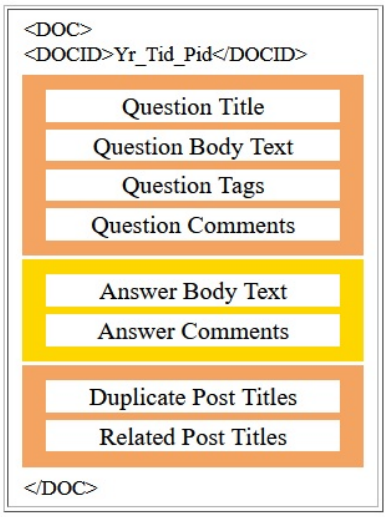

Figure 1: Structure of a corpus document containing a question-answer pair and its associated data.

Note that to be consistent with other search systems and datasets, we use a simple *TREC* encoding format for documents: each document starts and ends with the tags <DOC> and </DOC> and the first element inside a document is the document identifier tagged with <DOCNO>. . . </DOCNO> [9, p. 24]. In the Task 1 corpus, the DOCNO value encodes the year, threadid (id for the question-post), and postid of the answer-post. By using this convention, many documents can be stored in a single (compressed) file and still be appropriately identified, which saves space and avoids extensive file openings and closings.

2.2. Document Corpus for Task 2

Last year we determined that searching a corpus of visually distinct formulas outperforms picking the best formula from the Task 1 results [7]. Therefore, for ARQMath-3, we again

[3] https://github.com/kiking0501/MathDowsers-ARQMath

create a corpus of visually distinct formulas appearing in either question or answer posts and represented in Presentation MathML. Searching this corpus gives us a ranking R of visually distinct formulas, from which any occurrence in a question or answer post can be determined arbitrarily. (In this corpus, the DOCNO value for each document encodes the formula-id, the post-id containing the formula, and the visual-id representing the visually distinct form of the formula.)

Because some formulas failed to be converted from LaTeX into MathML and because there is potentially useful text buried in math tuples (particularly within tuples generated from text nodes in the symbol layout tree), we build a variant corpus in which each document includes the math tuples for a visually distinct formula together with words selected from the LaTeX representation of that same formula (Figure 2). More specifically, given a LaTeX formula, the text extractor treats all non-alphanumeric characters as white space and removes all resulting tokens that are shorter than three characters in length. Thus, for example, the expression

```
$x\in \mathbb{Q} \land x=\frac pq \text{ in lowest terms}$
```

(encoding "$x \in \mathbb{Q} \land x = \frac{p}{q}$ in lowest terms") produces "mathbb land frac text lowest terms."

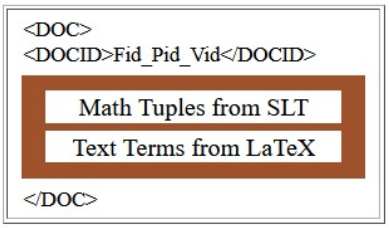

Figure 2: Structure of a corpus document containing the MathML representation for a visually distinct formula and text extracted from its LaTeX representation.

2.3. Query Formation

As in previous years, we represent queries using a simple XML encoding format as shown in Figure 3, and we convert the given MSE questions ("topics") to this form using the code originally developed for ARQMath-1[4] [4].

```
<Topics>
  <Query topic="topic-id">
    <input id="input-id" type="formula">...</input>
     ...
    <input id="input-id" type="keyword">...</input>
  </Query>
   ...
</Topics>
```

Figure 3: Structure of a query stream.

[4]https://github.com/kiking0501/MathDowsers-ARQMath

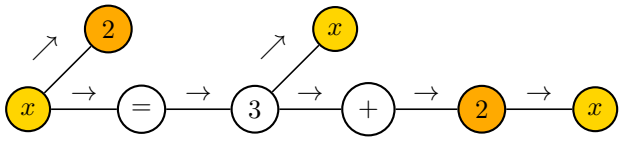

Figure 4: Symbol Layout Tree for $x^2 = 3^x + 2x$ with repetitions highlighted.

Our Task 2 queries do not use keywords, so they are not generated during conversion. In addition, to search the variant formula corpus, a second processing step augments each query with text extracted from the LaTeX corresponding to the given formula-id.

3. Representing Formulas by Math Tuples

3.1. Math Tuples in Tangent-L

The ability to match formulas is what makes a search engine "math-aware," though clearly there are many approaches to implementing this. The mechanism piloted by Tangent [10] and further developed for Tangent-L [5, 7] is to convert a symbol layout tree (expressed in Presentation MathML) into a set of features called *math tuples*. Since last year's Lab, the set of features supported by Tangent-L include symbol pairs, terminal symbols, compound symbols, duplicate symbols, and augmented locations, as shown in Table 1 for the symbol layout tree depicted in Figure 4. This year we use the same set of features, but restrict the number of tuples as explained in the remainder of this section.

3.2. Revisiting Tuple Generation for Duplicated Symbols

If a symbol repeats k times where $k > 1$, then $\binom{k}{2}$ *duplication tuples* (math tuples representing duplicated symbols) are generated for that symbol. As such, $O(n^2)$ math tuples are generated and indexed for a formula containing n symbols. This becomes problematic for large polynomials, matrices with repeated symbols in the entries, and many other formulas that include one or more specific symbols many times. In fact, some documents in the ARQMath collection produce huge numbers of repetitions. As a relatively small example, question post #2274526 includes the matrix

$$\begin{array}{cccc} & X & 0 & 1 & 2 \\ Y & & & & \\ 0 & & p_{0,0} & p_{0,1} & p_{0,2} \\ 1 & & p_{1,0} & p_{1,1} & p_{1,2} \\ 2 & & p_{2,0} & p_{2,1} & p_{2,2}. \end{array}$$

which contains 8 repetitions of 0, 8 repetitions of 1, 8 repetitions of 2, and 9 repetitions of p, resulting in 240 duplication tuples and another 240 corresponding location tuples. Including all these tuples deteriorates ranking performance and bloats the index. (We hypothesize that this blowup is the reason that based on the ARQMath-2 Lab, we recommend weighting duplication

[5] These were previously called *Repeated symbols* [7].

Table 1
Math tuples for the formula in Figure 4.

Tuple Type	*Math Tuples Generated*		*Remark*
Symbol pairs:	(x , 2, $\nearrow$) (= , 3, $\rightarrow$) (3 , +, $\rightarrow$) {2 , x, $\rightarrow$}	(x , =, $\rightarrow$) (3 , x, $\nearrow$) (+ , 2, $\rightarrow$)	Pairs of adjacent symbols with connecting edge
Terminal symbols:	(2 , ϕ) (x , ϕ)	(x , ϕ)	Symbols with no outedges
Compound symbols:	(x , $\rightarrow\nearrow$)	(3 , $\rightarrow\nearrow$)	Symbols multiple outedges
Duplicate symbols[5]:	{x, $\rightarrow\rightarrow\nearrow$} {$x$, $\rightarrow\rightarrow\rightarrow$, $\nearrow$}	{x, $\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow$} {2, $\rightarrow\rightarrow\rightarrow\rightarrow$, $\nearrow$}	Repeated symbols on the same path or on two paths from a common ancestor
Augmented locations:	(x , 2, $\nearrow$, -) (= , 3, $\rightarrow$, $\rightarrow$) (3 , +, $\rightarrow$, $\rightarrow\rightarrow$) (2 , x, $\rightarrow$, $\rightarrow\rightarrow\rightarrow\rightarrow$) (2 , ϕ, $\nearrow$) (x , ϕ, $\rightarrow\rightarrow\nearrow$) ($x$, $\nearrow\rightarrow$, -) {x, $\rightarrow\rightarrow\nearrow$, -} {$x$, $\rightarrow\rightarrow\rightarrow$, $\nearrow$, $\rightarrow\rightarrow$}	(x , =, $\rightarrow$, -) (3 , x, $\nearrow$, $\rightarrow\rightarrow$) (+ , 2, $\rightarrow$, $\rightarrow\rightarrow\rightarrow$) ($x$, ϕ, $\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow$) (3, $\nearrow\rightarrow$, $\rightarrow\rightarrow$) {x, $\rightarrow\rightarrow\rightarrow\rightarrow\rightarrow$, -} {2, $\rightarrow\rightarrow\rightarrow\rightarrow$, $\nearrow$, -}	Every tuple also represented with its path from the root

tuples so much lower than other math tuples when scoring how well a query matches a document [7].)

Two alternatives for limiting the number of tuples generated when a symbol repeats many times are to limit the number of repetitions to represent (e.g., only extract tuples for all pairs of the first few occurrences of a symbol in a formula), or to choose a subset of duplication tuples such that the cardinality is linear in the size of the formula rather than preserving all pairs of duplicated symbols. For either approach, it is important that small changes to a formula do not cause radically different tuples to be generated so that query formulas that are close to a corpus formula can be matched.

After some experimentation with the ARQMath-1 and ARQMath-2 benchmarks, we recommend the latter option: selecting a linear number of duplication tuples. More specifically, we perform a recursive traversal of the symbol layout tree (being sure to traverse the out-edges from a node in consistent order) and create a duplication tuple for each adjacent pair of repeated symbols in post-order. Thus, for the tree in Figure 4, the tuples $\{x, \rightarrow\rightarrow\rightarrow, \nearrow\}$ and $\{x, \rightarrow\rightarrow\nearrow\}$ would be produced, but $\{x, \rightarrow\rightarrow\rightarrow\rightarrow\rightarrow\}$ would not be produced because it would reflect a repetition between the first and third x encountered in a post-order traversal. For the example matrix shown above, this results in 29 duplication tuples plus another 29 corresponding location tuples, almost a 90% reduction. With this change, duplication tuples can be given the same

weight as the other math tuples generated, removing the need for a tuning parameter to choose a relative weighting.

A second characteristic of duplication tuples is that they include the repeated symbol in the first field. However, in many situations when a duplicated variable is matched in a formula, the specific value for that variable is not important; rather it is the fact that *some* variable is repeated with this particular pattern of relative paths. For the example in Figure 4, replacing the variable x by y should still be considered a good match.

Thus, instead of storing the specific variable name, it may be better to record that some variable is repeated with a specific relative path. Similarly, it may be more important to record that some operator repeats or some number repeats than to record which specific number or operator is duplicated. To accommodate this, a second modification to duplication tuples is to augment (or replace) the tuple with one that includes a wildcard that reflects the *type* of the symbol that is duplicated. For example, the wildcard equivalents of the *selected* duplicated tuples in Table 1 are {$?V$, $\rightarrow\rightarrow\nearrow$}, {$?V$, $\nearrow$, $\rightarrow\rightarrow\rightarrow$}, and {$?N$, $\nearrow$, $\rightarrow\rightarrow\rightarrow\rightarrow$}, as well as these tuples augmented with location.

3.3. Revisiting Tuple Generation for Augmented Locations

The bag-of-words model implemented by BM25 scoring means that the order of math tuples is immaterial. As a result, only the tuples that include locations (the path from the root to the feature being represented) anchor the feature with respect to the formula. Using the NTCIR benchmarks, we found that including location tuples improves ranking performance [5], although it doubles the index space for math tuples and increases the length of all documents that include formulas. However, at that time we did not explore whether *all* tuples should be so augmented or whether the benefit arises from having only some tuples augmented by location.

In fact, the original description states that in addition to indexing math tuples for symbol pairs: "we recommend including features to reflect terminal symbols, compound symbols, and locations of symbol pairs" [5], which would generate only the first seven math tuples for augmented locations in Table 1. However, the implementing code referenced by that paper augments tuples representing terminal symbols and compound symbols as well! Thus the first variant is to capture this distinction by allowing a user to specify which subset of the three types of tuple (symbol pairs, terminal symbols, or compound symbols) should be augmented with location tuples.

A second observation is that as features appear farther from the root of a query formula, they are increasingly less likely to be located in exactly the same locations in a closely matching document formula. Furthermore, those features are increasingly less likely to appear in *any* formula, and so when there happens to be a match, the inverse document frequency might be overwhelmingly high. Thus long location paths might be problematic in two ways: they might not match when they should and they might strongly match when they shouldn't. A solution is to generate augmented location tuples only for features that do not meet or exceed some maximum path length for their locations. If the cutoff for location paths is set at 4, for example, then location paths must have fewer than three nodes (and thus, two edges). With this setting, four of the location tuples shown in Table 1 would not be generated.

A related observation is that math tuples need not be anchored with respect to the root

Table 2
Location tuples for the formula in Figure 4.

Anchored at root only		*Anchored at root or = operator*	
(x , 2, $\nearrow$, -)	(x , =, $\rightarrow$, -)	(x , 2, $\nearrow$, -)	(x , =, $\rightarrow$, -)
(= , 3, $\rightarrow$, $\rightarrow$)	(3 , x, $\nearrow$, $\rightarrow\rightarrow$)	(= , 3, $\rightarrow$, -)	(3 , x, $\nearrow$, $\rightarrow$)
(3 , +, $\rightarrow$, $\rightarrow\rightarrow$)		(3 , +, $\rightarrow$, $\rightarrow$)	(+ , 2, $\rightarrow$, $\rightarrow\rightarrow$)
(2 , ϕ, $\nearrow$)		(2 , ϕ, $\nearrow$)	
		(x , ϕ, $\rightarrow\nearrow$)	
(x , $\nearrow\rightarrow$, -)	(3, $\nearrow\rightarrow$, $\rightarrow\rightarrow$)	(x , $\nearrow\rightarrow$, -)	(3, $\nearrow\rightarrow$, $\rightarrow$)
{x, $\rightarrow\rightarrow\nearrow$, -}	{x, $\rightarrow\rightarrow\rightarrow$, $\nearrow$, $\rightarrow\rightarrow$}	{x, $\rightarrow\nearrow$, -}	{x, $\rightarrow\rightarrow\rightarrow$, $\nearrow$, $\rightarrow$}
{2, $\rightarrow\rightarrow\rightarrow\rightarrow$, $\nearrow$, -}		{2, $\rightarrow\rightarrow\rightarrow$, $\nearrow$, -}	

of the formula only: other locations within the formula might well also serve as anchors for determining paths. One such set of potential anchors is the set of relational operators (i.e., =, <, $\neq$, $\equiv$, etc.). If relational operators can also serve as anchors, then symbols on the left side of the equation in Figure 4 would be anchored with respect to the root of the formula, but those on the right side of the equation would be anchored with respect to the location of the equality operator. Table 2 contrasts the augmented location tuples for root-only anchors vs. anchoring at relational symbols as well, assuming that the cutoff location length is set at 4 in both cases. Notice that all features on the left of the equals sign remain the same, but those on the right have (in this case) one fewer node preceding them. One side-effect of enabling additional anchors is that the "relative positions" stored in duplication tuples are now modified to reflect the fact that there are multiple anchor positions and each symbol is measured with respect to its nearest ancestor anchor. For example, because its path is measured from the equals sign, the superscript x on the right side is treated as if it were located one step closer to the x on the left side. A second side-effect is that more of the location tuples fall within the maximum path length threshold.

3.4. Converting With the Tuple-Generator

To support this year's experiments, the mathtuples program released last year was updated to support the extensions described in the previous two subsections of this paper. The revised program (Version 2) is now available,[6] and its interface is depicted in Table 3.

For each type of math tuple (e.g., symbol pair, terminal symbol), the user can specify whether or not that tuple should be included and whether or not it should be augmented by a corresponding location tuple. Thus, for example, `-T 0` indicates that no terminal symbols should be included; since all paths from the root include at least one node, `-T 1` indicates that terminal symbols should be included but not augmented with location tuples; because we reserve the value 99 to represent "unlimited," `-T 99` indicates that terminal symbols should be included and *all* should be augmented by location tuples; and (the default) `-T 8` indicates that terminal symbols should be included, and for each terminal symbol tuple, if the path from the root to the corresponding symbol includes fewer than 8 nodes, it should be augmented by a location tuple.

[6] https://github.com/fwtompa/mathtuples

Table 3
The math tuple generator.

Optional Arguments	Explanation	Default
`-W size`	Size of the window for symbol pairs (99 ⇒ unlimited)	1
`-S loc_lim`	Include Symbol pairs*	8
`-R loc_lim`	Include Relationship edge pairs*	0
`-T loc_lim`	Include Terminal symbols*	8
`-E loc_lim`	Include End-of-line symbols*	0
`-C loc_lim`	Include Compound symbols*	8
`-L loc_lim`	Include Long pairs without relationships*	0
`-A loc_lim`	Include Abbreviated relationships for long pairs*	0
`-D loc_lim`	Include Duplicate symbols*	8
`-docid tag`	String preceding each document identifier	"<DOCNO>"
`-a e|d`	Enable/disable "equality" operators as anchors	e
`-c`	Return the math tuples in context	tuples only
`-d dups`	Include duplication tuples for subset of 'VNOMFRTW'**	all
`-s`	Expand all tuples to include wildcard synonyms	no expansion
`-w wild_dups`	Include wild dupl'n tuples for subset of 'VNOMFRTW'**	all
*tuple inclusion:	$(loc\_lim \leq 0) \Rightarrow$ include no tuples of this type $(0 < loc\_lim < 99) \Rightarrow$ augment with location tuples whenever path has *fewer* than loc_lim nodes $(loc\_lim \geq 99) \Rightarrow$ augment with all location tuples	
**node types:	V(ariables), N(umbers), O(perations), M(atrices and parenthetical expressions), F(ractions), R(adicals), T(ext), W(ildcard of unknown type)	

Specifying which tuples to include for duplicate symbols is more complex because several options are involved. If `-D` is specified to be greater than 0, then the argument for `-d` signals which tuples should be created for duplication nodes. For example, `-d VN` specifies that duplication tuples for Variables and Numbers should be generated with the specific variable or number that is repeated being included in the first field. Similarly, the argument for `-w` signals which tuples with wildcards in the first field should be created for duplication nodes. For example, `-w VOT` specifies that duplication tuples for Variables, Operators, and Text should be generated with the corresponding typed wildcard in the first field. Finally, the argument for `-D` indicates whether or not the corresponding location tuples should also be generated.

Other arguments for the tuple generator indicate what window size to use for symbol pairs (`-W`) [5], whether (and where) document identifiers are included in the input (`-docid`), whether or not relational operators (equality et al.) are to serve as location anchors (`-a`), whether all the text outside each MathML expression is to be preserved (`-c`) or mathtuples only should be returned without the text within which the math expressions appear, and whether or not wildcard equivalents for every tuple should be generated (during indexing) so that wildcards in a query can be matched (`-s`) [5].

With these changes, every run in this year's experiments includes the math tuple generator in its pipeline, using arguments that match the experimental conditions being investigated.

4. Search System Architecture

The goal of the *μTextSearch* engine[7] is to produce state-of-the-art research level performance (both efficiency and effectiveness) with a very small amount of code. However, the variant used here only partially realizes this goal. In particular, query execution efficiency and extreme index compression have not yet been addressed.

The value in such a search engine is more than just demonstrating that complicated search engines might not add much value. Having an engine implemented in a small amount of code also makes it easier to understand the entire engine and, it is hoped, to implement changes. As such, this engine should make it easier to try new research experiments, as it did for us during the work presented in this paper.

4.1. Core Search Engine Components

The core *μTextSearch* engine is very simple and is constructed as a series of components that can be implemented (and understood) separately.

The *mstrip* component removes data that should not be indexed, such as tags and comments. A command line parameter allows interpretation of queries, converting each to a single line starting with the query identifier followed by a semicolon and then a list of tokens.

The *minvert* component splits the input data into documents and then into a stream of tokens of two types: text (stemmed using the Porter stemmer [11]) and math (each surrounded by # symbols). Document names and sizes are stored in a dictionary. The tokens are added to their respective in-memory postings lists stored in another dictionary. To reduce memory fragmentation, the dictionaries store a copy of the token keys flattened into a set of large sequential byte arrays. Each in-memory postings list is a byte array that grows by doubling its size and encodes a header followed by a variable byte (vbyte [12]) compressed list of `{delta-id,frequency}` pairs, each indicating the increment for the next document id and the number of occurrences of the given term in that document. When the input data ends, the document names and sizes are output, followed by all pairings of tokens with their corresponding postings lists. Since each postings list is already stored compressed in a byte array, the relevant portion of that array can be written directly to the output stream. The format of the resulting *mindex* allows for the simple implementation of a *mmerge* component that combines indexes produced from separate data partitions.

The *mencode* component reads in an *mindex* file and outputs an *mindex.meta* file containing the total token sizes and a dictionary mapping tokens into the location of the corresponding postings list within the *mindex* file. The *mindex.meta* file allows the search engine to quickly start processing queries instead of having to read the entire *mindex* file at startup.

Note that by splitting the indexing into two steps, both the inversion and merge components are simplified. In addition, multiple variants of the encoding step can be implemented without changing the previous steps. For example, the vbyte encoding of the *mindex* file is compact enough for most purposes, but the encoding step could output the index to a separate file using a different compression format if needed. For faster query execution, skips over the postings lists could be added to the *mindex* or *mindex.meta* files.

[7] https://github.com/andrewrkane/mtextsearch

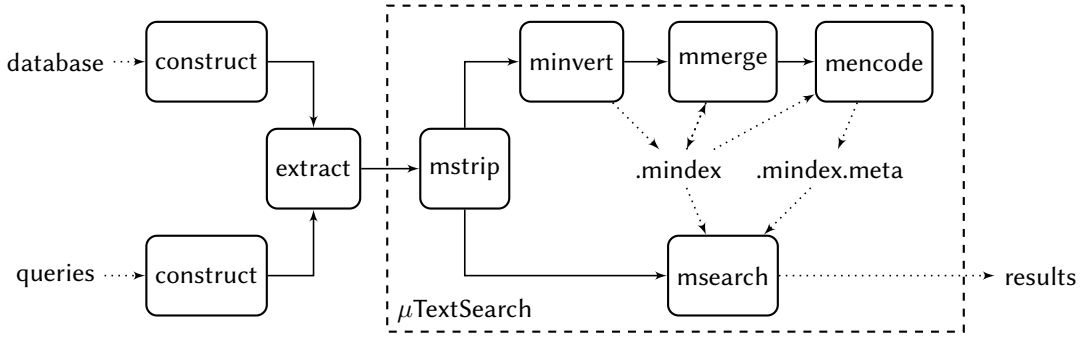

Figure 5: Indexing and querying pipelines.

The *msearch* component takes the *mindex* and *mindex.meta* files on the command line and loads the meta information into memory. Queries are read from *stdin*, one per line, then the postings lists' disk locations are found in the meta dictionary and read into memory. Next, the query is executed as an exhaustive-OR query,[8] and the top-1000 results are written to *stdout*.

In order to make the core engine "math-aware," the tokenizer code (shared between the *minvert* and *msearch* components) must recognize the encoded math tuples. Additionally, the search ranking code must be extended to support the α tuning parameter described in Section 4.3. Finally, to participate in the Lab, the query results must be converted to the specified formats.

4.2. Indexing and Querying Pipelines

Our system comprises several series of processing steps that can be piped together, connecting *stdout* from one step to *stdin* for the next. As a result, the output of any step can also be easily saved as a file for later reuse. The intermediate files passed from step to step can also be compressed using *gzip* and piped to the next step using *gunzip -c*. The separation of our pipelines into steps allows some parallel processing, since each step is run in a separate operating system process. In addition, if the data is extracted in separate partitions, for example by year, then those partitions can be processed in parallel for many steps, then joined back together later in the pipeline.

Recall from Section 2 that documents are enclosed by TREC-like tags. With this convention in mind, the processing steps for indexing, as depicted in Figure 5, are as follows:

1. construct documents : ARQMath data files $\rightarrow$ trec(html+mathml)
2. extract math features : trec(html+mathml) $\rightarrow$ trec(html+mathtuples-mathml)
3. core engine indexing
   - 3.1. *mstrip* unused $_{\text{(tags,comments)}}$: trec(html+mathtuples) $\rightarrow$ trec(text+mathtuples)
   - 3.2. *minvert* and compress : trec(text+mathtuples) $\rightarrow$ .mindex file
   - 3.3. *mmerge* : (.mindex file)$^{+}$ $\rightarrow$ .mindex file
   - 3.4. *mencode* metadata : .mindex file $\rightarrow$ .mindex.meta file

[8] that is, scoring all documents found on all inverted lists associated with query terms

Our query pipeline follows similar steps to indexing. As depicted in Figure 5, the processing steps for searching are as follows:

1. construct queries : ARQMath query files → xml(qid+text+mathml)
2. extract math features : xml(qid+text+mathml) → xml(qid+text+mathtuples-mathml)
3. core engine searching
   3.1. *mstrip* unused $_{\text{(tags,comments)}}$: xml(qid+text+mathtuples) → (qid+text+mathtuples)*
   3.2. *msearch* : (qid+text+mathtuples)* →$_{\text{.mindex + .mindex.meta}}$ queryresults

4.3. Ranking

Unlike Tangent-L which uses BM25+ [6], the *msearch* engine ranks documents using standard BM25 [13]. It has been shown elsewhere that this difference does not likely lead to significant differences in ranking effectiveness [14].

Given a collection of documents D containing N documents and a query q consisting of a set of query terms, the BM25 score for a document $d \in D$ is defined as the sum of scores for each query term as follows:

$$\text{BM25}(q,d) = \sum_{t \in q} \ln\left(\frac{N - df_t + 0.5}{df_t + 0.5} + 1\right) \cdot \frac{tf_{td} \cdot (k_1 + 1)}{tf_{td} + k_1 \cdot \left(1 - b + b \cdot \left(\frac{L_d}{L_{avg}}\right)\right)} \tag{1}$$

where the log factor is the Inverse-Document-Frequency component with df_t being the document frequency for t, the number of documents in D containing term t; and the other factor is the Term-Frequency component with tf_{td} being the term frequency of t in document d; L_d being the document length; L_{avg} being the average document length; and k_1 and b are constants chosen to be 1.2 and 0.75, respectively, in the absence of specific data training.

Because the scoring function is a sum over the query terms, it can be split into two sums, one for math terms and the other for text terms, and a weighting parameter α can be used to determine how heavily to rely on matching math terms:

$$\text{BM25}_{\text{w}}(q_m \cup q_t, d) = \alpha \cdot \text{BM25}(q_m, d) + (1 - \alpha) \cdot \text{BM25}(q_t, d) \tag{2}$$

where q_m is the set of math terms in a query q, and q_t is the set of text terms in q. It turns out that setting an appropriate value for α is critical to retrieval effectiveness. Note: the α used here covers *all* math tuples, as compared to our work from the previous year which split some math tuples into a separate tuning parameter.

4.4. Calculating Document Lengths

The calculation of BM25 depends on relative document length in the denominator of the term frequency factor. The rank should reflect the unexpectedness of seeing tf_{td} instances of term t in document d, and the inclusion of $\frac{L_d}{L_{avg}}$ is intended to reflect the fact that a longer document can be expected to have a higher term frequency.

In standard search engines, the length of a document D is measured by the number of terms in D. However, when there are two types of terms (math tuples and text tokens) and the formula

is split into two sums as in Equation 2, should the number of text terms influence the likelihood of seeing tf_{td} instances of a math tuple and the number of math tuples influence the likelihood of seeing tf_{td} instances of a text term? Better effectiveness might result from using the number of math tuples in D for the document length (and average document length) when calculating BM25 for q_m and from using the number of text terms in D when calculating BM25 for q_t.

Disappointingly, this change does not seem to give improved ranking performance with either the ARQMath-1 or the ARQMath-2 benchmarks.

5. Experimental Results

5.1. Conventions for Naming MathDowser Runs in ARQMath-3

We refer to experimental runs using the following naming convention that reflects the parameter settings used.

Li: All default values for the math tuple generator are used (Table 1), except that location tuples are created for symbol pairs, terminal symbols, compound symbols, and duplicate symbols for paths having fewer than i nodes. Thus "L1" has no location tuples, "L8" uses *all* default values, and "L99" includes augmented location tuples for all symbol pairs, terminal symbols, compound symbols, and duplicate symbols.

Li(X_1=j_1,...,X_n=j_n): Settings are like Li, but with argument X_k specified with value j_k. For example, "L8(D=0)" is like "L8" but with no duplicate symbols (i.e., `-D 0`) and "L99(a=d,s)" is like "L99" but with anchoring on equality operators disabled and all wildcard synonyms included (i.e., `-a d -s`).

For some experiments, all queries are executed against an index that was created with different parameters. In this case, the name of the run specifies the query parameters, followed by "on" and the parameters used for indexing. For example, "L1on8" indicates a run in which queries use parameters for "L1" against an index that was built using parameters for "L8," and "L99(a=d)on99(a=d,s)" indicates a query with parameters for "L99(a=d)" run against an index built with parameters "L99(a=d,s)."

Note that experiments run in the ARQMath-2 Lab with Tangent-L [7] use a configuration that is close to, but not identical to, "L99(a=d,w=)on99(a=d,s)": all four types of tuples are augmented with arbitrarily long locations, locations are anchored at the root only, duplicate tuples include the symbol that is duplicated but not the wildcard equivalents, and the index additionally includes all available wildcard variants (even though these are never used in ARQMath queries). Tangent-L also includes all possible pairings when generating tuples for duplicate symbols and creates tuples that attempt to account for commutative operations and symmetric relations,[9] but these options have not been made available through the math tuple generator.

In addition to setting parameters for the math tuple generator, a run must set the value of α to specify the relative weight of math tuples vs. text terms (Equation 2). This is indicated in the

[9]The superficial approach for handling commutative operations and symmetric relations considered last year has negligible effect on performance and is therefore deemed not worth re-implementing.

run's name by appending "**_a0**ij" for $\alpha = 0.ij$. For example, "L8_a018" indicates a run using "L8" for generating math tuples and $\alpha = 0.18$.

5.2. Task 1: Finding Answers to Math Questions

For the main task, participants are given mathematical questions selected from MSE posts from year 2019 (for ARQMath-1), year 2020 (for ARQMath-2), or year 2021 (for ARQMath-3). Each question is formatted as a *topic* that contains a unique identifier, the title, the question body text, and the tags. Participant systems should return the top-1000 potential answer-posts from the MSE collection for each of the topics.

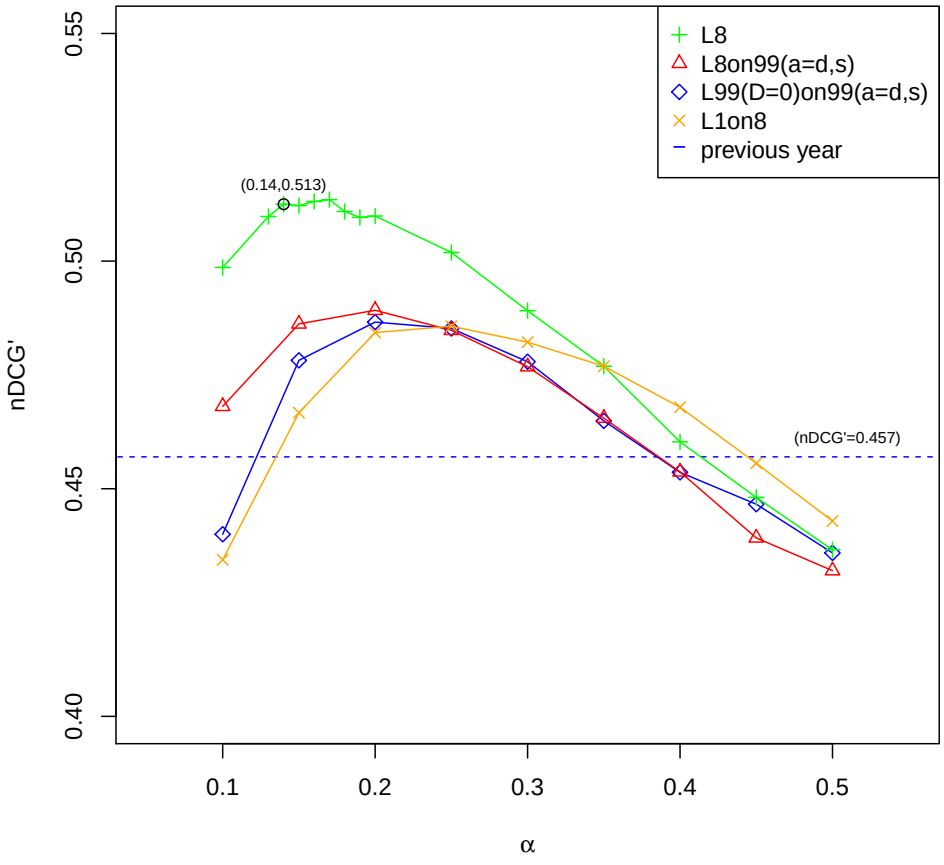

Figure 6: Effect of α on performance for ARQMath-1 Task 1 benchmark.

Experiments with the ARQMath-1 and ARQMath-2 benchmarks show that (1) when using a linear subset of tuples for duplicated nodes, storing both the form with the symbol that is duplicated and the form with the wildcard equivalent for that symbol gives the best performance;

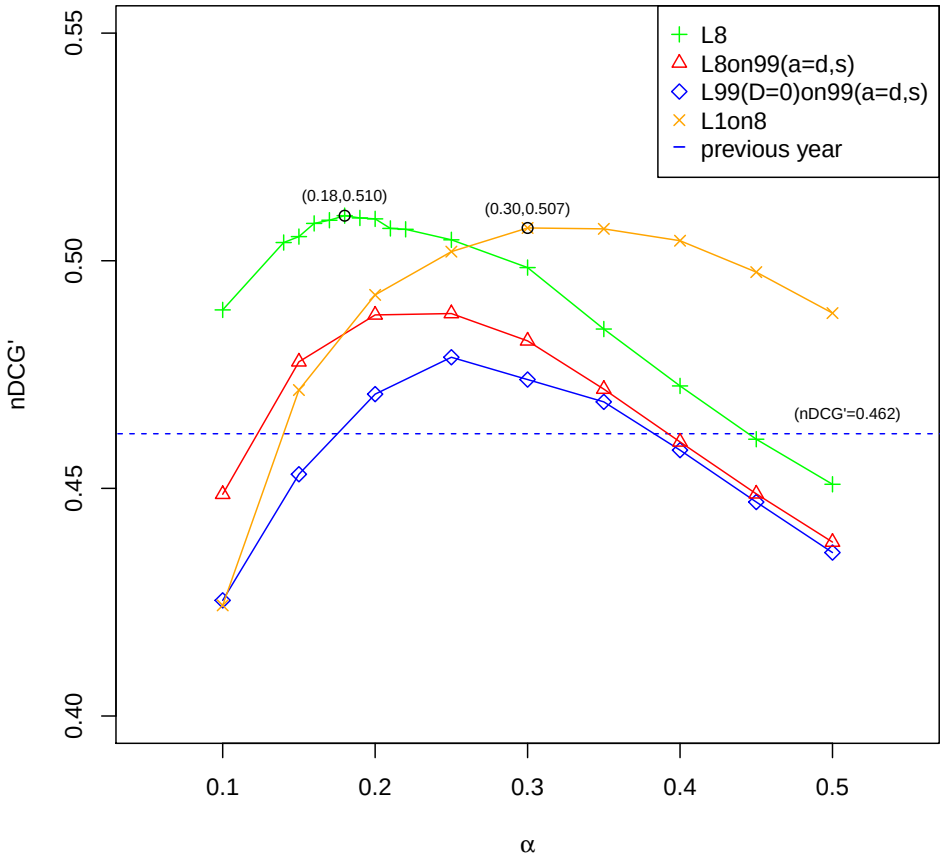

Figure 7: Effect of α on performance for ARQMath-2 Task 1 benchmark.

and (2) limiting the augmented locations for symbol pairs, terminal symbols, compound symbols, and duplicate symbols to having fewer than 8 nodes on the locating path outperforms other settings for locations. Figures 6 and 7 show the results for four of the experiments. (For comparison, the nDCG' value of the best performing configuration from 2021 is also included.)

It is apparent from these figures that the value of α has a substantial effect on the average nDCG' (and, although not illustrated here, on the average MAP' and P'@10 measures as well). Examination of the effect of α on individual queries reveals that larger values are better for some queries and smaller values are better for others. However, as in the past [8, 15], we could find no correlation between the best value for α and the number of formulas in a query, the relative number of text terms in a query, the sizes of query formulas, the frequency of occurrence of math tokens or their inverse document frequencies, or any other query characteristics. For now, we have no better approach than choosing a value for α heuristically, based on observed

average performance on some training data.

Because re-indexing the corpus is quite time-consuming, our initial experiments varied the location length cutoff at query time, but ran against an index storing all locations. When we found that using a cutoff value of 8 improves performance, we created an index with that setting and found that running queries with cutoff 8 against an index with cutoff 8 (L8) outperformed running queries with cutoff 8 against an index with all locations stored (L8on99). However, we found that running queries with cutoff 1 against an index with cutoff 1 (L1) performed *worse* than running those queries against an index with cutoff 8 (L1on8)!

We are surprised that using an index containing tuples that are never queried can substantially alter ranking performance.[10] Looking at the BM25 ranking formula, the only explanation can be that the somewhat increased document sizes and average document size in the larger index have a noticeable effect on ranking scores. This portion of the ranking algorithm is also affected by the k_1 and b constants (Equation 1), where b balances the document length normalization. Scoring matches instead with Anserini's default values (namely, $k_1 = 0.9, b = 0.4$)[11] [17] produces similar performance for ARQMath-1, slightly better performance for ARQMath-2, and slightly worse performance for ARQMath-3. It would seem that, in spite of the disappointing results reported in Section 4.4, more experimentation regarding how to calculate document length and the BM25 constants is justified.

Figure 6 indicates that using L8 with $\alpha \in [0.13, 0.20]$ might be suitable for ARQMath-3: on the ARQMath-1 benchmark, these achieve improvements of at least 0.05 over last year's best nDCG' value. Figure 7 indicates that "L8_a018" may be a good configuration with an improvement near 0.05 over last year's best nDCG' value for the ARQMath-2 benchmark. To vary our submitted runs, we also include "L8_a014" to cover the range indicated by the ARQMath-1 benchmark.

Surprisingly, "L1on8" performs almost as well on the ARQMath-2 benchmark, even though it does not perform very well on the ARQMath-1 benchmark. In case the ARQMath-3 benchmark is more similar to the latter than it is to the former, we include "L1on8_a030" as another alternative.

In summary, for ARQMath-3, we prepared three automatic runs:

L8_a018: The primary submitted run with all default values for the math tuple generator (including augmented location tuples for paths having fewer than 8 nodes and $\alpha = 0.18$;

L8_a014: An alternate run that uses the same setup as the primary run, but with $\alpha = 0.14$; and

L1on8_a030: An alternate run that *indexes* the corpus using the default values (including location tuples for paths having fewer than 8 nodes), but *searches* it with no location tuples generated for the queries (i.e., "L1" for querying but "L8" for the index). For this configuration, α is set to 0.30.

The results of these runs for all three years are shown in Table 4. In general, after parameter selection based on the ARQMath-1 and ARQMath-2 benchmarks, our updated system produces

[10] This may be a similar effect to changes in ranking performance observed by others after text cleaning, i.e., removing various forms of non-content-related markup [16].

[11] These are also the values used to compare BM25 implementations [14].

Table 4
Task 1: Evaluation of the MathDowsers runs in three years' ARQMath Labs and the best runs for other participants in ARQMath-3. Italics indicate MathDowser runs for which parameters were tuned on the same data.

	ARQMath-1 (77 Topics)			*ARQMath-2 (71 Topics)*			*ARQMath-3 (78 Topics)*		
	nDCG′	*MAP′*†	*P′@10*†	*nDCG′*	*MAP′*†	*P′@10*†	*nDCG′*	*MAP′*†	*P′@10*†
MathDowsers									
¶L8_a018	0.511	0.261	0.307	*0.510*	0.223	0.265	0.474	0.164	0.247
∗L8_a014	*0.513*	0.257	0.313	0.504	0.220	0.265	0.468	0.155	0.237
∗L1on8_a030	0.482	0.241	0.281	*0.507*	0.224	0.282	0.467	0.159	0.236
MathDowsers (year 2021)									
duplicateTerms	*0.457*	0.207	0.267	0.462	0.187	0.241	0.447	0.159	0.236
¶primary	*0.433*	0.191	0.249	0.434	0.169	0.211	-	-	-
holisticSearch	*0.405*	0.192	0.266	0.414	0.167	0.225	-	-	-
∗proximityReRank	*0.373*	0.117	0.131	0.335	0.081	0.049	-	-	-
MathDowsers (year 2020)									
∗alpha05-noR	0.345	0.139	0.162	-	-	-	-	-	-
∗alpha02	0.301	0.069	0.075	-	-	-	-	-	-
∗alpha05-trans 𝕄	0.298	0.074	0.079	-	-	-	-	-	-
¶alpha05	0.278	0.063	0.073	-	-	-	-	-	-
∗alpha10	0.267	0.063	0.079	-	-	-	-	-	-
Best runs for each other participating team (year 2022)									
¶Approach0 𝕄	0.462	0.244	0.321	0.460	0.226	0.296	**0.514**	**0.219**	**0.349**
¶MSM	0.422	0.172	0.197	0.381	0.119	0.152	0.504	0.157	0.241
¶MIRMU	0.466	0.246	0.339	0.487	0.233	0.316	0.498	0.184	0.267
¶TU-DBS	0.446	0.268	0.392	0.454	0.228	0.321	0.436	0.158	0.263
¶DPRL	0.508	0.467	0.604	0.533	0.460	0.596	0.283	0.067	0.101
∗DPRL	**0.587**	**0.519**	**0.625**	**0.582**	**0.490**	**0.618**	0.278	0.055	0.022
¶SCM	0.254	0.102	0.182	0.197	0.059	0.149	0.257	0.060	0.119
Hybrids of primary runs									
Approach0^{400}+MSM 𝕄	-	-	-	-	-	-	**0.594**	**0.234**	**0.345**
MIRMU^{500}+L8_a018	-	-	-	-	-	-	0.554	0.201	0.267

¶ submitted primary run ∗ submitted alternate run 𝕄 manual run † using H+M binarization

results that have a statistically significant (p<.001)[12] improvement in nDCG', and substantially improved results for MAP' and P'@10 compared with those from last year's system over the previous two years' topics.

Although the performance of the system is nearly identical for ARQMath-1 and ARQMath-2, it apparently deteriorates slightly for all three configurations and with respect to all three effectiveness measures when used with the ARQMath-3 benchmark. Nevertheless, all three

[12]Throughout this section, statistical significance is determined by a paired t-test comparing the nDCG' scores for individual queries, assuming those scores are normally distributed. The p value indicates the likelihood that there is no difference in the average value over all possible queries.

runs significantly outperform a run using last year's Tangent-L configuration (*duplicateTerms*, but run on this year's corpus), with p=.002, p=.018, and p=.016, respectively. The changes introduced this year are indeed effective in improving ranking performance.

The apparent deterioration in performance might be caused by the parameter values being overfit to the training data. However, an examination of the setting of α for L8 and L1on8 shows performance curves shaped essentially like those in Figure 7, but with uniformly lower nDCG' values. We conclude that the choice of α is appropriate (at or near the peak performance), and so not overfit to the training data.

Comparing to other systems, we find that Approach0, which outperformed our system on Task 2 last year, also outperforms our system on Task 1 this year, as do runs from MSM and MIRMU. The differences in nDCG' scores (as compared to L8_a018) on the ARQMath-3 benchmark are statistically significant for Approach0 (p=.029) and for MSM (p=.037), but not for MIRMU (p=.103). Similarly, the difference in performance with respect to TU-DBS is statistically significant (p=.031).

The nDCG' scores vary substantially from topic to topic (with L8_a018 having a low of 0.028 for Topic A.327, a high of 0.781 for topic A.325, a mean of 0.474, and standard deviation of 0.165). With a closer look at the effectiveness breakdown by topic category in Table 5, we observe that, in spite of a different set of math topics being evaluated, the observed strengths and weaknesses are not much different from those of our best participant runs from previous years [4, 7], where we found that *Text*, *Both*, *Concept*, and *Medium* queries lagged the average nDCG' performance. Nevertheless, it appears that the system's performance substantially improves for topics that depend primarily on text and for topics concerned with mathematics of medium difficulty, and that it performs very well on topics fitting the new category *Proof and Concept*. A possible hypothesis that the decrease in performance is due to the increase in topics of medium difficulty (now 43 of 78 judged queries) is thus unjustified. There is a moderate correlation (with Pearson coefficient of 0.573), however, between the number of judged answers among the top-1000 results from L8_a018 and nDCG'; perhaps the scores would be higher if more of our submitted answers had been judged.

Table 4 shows that on the ARQMath-2 benchmark, some systems outperform others with respect to P'@10 yet have poorer nDCG' ratings. In light of the fact that ensemble runs often perform better than their constituent systems [18], this leads us to propose an alternative ensemble scheme: combine two runs by taking the top-k from the first run and the remainder from the second run (with duplicates removed and results limited to 1000). When exploring this simple approach on the primary runs of the different teams, we note that the nDCG' values improve substantially with k values as high as 500. Two example hybrid runs are presented at the bottom of Table 4, giving the hybrid configurations with the best nDCG' scores from our exploration containing either a manual run or two automatic runs. We acknowledge that these particular hybrid configurations may be overfit to the ARQMath-3 benchmark, but note there are improvements from all combinations we explored between group runs. There seems to be no substantial gain from combining two of our own runs, so the inter-team nDCG' improvement implies that the various teams' runs include many non-overlapping relevant results. Continued exploration of ensemble methods is certainly warranted.

Table 5
Category performance of the L8_a018 run in ARQMath-3. The best performance measure for each sub-category and each evaluation measure is highlighted in bold.

	Topic Count	*L8_a018* *nDCG′*	*MAP′*	*P′@10*
Overall	78	0.474	0.164	0.247
Dependency				
Text	10	**0.544**	0.144	0.190
Formula	22	0.516	**0.231**	**0.354**
Both	46	0.439	0.137	0.209
Topic Type				
Computation	21	0.481	**0.202**	**0.286**
Concept	16	0.491	0.151	0.244
Proof	36	0.455	0.147	0.222
Proof and Concept	5	**0.536**	0.175	0.280
Difficulty				
Low	17	**0.489**	**0.226**	**0.312**
Medium	43	0.472	0.147	0.233
Hard	18	0.467	0.149	0.222

5.3. Task 2: In-context Formula Retrieval

For Task 2, participants are asked to retrieve the top matching formulas, together with their associated posts, for each *formula query* chosen from the set of topics used for Task 1. As in 2021, the relevance of a retrieved formula is evaluated in context: both the associated post of a retrieved formula and the associated topic content of the formula query are presented to the assessors for evaluation. Assessments are then aggregated so that each *visually distinct* formula is assigned the maximum of the relevance scores of the instances for that formula in context. Thus, if any instance is judged to be relevant, the formula is deemed to be relevant: the performance of a system is determined by its performance with respect to visually distinct formulas only.

For ARQMath-3, we include three automatic runs:

L8: The primary submitted run over a corpus of visually distinct formulas only (thus α is not needed, as there are no text terms), with the default settings for the math tuple generator; for each matching visually distinct formula, we simply return the first formula_id and post_id we encounter for that formula during our extraction;

L8_a035: An alternate run with the same settings as the primary run, but running over a corpus of visually distinct formulas augmented with text extracted from the LaTeX encoding and $\alpha = 0.35$, the value that maximized nDCG' for ARQMath-1; and

L8_a040: An alternate run with the same setup as above, but with $\alpha = 0.40$, the value that maximized nDCG' for ARQMath-2.

Table 6
Task 2: Evaluation of MathDowsers runs, the best participant runs, and baseline runs. Italics indicate MathDowser runs for which parameters were tuned on the same data.

	ARQMath-1			*ARQMath-2*			*ARQMath-3*		
	nDCG′	*MAP′*†	*P′@10*†	*nDCG′*	*MAP′*†	*P′@10*†	*nDCG′*	*MAP′*†	*P′@10*†
Baseline									
Tangent-S	0.691	0.446	0.453	0.492	0.272	0.419	0.540	0.336	0.511
MathDowsers									
L8_a040 (cleaned) 𝕄	-	-	-	-	-	-	0.682	0.485	0.601
L8_a035 (cleaned) 𝕄	-	-	-	-	-	-	0.681	0.485	0.601
L8 (cleaned) 𝕄	-	-	-	-	-	-	0.672	0.477	0.601
∗L8_a040	0.657	0.460	0.516	*0.624*	0.412	0.524	0.640	0.451	0.549
∗L8_a035	*0.659*	0.461	0.516	0.619	0.410	0.522	0.640	0.450	0.549
¶L8	0.646	0.454	0.509	0.617	0.409	0.510	0.633	0.445	0.549
MathDowsers (year 2021)									
¶formulaBase	0.562	0.370	0.447	0.552	0.333	0.450	-	-	-
∗docBase	0.404	0.251	0.386	0.433	0.257	0.359	-	-	-
Best runs for each other participating team (year 2022)									
¶Approach0 𝕄	0.647	0.507	0.529	**0.652**	**0.471**	**0.612**	**0.720**	**0.568**	**0.688**
¶DPRL	0.648	0.480	0.502	0.569	0.368	0.541	0.694	0.480	0.611
∗DPRL	0.733	**0.532**	0.518	0.550	0.333	0.491	0.575	0.377	0.566
∗XYPhoc	0.492	0.316	0.433	0.448	0.250	0.435	0.472	0.309	0.563
¶JU_NITS	0.238	0.151	0.208	0.178	0.078	0.221	0.161	0.059	0.125
Best participant run (year 2021)									
∗Approach0 𝕄	0.507	0.342	0.441	0.555	0.361	0.488	-	-	-
∗DPRL	**0.738**	0.525	**0.542**	0.445	0.216	0.333	-	-	-
Best participant run (year 2020)									
∗DPRL	0.563	0.388	0.436	-	-	-	-	-	-

¶ submitted primary run ∗ submitted alternate run 𝕄 manual run † using H+M binarization

The results of all three runs over the three years' benchmarks are shown in Table 6, together with the baseline run, the best participant runs during the previous years' Labs, and the best run from each team for ARQMath-3. The differences in the nDCG' values among the best runs from each participating team are all statistically significant ($p<.05$), except that the difference between the best run from Approach0 and that from DPRL is not statistically significant ($p=.106$).

The three unsubmitted MathDowser runs, annotated as "cleaned," manually correct an oversight in processing the ARQMath-3 queries, where several of the SLT formulas include invalid MathML (namely, instances of &, <, and > symbols that are not escaped). For all three configurations, performance is improved after cleaning: not enough to change the rankings, but the differences in performance of L8_a040 (cleaned) with respect to DPRL and Approach0 are no longer statistically significant ($p=.243$ and $p=.053$, respectively).

The improvements to the selection of math tuples this year results in L8 achieving better

scores on the ARQMath-1 and ARQMath-2 benchmarks than any of last year's systems, including Tangent-L's formulaBase approach which similarly searches a corpus of visually distinct formulas. Comparing the three MathDowser runs, however, shows that including text from the LaTeX encoding of the formulas further improves performance, with statistically significant differences (p=.009 for latex_L8_a040 and p=.026 for latex_L8_a035). The success of the Approach0 team, which augments query formulas with manually chosen text terms, supports the hypothesis that augmenting math tuples with suitably chosen text is beneficial. We recommend exploring the inclusion of text terms that appear within formulas for Task 1, where some benefit might also accrue from cross-matching extracted text with terms found elsewhere in the documents.

It is possible that the approach of arbitrarily choosing a single instance of each formula to be assessed is overly naive. Perhaps choosing candidate instances for assessment based on matching text, or even just presenting additional candidate instances chosen at random, would have resulted in more favourable assessments for the formulas matched by our system and thus higher scores. Unfortunately, this hypothesis can only be tested by performing additional assessments.

5.4. Efficiency

All experiments were run on a Mac OSX 10.11.6 laptop with an Intel Core i7-4770HQ Processor (4 Cores 8 Threads, 2.2GHz up to 3.4Ghz) and 16GB RAM with a 256GB flash disk.

Preparing the data for Task 1 using the L8 configuration is quite time-consuming, but the indexing itself is relatively efficient, as shown in the following table:

Step	Processing	See Section	Indexing Speed (sec)
1.	Construct - generate corpus	2.1	46581
2.	Extract math tuples	3.4	15272
3a.	mstrip	4.1	2376
3b.	minvert	4.1	4176
3c.	mencode	4.1	154

Clearly, many steps in our indexing pipeline can take a long time to run. However, the data can be partitioned so that most individual steps can be split and run in parallel. In addition, multiple steps can run in parallel, each as a separate process. For the experiments presented in this paper, we used the same constructed output stored in compressed files, so step 1 needs to be re-run only when the data itself changes.

The corpus construction steps for Task 1 output 15.9GB of data, which compresses down to 1.6GB using *gzip*. The resultant index size includes 1.9GB output from *minvert*, plus an additional 174MB of metadata output from *mencode*. This encoding step allows the *msearch* process to load the index information in less than one second. Currently the *msearch* exhaustive-OR query execution is slow: to run all 298 queries from the three ARQMath years on this L8 index requires 3326 seconds giving an average of 11 seconds per query.

6. Conclusions and Further Work

Replacing Tangent-L by a new search engine proved to be fairly straightforward, and the particular search engine we used proved itself to be effective. The indexing and query pipelines that we describe in this paper can be used with any search engine to make it "math-aware" and suitable for addressing both Task 1 and Task 2. Furthermore, using compressed data between each step of the indexing pipeline is a straightforward technique that greatly improves the efficiency of experiments when intermediate results are reused. There are also several opportunities to improve the engine's efficiency, including more compact representations for postings lists, smarter algorithms for combining postings lists, and incorporation of parallel and distributed engine technology.

Comparing this year's results to experiments from previous years shows that trying to match too many features for each formula can be detrimental to effectiveness, as well as wasting space in the index and processing time to make those matches. Furthermore, we have shown that the choice of the tuning parameter α to balance the weight of math tuples against conventional text terms is crucial to effectiveness, and the optimal value depends on parameter settings used for generating tuples. Unfortunately, determining which characteristics of the data and the queries affect the optimal value of α remains somewhat of a mystery, and thus further work is required to understand the dependency.

We believe that we have now pushed the traditional IR approach as far as we can, and that substantial gains in effectiveness will require new techniques rather than merely better tuning of the parameters we have introduced. Because α is tuned based on experiments using earlier benchmarks, choosing an appropriate value tailored to each query may be a perfect opportunity to apply machine learning to improve math-aware search systems. Ensemble methods to combine the results from several diverse approaches also holds much promise.

Acknowledgments

The NTCIR Math-IR dataset used as a source of relevant keywords when translating topics to queries for Task 1 [4] was made available through an agreement with the National Institute of Informatics. We thank the anonymous ARQMath participant reviewers for their constructive comments, which helped us to improve the explanations in this paper.

References

[1] B. Mansouri, V. Novotný, A. Agarwal, D. W. Oard, R. Zanibbi, Overview of ARQMath-3 (2022): Third CLEF lab on Answer Retrieval for Questions on Math, in: CLEF 2022, volume 13390 of *LNCS*, Springer, 2022.

[2] R. Zanibbi, D. W. Oard, A. Agarwal, B. Mansouri, Overview of ARQMath 2020 (updated working notes version): CLEF lab on answer retrieval for questions on math, in: CLEF 2020, volume 2696 of *CEUR Workshop Proceedings*, 2020.

[3] B. Mansouri, R. Zanibbi, D. W. Oard, A. Agarwal, Overview of ARQMath-2 (2021): second

CLEF lab on answer retrieval for questions on math, in: CLEF 2021, volume 12880 of *LNCS*, Springer, 2021, pp. 215–238.
[4] Y. K. Ng, D. J. Fraser, B. Kassaie, F. W. Tompa, Dowsing for math answers, in: CLEF 2021, volume 12880 of *LNCS*, 2021.
[5] D. J. Fraser, A. Kane, F. W. Tompa, Choosing math features for BM25 ranking with Tangent-L, in: DocEng 2018, 2018, pp. 17:1–17:10.
[6] Y. Lv, C. Zhai, Lower-bounding term frequency normalization, in: CIKM'11, 2011, pp. 7–16.
[7] Y. K. Ng, D. J. Fraser, B. Kassaie, F. W. Tompa, Dowsing for math answers to math questions: Ongoing viability of traditional MathIR, in: CLEF 2021, volume 2936 of *CEUR Workshop Proceedings*, 2021.
[8] Y. K. Ng, Dowsing for Math Answers: Exploring MathCQA with a Math-aware Search Engine, Master's thesis, University of Waterloo, Cheriton School of Computer Science, 2021.
[9] S. Büttcher, C. L. A. Clarke, G. V. Cormack, Information Retrieval - Implementing and Evaluating Search Engines, MIT Press, 2010.
[10] D. Stalnaker, Math expression retrieval using symbol pairs in layout trees, Master's thesis, Rochester Institute of Technology, Golisano College of Computer and Information Sciences, 2013.
[11] M. F. Porter, An algorithm for suffix stripping, Program: Electron. Lib. and Infor. Sys. 14 (1980) 130–137.
[12] H. E. Williams, J. Zobel, Compressing integers for fast file access, Comput. J. 42 (1999) 193–201.
[13] S. Robertson, H. Zaragoza, The probabilistic relevance framework: BM25 and beyond, Foundations and Trends in Information Retrieval 3 (2009) 333–389.
[14] C. Kamphuis, A. P. de Vries, L. Boytsov, J. Lin, Which BM25 do you mean? A large-scale reproducibility study of scoring variants, in: ECIR 2020, Part II, volume 12036 of *Lecture Notes in Computer Science*, Springer, 2020, pp. 28–34.
[15] D. J. Fraser, Math Information Retrieval using a Text Search Engine, Master's thesis, University of Waterloo, Cheriton School of Computer Science, 2018.
[16] D. Roy, M. Mitra, D. Ganguly, To clean or not to clean: Document preprocessing and reproducibility, ACM J. Data Inf. Qual. 10 (2018) 18:1–18:25.
[17] P. Yang, H. Fang, J. Lin, Anserini: Enabling the use of lucene for information retrieval research, in: SIGIR 2017, ACM, 2017, pp. 1253–1256.
[18] V. Novotný, P. Sojka, M. Štefánik, D. Lupták, Three is Better than One Ensembling Math Information Retrieval Systems, in: CLEF 2020, volume 2696 of *CEUR Workshop Proceedings*, 2020.