Tangent-3 at the NTCIR-12 MathIR Task

Kenny Davila
Richard Zanibbi
Rochester Institute of Technology, USA
{kxd7282,rxzvcs}@rit.edu

Andrew Kane
Frank Wm. Tompa
University of Waterloo, Canada
{arkane,fwtompa}@uwaterloo.ca

ABSTRACT

We present the math-aware search engine Tangent-3 and report its results for the NTCIR-12 MathIR task. Tangent uses a federated search over two indices: 1) a TF-IDF textual search engine (Solr), and 2) a query-by-expression engine. We use an inverted index to store math expressions using pairs of symbols extracted from a Symbol Layout Tree representation built from Presentation MathML. We use a cascade model with two stages for retrieval. In the first stage, relevant expressions are retrieved quickly using iterator trees over posting lists to find matches and expressions are ranked using the Dice coefficient of matched symbol pairs. In the second stage, the top-k best candidates are reranked with a more strict similarity metric supporting unification and wildcard matching. Our system produces relevant (and partially relevant) Precision@5 values of 21% (50%) for the main arXiv task, 25% (49%) for the Main Wikipedia subtask and 45% (84%) for the Wikipedia Formula Browsing subtask.

Team Name

RITUW (Rochester Institute of Technology, USA and University of Waterloo, Canada)

Subtasks

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

Keywords

mathematical information retrieval, reranking, MathML, unification, wildcard matching, symbol layout tree

1. INTRODUCTION

Large collections of data exist which are full of mathematical notation, and the task of finding relevant documents in such collections based on their mathematical content is a difficult one. Multiple approaches have being proposed in the field of Mathematical Information Retrieval (MathIR) with the intent of producing scalable solutions that facilitate search of math-related content on these large datasets.

The NTCIR-12 MathIR competition [17] provides a common ground to evaluate and compare different approaches using medium and large datasets and fixed sets of queries. The MathIR arXiv Main Task at NTCIR-12 consists of finding the most relevant documents from the arXiv[1] collection for a set of 29 queries containing keywords and math expressions. The optional MathIR Wikipedia Task includes 30 queries containing keywords and math expressions, and the optional MathIR Wikipedia Formula Browsing Task includes 40 queries for isolated math expressions. Participating teams submitted up to four runs for each task, and the top-20 results from submissions for each query received relevance assessments from evaluators.

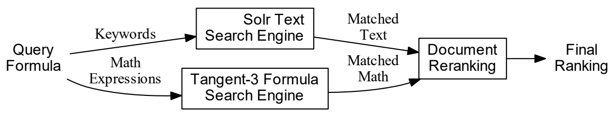

Figure 1: Tangent-3 Retrieval Model.

Our submission is based on the Tangent-3 system [19]. Our approach uses two indices, the first for text using Solr[2], and the second for math expressions using pairs of symbols and their relationships in Symbol Layout Trees (SLTs) [19] built from a Presentation MathML representation. For each query, the system processes the keywords and math expressions separately to find relevant documents for each, assigning relevance scores to each document where a match is found (see Figure 1). The math expression search is two-staged, where the first stage finds the top-k highest scored expressions using the Dice coefficient of matched symbol pairs, and then the second stage applies a more detailed similarity metric between the query expression and each of the top-k candidates (See Figure 2).

Once text and math expression search are complete, a combination step is required to produce a final ranking, where documents that contain matches for both keywords and math expressions will be ranked higher than documents containing matches for just text or math.

Most queries in the arXiv main task contain multiple keywords and at least one math expression. Our current model uses a linear combination of relevance scores for matched text and math expressions. For text, traditional TF-IDF can be used to assign relevance scores to matched keywords. However, it is not clear that TF-IDF is effective for scoring math expressions. In addition, our system supports partial matching with unification and wildcards for formulae. Partial matches are scored using various similarity metrics (Section 3.2.5).

In this competition, our experiments were based on the following research questions. The first two questions are related to the distribution of weights for final document

[1]http://arxiv.org/

[2]http://lucene.apache.org/solr/

ranking: **RQ1.** How to assign weights to keywords and math expressions in a single query? and **RQ2.** Should larger math expressions have heavier weights? The last four questions are related to the similarity metric used for math expressions: **RQ3.** How good is the approximated Dice coefficient similarity metric without matching constraints? **RQ4.** What is the effect of applying matching constraints to the Dice coefficient similarity metric during re-ranking? **RQ5.** How does unification affect perceived similarity between math expressions? **RQ6.** How does the Dice Coefficient similarity metric compare to the Maximum Subtree Similarity (MSS) [19] metric?

2. RELATED WORK

Existing techniques for formula retrieval are described in this section. Depending on the primitives used for formulae representation, we have divided these techniques into *text-based*, *tree-based*, and *spectral.*

Text-Based Approaches. Math expressions are structured data typically represented as trees. Text-based approaches require linearization of math expressions in order to represent them as strings. Normalization procedures like generalization and canonicalization are applied to ensure that equivalent math expressions can be matched on their flat representation. Generalization techniques replace variable names with generic identifiers to allow matching of equivalent expressions using different variable names [9]. Canonicalization sorts the elements within a math expression based on the commutativity of certain operations to achieve a consistent representation [9,12,13,18].

Linearization masks significant amount of structure, but it allows taking advantage of optimized text search engines. Most text-based approaches use TF-IDF (term frequency-inverse document frequency) for retrieval [8,13].

Tree-Based Approaches. Trees are used to represent either formula appearance or semantics. Expressions are indexed as complete trees, along with their subtrees to support partial matching. It is possible to compress the trees by storing identical subtrees uniquely [5]. Exact trees and/or subtrees can be matched, but some methods consider similarity matching using metrics like the tree-edit distance [6]. The *substitution tree* [2] has been used to create indices for operator trees in the Math Web Search systems [3].

One method adapts TF-IDF retrieval for SLTs, using vectors of subexpressions, along with subexpressions where arguments are replaced by wildcards [7]. SLTs are modified, normalizing argument order for commutative operators and representing operator precedences. Text in the paragraphs preceding and following formulae are added to provide contextual features for improved ranking [16].

Spectral Approaches. These approaches use paths or partial subtrees rather than complete subtrees as retrieval primitives. This can improve recall through more flexible partial matching of expressions (e.g. in the SLT for x_a^2, x^2 is a subtree, but x_i^2 is not). Nguyen et al. convert operator trees to a bag of 'words' representing individual arguments and operator-argument triples [10]. A lattice is defined over generated word sets for formulae, and a breadth-first search starting from the query formula set is used to find similar formulae. Hiroya and Saito [4] use bags of paths from the root to each operator and operand in an operator tree, with an inverted index used for retrieval. The large number of possible paths from the root make this technique brittle.

The *Tangent* search engine uses *relative* positions of symbols in SLTs to create a bag of symbol pairs representation [14,19], along with extensions to represent matrix and grid structures [11]. This representation supports partial matches well, while preserving enough information to return exact matches. Formula similarity is defined by the harmonic mean for the percentage of matched pairs in the query and a candidate (i.e., *Dice's coefficient*).[3] A Dice coefficient variant incorporating symbol pair frequencies was found to perform similarly [14]. This technique combined with keyword retrieval in Solr produced the highest Precision@5 result for the NTCIR-11 math retrieval task (92%) [1].

The Tangent-3 system [19] incorporates a two-stage retrieval model with an optimized search engine based on symbol pairs sets, using the Dice coefficient similarity metric for candidate selection, and then a reranking stage using the Maximum Subtree Similarity (MSS) metric [19]. In this paper, we take advantage of the improvements in the Tangent-3 system and evaluate the effect of using various scoring metrics for MathIR (see Section 4).

3. METHODOLOGY

In this work, we use a simple model where keywords and math expressions in a query are processed separately and results are then combined to produce a final document ranking as illustrated in Figure 1.

3.1 Text Retrieval

The text retrieval component of Tangent-3 is unchanged from Tangent-2 [11]. We have used Solr which provides an inverted index and TF-IDF scoring algorithm. The text similarity score used in our system is provided by Solr's Disjunction query, which applies TF-IDF to multiple fields of a document. For a single field, Solr scores a document d for query $q = \{t1, ..., tn\}$ by:

$$s_t(q,d) = c(q,d)idn(q) \sum_{t \in q} \sqrt{freq(t,d) \cdot idf(t)^2} \cdot norm(t,d)$$

where c is the ratio of query terms (t_i) matched in the document, $idn(q) = \sqrt{\sum_{t \in q} idf(t)^2}^{-1}$ normalizes the squared idf values, $freq(t,d)$ the number of times t appears in d, and $norm(t,d)$ is the normalized number of tokens in the field (in our case, body text or title).

To emphasize title text, the score for the title field is doubled, and then the maximum of the title and body text scores is returned as the final text score.

3.2 Formula Retrieval

The Tangent-3 formula search engine [19] employs a two-stage cascading search [15] for fast retrieval and intuitive rankings (see Figure 2). Queries are parsed into a Symbol Layout Tree (SLT, see Figure 3), which is then traversed from the root, generating tuples of the form $(s_1, s_2, R, \#)$ with ancestor symbol s_1, descendant symbol s_2, edge label sequence R from s_1 to s_2, and a count $(\#)$. Two parameters control the maximum path length between symbols in tuples (the window size, w) and whether to include tuples for symbols at the end of writing lines (EOL).

[3]Given query tree T_q and candidate tree T_c with symbol pair sets F_q and F_c, Dice's coefficient of similarity is given by $\frac{2|F_q \cap F_c|}{|F_q|+|F_c|}$.

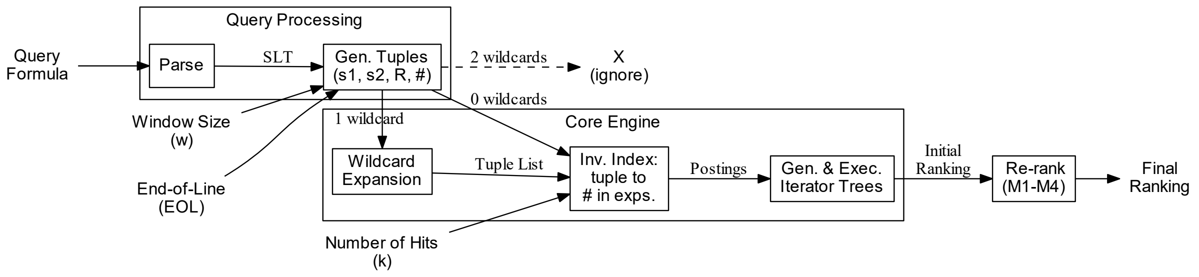

Figure 2: Tangent-3 Formula Search Engine.

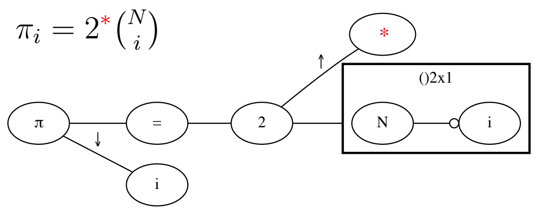

Figure 3: Query Formula with Corresponding SLT. The query has one wildcard, and a tabular structure.

After parsing, the candidate selection retrieval stage (the *core engine*) ranks a given number of expressions k by matching query tuples, using an inverted index mapping symbol pair relationships to expressions and counts (see Sections 3.2.2 and 3.2.3). Tuples with one wildcard are expanded, but tuples with two wildcards are ignored for efficiency.

The initial ranking weights matched vs. unmatched symbol pairs in the query and candidates. The second (*re-ranking*) stage finds an approximate best matching subtree (see Section 3.2.4) and re-scores candidates for the query using a detailed similarity metric (see Section 3.2.5)

3.2.1 Representation

SLTs in Tangent-3 use a unified representation of all parenthesized subexpressions regardless of their interpretation (for example, as function arguments vs. parenthesized matrices). To use greedy unification, we also add a crude representation of type. We describe these in more detail elsewhere [19].

Every node has a label, and a node's type (*number*, *variable*, *operator*, etc.) is reflected in its label. If a node's label includes an exclamation mark (e.g., V!), the type is the label prefix up to the (first) exclamation mark. Node labels starting with an asterisk (*) have type *wildcard*, and other node labels without exclamation marks have type *operator*.

Labeled edges in the SLT capture the spatial relationships between objects represented by the nodes. With respect to a given object O, nine axes reflect the following relationships: next, within, element, above, below, pre-above, pre-below, over and under. The 'element' relationship references the next element appearing in row-major order inside a structure represented by M! (matrices, tabular structures, and parenthesized expressions).

An SLT is rooted at the leftmost object on the main writing line of the formula it represents. Figure 3 shows an example of an SLT, where for simplicity, unlabeled edges represent the *next* relationship and the type prefixes are omitted. Additional details on tuple generation from this representation are available in our previous work [19].

3.2.2 Index

Indexing of math expressions is done through the core engine of Tangent-3. Since runtime performance is a high priority, the core engine uses a customized inverted index data structure implemented in C++.

The input to the indexer is a set of document names and the extracted mathematical formulae found in each document. Each formula is converted to a set of tuples that serve as index and search "terms."

At index time, an inverted index is built over the given document-formula-tuple relationships. The index includes postings lists that map each tuple to all formulae containing that tuple. In order to return document information for query results, the engine stores postings lists mapping each formula identifier to the identifiers of the documents containing those formulae, along with their positions in documents.

More details about the core engine index, its data structures and additional optimizations can be found in our previous work [19].

3.2.3 Candidate Selection

The core engine is the first retrieval stage in Tangent-3, quickly finding the top-k highly relevant matches for a formula. The engine ranks these top-k formulae using Dice's coefficient over tuples, counting the number of tuples that overlap between the query and a candidate formula using the query iterators. In addition, the engine evaluates only a subset of the query language functionality to allow the use of a fast and simple ranking algorithm that can still find a good set of candidate results. For example, the core engine supports limited wildcard functionality.

Query processing follows the architecture shown in Figure 2. First, the query is parsed into an SLT, and tuples are extracted. Then wildcard tuples are expanded, the associated postings lists for each tuple are found, iterators over these lists are created, and an iterator tree that implements the query is formed. Next, the iterator tree is advanced along formula identifiers in order, the scores are calculated, and the top-k formulae are stored in a heap. During this process, non-wildcard iterators are advanced first so that wildcard iterators only match unallocated tuples. After the iterators are finished, matching formulae and scores are returned along with the associated document names.

3.2.4 Reranking: Matching

Since our formula retrieval system supports partial matching, it requires a detailed matching procedure that involves identifying corresponding parts of the SLTs that represent a query and a candidate match. We base such a correspondence on structural equivalence of those parts. In this context, we refer to those corresponding parts as aligned SLTs.

Formally, we say that SLTs T_1 and T_2 are *aligned* if there is an isomorphism f mapping nodes from T_1 onto nodes from T_2 such that for every edge $(n_a, n_b) \in T_1$, there is a corre-

Table 1: Tangent-3 greedy wildcard expansion. Wildcards are shown with red asterisks (e.g. * or *1*). Exact matches are shown in green, wildcard matches in red and wildcard expansion anchors in blue. Symbols in black are unmatched.

Behavior	Query	Match
Unrestricted	$x + *$	$x+1$
		$x+y+z+sin(x)$
		$y+x+z=\frac{\pi}{4}$
	e^{*}	$f(x) = e^{x+1} + 2$
Restricted by children	$*^{2}+1$	$x^{2}+y^{2}+1$
		$x^{2}+y+1$
		$x^{2}+(y+z)^{2}+1$
Restricted by binding	$*1*^{2}+*1*+1$	$x^{2}+x+1$
		$(x+1)^{2}+(x+1)+1$
		$x^{2}+y+1$
Horizontal expansion (right)	$x+*+1$	$x+y+1$
		$x+y+z+1$
		$x+y-z+1$
		$x+\frac{1}{2+y}-3z+1$
Horizontal expansion (left)	$*+1$	$x+y+z+1$
		$\alpha = f(x+y+1, x^{2})$
		$f(x,y) = \frac{1}{x+y+1}$

sponding edge $(f(n_a), f(n_b)) \in T_2$ that has the same label. (Note that *node* labels in aligned trees need not match.).

In addition, approximate matches might also involve unification which in this context means simple substitutions of symbols in one SLT by alternative symbols (e.g., x for y or 3 for 2). When matching T_1 with T_2 and allowing substituted symbols, it is important that the substitutions are consistent when determining that T_1 and T_2 match approximately. We identify candidate sets of nodes in T_1 that can be consistently relabelled to match T_2.

A greedy matching algorithm is applied to find the approximately best alignment between a given query SLT T_q and a candidate SLT T_c. The algorithm tests all potential alignments between T_q and T_c and selects the alignment that has the maximum value for a given similarity metric. For each candidate alignment, the procedure applies a greedy unification algorithm that attempts to maximize the number of nodes with matching labels after substitution. In the end, the best alignment found is used to define the similarity score between query and candidate. Additional definitions, theorems and algorithms describing our basic matching procedure in detail can be found elsewhere [19].

Wildcards in queries can be expanded to match subtrees of the candidte SLT. Additional restrictions are applied to ensure that wildcards with the same name are bounded consistently to the same subtrees. The number of potential alignments between a query and a candidate grows quickly-when the number of wildcards increases because wildcards can be expanded to match subtrees of any arbitrary size.

To keep our greedy matching algorithm efficient, we only expand wildcards if certain conditions are met. For example, we always expand wildcards completely if they are located at leaf nodes of the query SLT. Otherwise, we only allow horizontal expansion which means matching elements to the right on the same baseline by following 'next' links in the SLT. We expand wildcards horizontally until we find a node in the candidate that can be matched with the 'next' child of the wildcard node in the query. We restrict horizontal expansion if the wildcard node has children other than a 'next' child. When the root of the query subtree is a wildcard and it is horizontally expandable, we treat it as a special case and allow left-side horizontal expansion to match parents of the current alignment root located on the same baseline on the candidate expressions. Examples of different types of wildcard expansions behaviors are shown in Table 1.

3.2.5 Reranking: Scoring

Because the whole SLT for an arbitrary query formula will not necessarily align with the SLT for an arbitrary candidate match formula, we need to consider subtrees of the SLTs that can be aligned. In so doing, we need to allow (but penalize) situations in which superfluous or mismatched symbols might appear in the query or in the candidate match. We wish to balance the amount of structural match with the number of symbols that are identically preserved, but also take into account the size of the "excess" unmatched structure and the extent to which wildcards engulf candidate nodes.

We suggest the following properties for a scoring function (see Figure 4): alignments with more matched symbols, and especially identical symbols, in close proximity to each other score higher than those with fewer matched symbols or more disconnected matches; if two candidates score equally with respect to matched symbols and their proximity, the one with fewer superfluous symbols scores higher; and everything else being equal, alignments with more matched symbols that are identical scores higher. Finally, alignments in which wildcards match only a few symbols score higher than alignments with large parts absorbed by wildcards.

Tangent-3 uses two different scoring functions: the Dice coefficient of pairs of symbols matched for candidate selection, and Maximum Subtree Similarity (MSS) for re-ranking [19]. Here, we describe these two metrics and add two new variations of the Dice coefficient metric that can be used for re-ranking the candidate set.

First, we define $\mathit{Matching}(T_q, T_c)$ as a function that returns the approximated maximum subtree between query tree T_q and candidate tree T_c as defined in Section 3.2.4. Let $T_m = \mathit{Matching}(T_q, T_c)$. Let E_m be the set of nodes from T_m that are exact matches. Let U_m be the set of nodes from T_m that are unified matches. Let W_m be the set of nodes from the candidate in T_m that are matched by wildcards, and let W_q be the corresponding set of wildcard nodes from the query in T_m. Note that E_m, U_m, W_m are mutually exclusive subsets of T_m. Let $EW_m = E_m \cup W_m$, $EW_q = E_m \cup W_q$, $EWU_m = E_m \cup W_m \cup U_m$ and $EWU_q = E_m \cup W_q \cup U_m$. In addition, let $\mathit{Pairs}(x, w)$ be a function that counts the number of pairs of symbols defined by a set of nodes x given the window parameter w. In this case x will be a subset of nodes from T_c.

In multiple cases, just using the main scores for some similarity metrics is not enough to distinguish certain matches. To handle these cases, a score vector is formed by adding tie-breakers. A lexicographical comparison between score vectors of two given candidates is used to determine which one ranks higher. The value W_m that represents the total number of elements in the candidate that were matched to wildcards is used as a tie-breaker in some of the functions proposed. In addition, let WDev be the standard deviation of the number of elements matched to each individual wild-

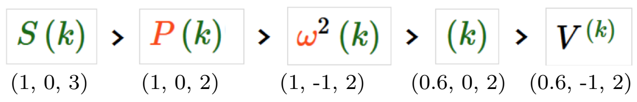

Figure 4: MSS Scoring for Query $S(k)$. Candidate formulae are ranked using a triple: 1) unified structural match, i.e., Dice coeff. of query symbol & relationship recall, 2) unmatched symbols (-ve), and 3) matched symbols identical to query symbols. Parentheses are represented as single nodes/symbols.

card in W_m. Finally, Let L_m be a score that depends on the left to right location of the root of T_m in T_c. L_m will give preference to subtrees located closer to the root of T_c.

M$_1$ - Approximated Dice Coefficient. This is the metric used by the Core Engine for selection of candidates. It is the Dice coefficient of matched pairs of symbols. The metric is considered an approximation because wildcards are not expanded beyond a single symbol, so additional symbols that could be covered by wildcards are considered unmatched by this metric. When the candidate expression and/or query expressions have a depth less than two, this metric considers additional tuples for symbols at the end of writing lines (EOL). Also, note that the core engine does not perform unification.

M$_2$ - Constrained Dice Coefficient. This is a modified version of function M_1. Unlike the original metric, this constrained version is computed after finding the largest common subtree using $Matching(T_q, T_c)$. As a result, many inconsistent tuple matches will be removed resulting in lower scores for some expressions that M_1 would rank higher. However, matching does consider wildcard expansion resulting in higher scores for some candidates that M_1 would rank lower. The unification algorithm is disabled for this metric. In addition, this metric does not require the usage of additional tuples for symbols at the end of writing lines, instead it does add a dummy pair to the count to deal with small expressions. The M_2 score is defined as:

$$R_{m2} = \frac{Pairs(EW_q, w) + 1}{Pairs(T_q, w) + 1}, \quad P_{m2} = \frac{Pairs(EW_m, w) + 1}{Pairs(T_c, w) + 1}$$

$$M_2 = \left(\frac{2R_{m2}P_{m2}}{R_{m2} + P_{m2}}, -W_m, -WDev, -L_m\right)$$

M$_3$ - Dice Coefficient with unification. Similar to M_2, this is also based on the Dice coefficient of matched triplets after matching. However, this metric considers unification during matching and scores unified matches lower than regular matches. The final M_3 is computed as follows:

$$R_{m3} = \frac{Pairs(EW_q, w) + Pairs(EWU_q, w) + 2}{2(Pairs(T_q, w) + 1)}$$

$$P_{m3} = \frac{Pairs(EW_m, w) + Pairs(EWU_m) + 2}{2(Pairs(T_c, w) + 1)}$$

$$M_3 = \left(\frac{2R_{m3}P_{m3}}{R_{m3} + P_{m3}}, -W_m, -WDev, -L_m\right)$$

M$_4$ - Maximum Subtree Similarity. This metric is the same as defined in our previous work [19]. The only difference is that our newest matching algorithm does wildcard expansion. An example of the sorting produced by this metric is given in Figure 4. The metric is defined as follows:

$$S = \begin{cases} \dfrac{2}{\frac{|T_q|}{|EWU_q|} + \frac{|T_q|-1}{max(|Pairs(EWU_q,1)|,0.5)}} & if|EWU_q| > 0 \\ 0 & otherwise \end{cases}$$

$$M_4 = (S, |EWU_m| - |T_c|, |E_m|)$$

In this function, S represents the harmonic mean of the matched symbols and spatial relationships on the query, to weight large, connected matches higher. The sub-expression $max(|Pairs(EWU_q, 1)|, 0.5)$ is used to avoid S becoming 0 when no tree edges are matched.

3.3 Combining Results

After searching for keywords on the text index and finding documents with matching formulas, these outputs are combined into a single final ranking of documents. For this task, we compute the final score s_f of a document d for query q using a simple linear combination as follows:

$$s_f = \alpha \left(\sum_{e \in q} bestmatch\,(e, d)\, w_e \right) + (1 - \alpha)\, s_t\,(q, d)$$

where α is the weight given to the math component of the query, $bestmatch(e, d)$ is a function that returns the score for the best matching formula in d for math expressions e, w_e is the relative weight given to math expression e. $s_t(q, d)$ represents the text search score for document d as defined in Section 3.1. If there are no keywords on the query q, then $s_t(q, d)$ is equal to 1. Note that $\sum_{e \in q} w_e = 1$ and $0 \leq \alpha \leq 1$.

The function $bestmatch(e, d)$ returns a score vector with m dimensions as defined in Section 3.2.5. The result from s_t is treated as a vector with the actual text score value on the first dimensions and 0 for each of the remaining m - 1 dimensions. The final score of a document is a vector of scores. Documents are sorted by lexicographical order of these score vectors.

4. EXPERIMENTS

In order to investigate how to assign weights to keywords and math expressions, we tested two different ways to assign the value α representing the weight given to math expressions in the query. The first way is *fixed* ($\alpha = 0.5$) where math expressions and keywords weigh the same. The second way is *dynamic* defined as $\alpha = \frac{|E|}{|E|+|T|}$ for each query q, where E and T are the sets of all math expressions and keywords in q respectively.

In order to investigate if larger expressions should have heavier weights than smaller ones on queries with multiple math expressions, we evaluated two different ways to assign the individual weights w_e assigned to each math expression e on a given query q. The first way is *balanced*, where $w_e = \frac{1}{|E|}$ meaning that all math expressions weight the same independently of their size. The second way is *by size*, where larger expressions in the query are given heavier weights using $w_e = \frac{size(e)}{\sum_{x \in E} size(x)}$, where $size(x)$ returns the count of symbols in x.

We configured our system using 4 different settings using combinations of these 2 variables. We used these 4 settings for both the main arXiv tasks and the Wikipedia subtask. We fixed the similarity metric to M_3 as defined in Section 3.2.5. The window size w was unbounded (all possible pairs)

Table 2: NTCIR-12 MathIR arXiv Main Task. Precision@K Results using TREC evaluation tool

	Relevant				Partially Relevant			
Submission	P@5	P@10	P@15	P@20	P@5	P@10	P@15	P@20
MCAT_allfields_lr_unif	0.2621	**0.2448**	0.2046	0.1810	**0.5586**	**0.5483**	0.5126	0.4707
MCAT_allfields_nowgt_unif	**0.2828**	0.2379	**0.2184**	**0.1948**	0.5448	0.5345	**0.5149**	**0.4897**
RITUW_ArXiv_run1 (α-*fixed* w_e-*by-size*)	0.2069	0.1517	0.1126	0.0948	0.4966	0.3966	0.3310	0.2879
RITUW_ArXiv_run2 (α-*fixed* w_e-*balanced*)	0.2069	0.1517	0.1126	0.0948	0.4966	0.3966	0.3310	0.2879
RITUW_ArXiv_run3 (α-*dynamic* w_e-*by-size*)	0.1379	0.1138	0.1034	0.0914	0.4897	0.4586	0.4207	0.3983
RITUW_ArXiv_run4 (α-*dynamic* w_e-*balanced*)	0.1379	0.1138	0.1034	0.0914	0.4897	0.4586	0.4207	0.3983
Ideal Pool Ranking	0.6966	0.5586	0.4644	0.4086	0.9655	0.9552	0.9172	0.8828

Table 3: NTCIR-12 optional MathIR Wikipedia Task. Precision@K Results using TREC evaluation tool

	Relevant				Partially Relevant			
Submission	P@5	P@10	P@15	P@20	P@5	P@10	P@15	P@20
ICST_WikiMainTask	**0.4733**	**0.3767**	**0.2978**	**0.2617**	**0.8533**	**0.7900**	**0.7133**	**0.6600**
RITUW_wiki_run1 (α-*fixed* w_e-*by-size*)	0.2533	0.2367	0.2089	0.1983	0.4933	0.4833	0.4889	0.4733
RITUW_wiki_run2 (α-*fixed* w_e-*balanced*)	0.2533	0.2467	0.2156	0.2017	0.4933	0.4900	0.4822	0.4700
RITUW_wiki_run3 (α-*dynamic* w_e-*by-size*)	0.1600	0.1300	0.1244	0.1267	0.3867	0.3633	0.3733	0.3617
RITUW_wiki_run4 (α-*dynamic* w_e-*balanced*)	0.1600	0.1400	0.1311	0.1300	0.3800	0.3667	0.3644	0.3583
Ideal Pool Ranking	0.8400	0.6967	0.5956	0.5133	0.9467	0.9400	0.9289	0.9217

for the Wikipedia main subtask, and it was set to 2 for the arXiv main task because of memory limitations for larger index. Final submissions can be described as: α-*fixed* w_e-*by-size*, α-*fixed* w_e-*balanced*, α-*dynamic* w_e-*by-size*, α-*dynamic* w_e-*balanced*. Table 2 shows results using Precision@K for $K = \{5, 10, 15, 20\}$ on MathIR arXiv Main Task for relevant and partially-relevant matches. Table 3 shows the same evaluation metrics on optional MathIR Wikipedia Task. For space reasons, we have only included submissions from other systems containing highest scores for any category.

Note that a TREC evaluation tool was used to compute Precision@K for all participants. This tool's default behavior re-ranks documents with the same score based on their ids. Systems like ours use internal tie-breakers that were not submitted as part of the final score, and as a result Precision@K values were affected by re-ranking of documents. If we use the original ranks submitted, our best submission (RITUW_wiki_run2) obtains higher precision values (P@5, P@10, P@15, P@20) of (26.21%, 20.00%, 16.32%, 13.62%) for relevant results and (54.48%, 45.52%, 39.08%, 35.17%) for partially-relevant results on the MathIR arXiv Main Task, and (25.33%, 25.00%, 22.00%, 20.50%) for relevant results and (49.33%, 49.33%, 48.67%, 47.67%) for partially-relevant results on the optional MathIR Wikipedia Task. Complete precision values for all submission are available in the main NTCIR-12 MathIR competition paper [17].

In all result tables, we have included additional rows for "Ideal Pool Ranking" representing the performance of an ideal system with perfect relevance-based ranking for all results in each pool. This represents a soft upper bound for the current pool of results that does not consider additional relevant and partially-relevant documents/formulas in the collection that might be absent from each pool. In addition, we have included rows for "Re-ranking Upper Bound" in Tables 5 and 6 to represent the maximum Precision@K values that our system could get if the top-1000 candidates selected by the core are re-ranked according to their relevance scores.

In order to address our research questions regarding similarity between math expressions, we tested 4 different metrics for the optional MathIR Wikipedia Formula Browsing Task. Table 4 shows results using Precision@K for $K = \{5, 10, 15, 20\}$ on this task for relevant and partially-relevant matches. Table 5 shows the same evaluation when the submitted ranks are used without reranking ties by document name. For metrics M_1, M_2 and M_3, w was unbounded (all pairs). Finally, Table 6 shows our Precision@K results for this task for relevant and partially-relevant matches divided between queries with and without wildcards. There were a total of 20 concrete queries (without wildcards) and 20 wildcard queries in the workload.

In terms of space, our system required 8,348.39 MB in hard drive to index all formulas in the arXiv collection and 580.51 MB for the Wikipedia dataset. These quantities require between 2.0 and 2.5 times that space when loaded into RAM. In terms of run time, we had the following (mean, minimum, maximum, median) times (in seconds): MathIR arXiv Main Task (27.54, 2.77, 178.51, 16.014) and optional MathIR Wikipedia Task (37.83, 1.33, 176.06, 33.84). In the case of the optional MathIR Wikipedia Formula Browsing Task, we obtained: M_1 (2.67, 0.10, 64.13, 1.07), M_2 (12.75, 0.17, 109.61, 3.61), M_3 (45.26, 0.58, 1032.39, 8.58), M_4 (29.80, 0.18, 718.70, 4.67). Note that these run times are typically longer for many queries containing wildcards. For example, M_4 requires (13.05, 1.26, 66.97, 4.50) for concrete queries, but (46.55, 0.18, 718.70, 4.82) for wildcard queries. We use a Ubuntu Linux 14.04 server with 24 Intel Xeon processors (2.93GHz) and 96GB of RAM. While some indexing operations were parallelized, *all retrieval times are reported for single threaded processing.*

4.1 Discussion

For both the arXiv and Wikipedia main tasks, we generally obtained better precision using α-*fixed*. Most queries in the MathIR arXiv Main Task contained multiple keywords and at most one math expression making the *dynamic* setting give larger weights to keywords. In most cases, our system performed better when math expressions in the query had the same weight as keywords. A detailed analysis of results showed that this might be due to multiple cases where documents that only matched keywords were considered irrelevant. In particular, this happened often when these keywords were very generic terms like "define", "name", "arithmetic". Also, certain keywords like "mean" are ambiguous and should only be matched in context. Our current search

Table 4: NTCIR-12 optional MathIR Wikipedia Formula Browsing Task. Precision@K Results using TREC evaluation tool

	Relevant				Partially Relevant			
Submission	P@5	P@10	P@15	P@20	P@5	P@10	P@15	P@20
MCAT-browse_allfields_nowgt_unif	**0.4900**	**0.3900**	**0.3317**	**0.2825**	**0.9100**	**0.8400**	**0.8067**	**0.7687**
RITUW-formula_run1 (M_1)	0.4150	0.3150	0.2650	0.2200	0.8100	0.7450	0.7117	0.6737
RITUW-formula_run2 (M_2)	0.4250	0.3175	0.2567	0.2200	0.8150	0.7550	0.7200	0.6938
RITUW-formula_run3 (M_3)	0.4400	0.3225	0.2700	0.2300	0.8400	0.7650	0.7317	0.7063
RITUW-formula_run4 (M_4)	0.4450	0.2925	0.2517	0.2200	0.8250	0.6825	0.6533	0.6100
Ideal Pool Ranking	0.7900	0.6400	0.5383	0.4725	1.0000	1.0000	0.9933	0.98

Table 5: NTCIR-12 optional MathIR Wikipedia Formula Browsing Task. Precision@K Results using submitted ranks

	Relevant				Partially Relevant			
Submission	P@5	P@10	P@15	P@20	P@5	P@10	P@15	P@20
MCAT-browse_allfields_nowgt_unif	**0.5150**	**0.4050**	**0.3450**	**0.3000**	**0.9300**	**0.8650**	**0.8300**	**0.8012**
RITUW-formula_run1 (M_1)	0.4300	0.3400	0.2933	0.2450	0.8400	0.7800	0.7533	0.7225
RITUW-formula_run2 (M_2)	0.4450	0.3675	0.3100	0.2687	0.8550	0.8125	0.7833	0.7638
RITUW-formula_run3 (M_3)	0.4900	0.3750	0.3283	0.2812	0.8750	0.8175	0.7833	0.7563
RITUW-formula_run4 (M_4)	0.4900	0.3750	0.3217	0.2937	0.9000	0.8250	0.8033	0.7762
Re-ranking Upper Bound	0.7450	0.5625	0.4433	0.3700	1.0000	0.9925	0.9683	0.9375
Ideal Pool Ranking	0.7900	0.6400	0.5383	0.4725	1.0000	1.0000	0.9933	0.9800

Table 6: NTCIR-12 optional MathIR Wikipedia Formula Browsing Task. Precision@K Results using ranks for concrete and wildcard queries

		Relevant				Partially Relevant			
Query Type	Submission	P@5	P@10	P@15	P@20	P@5	P@10	P@15	P@20
Concrete	RITUW-formula_run1 (M_1)	0.4800	0.3550	0.2900	0.2375	**0.9400**	**0.8850**	**0.8267**	**0.7950**
	RITUW-formula_run2 (M_2)	0.4200	0.3300	0.2667	0.2300	0.9200	0.8550	0.8000	0.7700
	RITUW-formula_run3 (M_3)	0.5200	0.3500	0.2933	0.2500	0.9100	0.8600	0.8133	0.7750
	RITUW-formula_run4 (M_4)	**0.5300**	**0.3700**	**0.3167**	**0.2775**	0.9100	0.8250	0.8067	0.7700
	Re-ranking Upper Bound	0.7200	0.5400	0.4167	0.3375	1.0000	1.0000	0.9800	0.9325
	Ideal Pool Ranking	0.7300	0.5800	0.4733	0.4000	1.0000	1.0000	0.9967	0.9800
Wildcard	RITUW-formula_run1 (M_1)	0.3800	0.3250	0.2967	0.2525	0.7400	0.6750	0.6800	0.6500
	RITUW-formula_run2 (M_2)	**0.4700**	**0.4050**	0.3533	0.3075	0.7900	0.7700	0.7667	0.7575
	RITUW-formula_run3 (M_3)	0.4600	0.4000	**0.3633**	**0.3125**	0.8400	0.7750	0.7533	0.7375
	RITUW-formula_run4 (M_4)	0.4500	0.3800	0.3267	0.3100	**0.8900**	**0.8250**	**0.8000**	**0.7825**
	Re-ranking Upper Bound	0.7700	0.5850	0.4700	0.4025	1.0000	0.9850	0.9567	0.9425
	Ideal Pool Ranking	0.8500	0.7000	0.6033	0.5450	1.0000	1.0000	0.9900	0.9800

for keywords in text does not consider context, order or proximity of matched keywords, which results in many irrelevant documents matching only keywords ranked very high.

In terms of weights assigned to each math expression in queries with multiple expressions, we found that there was not much difference between setting $by-size$ and *balanced*. However, in the current query sets most queries contain only one math expression meaning that current data might not be enough to observe a difference if any exists. Analysis of queries with multiple expressions shows that, in many cases, a document is not considered relevant unless all math expressions are matched. The relevance of a math expression in a query is likely to depend on other factors.

Overall, our results show that simple linear combination of weights for keywords and math expressions is not enough for search, and that interactions between math expression and text in their context should be considered in the future.

We can compare our proposed similarity metrics using the results from the optional MathIR Wikipedia Formula Browsing Task shown in Tables 5. Note that for this analysis we only consider Precision@K produced by our ranks originally submitted. As expected, our core engine ranking (M_1) provides a baseline performance before re-ranking with slightly worse results than our re-ranking metrics. Overall, the Maximum Subtree Similarity (M_4) has the best performance of our four metrics. However, a detailed analysis separating concrete from wildcard queries shows that different metrics perform better under certain conditions (see Table 6).

For concrete queries, M_4 performed best in terms of ranking the most relevant matches higher, but M_1 was the best for partially relevant matches. M_4 achieved better performance for relevant results because in many cases some expressions became exact matches after variable unification and were considered relevant by evaluators. In the case of disconnected exact sub-matches, our detailed matching algorithm enforces a connected component constraint and as a result some good candidates might be ranked low by the scoring functions M_2, M_3 and M_4 based on the size of the largest sub-match. However, disconnected exact sub-matches tend to be good only if they appear on similar baselines. For example, for the query $u+v+x+y+z$, the match $u+v-y+z$ might be considered acceptable, but $\frac{u+v}{y+z}$ might not. Also, M_3 and M_4 increase the rank of smaller unified matches that often were considered to be less relevant than disconnected exact matches.

For wildcard queries, all metrics using wildcard expansion (M_2, M_3, M_4) performed better on average than the core engine (M_1) that only considers single symbol matches. In terms of relevant matches, the gap between these three metrics that consider wildcard expansion is small. At the level of partial matches, M_4 performs better than M_3 because multiple slightly relevant unified matches are ranked higher. For queries containing polynomials, wildcard horizontal expansion to the left matched high degree polynomials that were considered irrelevant.

Unification of variables and constants was useful when applied over candidates with high structural match. However, multiple irrelevant expressions are ranked too highly when unification is applied over partial structural matches.

By comparing the best results for each metric (M_1-M_4) to the "Re-ranking Upper Bound" and "Ideal Pool Ranking" in Tables 5 and 6, we can measure how much room there is for improvement both in terms of initial candidate selection and re-ranking. For Precision@20, the gap between "Ideal Pool Ranking" and "Re-ranking Upper Bound" is currently 10.25% for relevant results meaning that, in average, there are at least 2 out of 20 relevant results in the ideal pool that are not selected by our system as initial candidates. For concrete queries only, this gap is just 6.25% (around 1 candidate), and for wildcard queries the gap is 14.50% (around 3 candidates). In terms of re-ranking, the gap for Precision@20 for relevant results between the best re-ranking metric and the current upper bound 7.63% overall, 6.00% for concrete-only and 9% for wildcard-only. The larger performance gaps are currently on queries containing wildcards. Note that the values for our "Re-ranking Upper Bound" are higher than the best submission (7% of difference in Precision@20 for relevant results), meaning that we could achieve similar precision values by improving our re-ranking functions using the same candidates from the core.

5. CONCLUSION

A math-aware retrieval system has been presented, and its performance has been evaluated on multiple MathIR tasks. Our research questions from the introduction have also been partially answered. In terms of weighting query terms (**RQ1**, **RQ2**), we need a better model for weight distribution that considers interactions between math expressions and text. Some query terms might be considered more important based on their intrinsic value to the query topic independently of their size in terms of count of symbols.

The optional MathIR Wikipedia Formula Browsing Task in particular was useful to provide additional answers to our questions. Our current set of candidates obtained by the core using the Dice coefficient (M_1) is competitive, but there is considerable room for improvement (**RQ3**). A detailed matching algorithm was generally helpful to rank the most relevant matches higher, but some partially-relevant disconnected matches are ranked too low (**RQ4**). A less constrained matching algorithm can be helpful in ranking these candidates higher. On the other hand, the unification process might need to be constrained to avoid ranking many irrelevant partial matches too high (**RQ5**). Finally, both the Dice Coefficient and the MSS performed better under different circumstances (**RQ6**). Further analysis can be used to create similarity metrics that perform better.

6. REFERENCES

[1] A. Aizawa, M. Kohlhase, I. Ounis, and M. Schubotz. NTCIR-11 Math-2 task overview. In *NTCIR*, pages 88–98, 2014.

[2] P. Graf. Substitution tree indexing. In *RTA*, pages 117–131, 1995.

[3] R. Hambasan, M. Kohlhase, and C.-C. Prodescu. Mathwebsearch at ntcir-11. In *NTCIR*. Citeseer, 2014.

[4] H. Hiroya and H. Saito. Partial-match retrieval with structure-reflected indices at the NTCIR-10 math task. In *NTCIR*, pages 692–695, 2013.

[5] S. Kamali and F. W. Tompa. A new mathematics retrieval system. In *CIKM*, pages 1413–1416. ACM, 2010.

[6] S. Kamali and F. W. Tompa. Structural similarity search for mathematics retrieval. In *Intelligent Computer Mathematics*, pages 246–262. Springer, 2013.

[7] X. Lin, L. Gao, X. Hu, Z. Tang, Y. Xiao, and X. Liu. A mathematics retrieval system for formulae in layout presentations. In *SIGIR*, pages 697–706, New York, NY, USA, 2014. ACM.

[8] B. R. Miller and A. Youssef. Technical aspects of the digital library of mathematical functions. *Annals of Mathematics and Artificial Intelligence*, 38(1-3):121–136, 2003.

[9] J. Mišutka and L. Galamboš. System description: Egomath2 as a tool for mathematical searching on wikipedia. org. In *Intelligent Computer Mathematics*, pages 307–309. Springer, 2011.

[10] T. T. Nguyen, S. C. Hui, and K. Chang. A lattice-based approach for mathematical search using formal concept analysis. *Expert Syst. Appl.*, 39(5):5820 – 5828, 2012.

[11] N. Pattaniyil and R. Zanibbi. Combining TF-IDF text retrieval with an inverted index over symbol pairs in math expressions: The Tangent math search engine at NTCIR 2014. In *NTCIR*, pages 135–142, 2014.

[12] M. Ružicka, P. Sojka, and M. Líška. Math indexer and searcher under the hood: History and development of a winning strategy. In *NTCIR*, 2014.

[13] P. Sojka and M. Líška. Indexing and searching mathematics in digital libraries. In *Intelligent Computer Mathematics*, pages 228–243. Springer, 2011.

[14] D. Stalnaker and R. Zanibbi. Math expression retrieval using an inverted index over symbol pairs. In *DRR*, volume 9402, pages 940207–1–12, 2015.

[15] L. Wang, J. Lin, and D. Metzler. A cascade ranking model for efficient ranked retrieval. In *SIGIR*, pages 105–114, 2011.

[16] Y. Wang, L. Gao, X. Liu, and K. Yuan. WikiMirs 3.0: a hybrid MIR system based on the context, structure and importance of formulae in a document. In *Proc. JCDL*, pages 173–182, 2015.

[17] 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.

[18] R. Zanibbi and D. Blostein. Recognition and retrieval of mathematical expressions. *IJDAR*, 15(4):331–357, 2012.

[19] R. Zanibbi, K. Davila, A. Kane, and F. Tompa. Multi-stage math formula search: Using appearance-based similarity metrics at scale. *SIGIR*, 2016.