Multi-Stage Math Formula Search: Using Appearance-Based Similarity Metrics at Scale

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

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

ABSTRACT

When using a mathematical formula for search (*query-by-expression*), the suitability of retrieved formulae often depends more upon symbol identities and layout than deep mathematical semantics. Using a *Symbol Layout Tree* representation for formula appearance, we propose the *Maximum Subtree Similarity (MSS)* for ranking formulae based upon the subexpression whose symbols and layout best match a query formula. Because MSS is too expensive to apply against a complete collection, the Tangent-3 system first retrieves expressions using an inverted index over symbol pair relationships, ranking hits using the Dice coefficient; the top-k formulae are then re-ranked by MSS. Tangent-3 obtains state-of-the-art performance on the NTCIR-11 Wikipedia formula retrieval benchmark, and is efficient in terms of both space and time. Retrieval systems for other graphical forms, including chemical diagrams, flowcharts, figures, and tables, may benefit from adopting this approach.

Keywords

mathematical information retrieval (MIR), inverted index, query-by-expression, subtree similarity

1. INTRODUCTION

The Web is a rich repository of mathematical information, including large repositories of technical documents in online paper databases, tutorials and instructional materials, and other publications in mathematics and engineering. Numerous online resources are available for both experts and non-experts. Students may explore and review concepts using Wikipedia, MathPlanet, and the Khan Academy, while experts consult resources such as the On-Line Encyclopedia of Integer Sequences (https://oeis.org/).

While mature technologies for text search are readily available, search using formulae as queries *(query-by-expression)* remains an open research problem [8, 18, 28]. Interest in Mathematical Information Retrieval (MIR) has increased in recent years, as witnessed by math retrieval tasks at the NTCIR conferences [1, 2].

For the most part, large-scale search engines support formula search indirectly, e.g., through LaTeX strings as search terms. There is no wildcard support for formula subexpressions; for example, we cannot reliably locate expressions containing α with an unspecified exponent (α^*), or coefficient ($*\alpha$). This makes it difficult for users to browse for similar formulae in documents, clarify unfamiliar notation [25], or discover non-obvious connections between research papers based on their mathematical contents [27]. A recent report from the National Research Council (USA) considers developing a global library for mathematics research, and includes a discussion of limitations in existing search tools with regards to mathematical information [5].[1]

Math encodings are naturally hierarchical, defining formula appearance (through the arrangement of symbols on writing lines) in *Symbol Layout Tree (SLT)* encodings such as LaTeX or formula semantics in *Operator Tree (OT)* encodings such as Content MathML. Math symbol definitions are dialectic and context-dependent, and in general translating between appearance and semantic encodings is heuristic [28].

Both layout tree and operator tree representations have been used for query-by-expression (see Section 2). In the most general sense, query-by-expression involves tree matching for a class of SLTs or OTs. Open problems include defining appropriate similarity metrics, identifying what primitives to use for representing formula content, how to index and retrieve these primitives, and finally how to handle trade-offs between recall and both retrieval time and index size. In general, retrieval becomes more difficult when the best matches are less similar to the query formula (e.g., due to large expressions in the corpus with subexpressions similar to one or more parts of the query), and when wildcards are included in the query [10]. As SLT formula encodings are more common in practice, we focus on them in this paper.

Problem Statement. Create a formula search engine model that 1) effectively retrieves and ranks formulae based on formula appearance, 2) is fast enough to use in real-time with large corpora, 3) has index sizes and indexing speeds that are scalable, and 4) supports wildcards.

Contributions. This paper makes four primary contributions: 1) the *Maximum Subtree Similarity (MSS)* metric for ranking formulae using the best query match with unification and wildcard support (see Section 6), 2) an improved SLT model with consistent representation for lists, grids and matrices (see Section 4), 3) a novel, effective and efficient

SIGIR 2016 July 18–20, 2016, Pisa, Italy
© 2016 ACM. ISBN 978-1-4503-4069-4/16/07…$15.00
DOI: http://dx.doi.org/10.1145/2911451.2911512

[1]See p.28, "What is missing from the mathematical landscape?"

core engine for symbol pair-based retrieval (see Section 5), and 4) state-of-the-art performance on the NTCIR-11 formula retrieval benchmark: our method is efficient in terms of time and space, and produces effective formula search results (see Section 7).

2. RELATED WORK

In this section we provide an overview of existing techniques for formula retrieval. We categorize approaches to query-by-expression as *text-based*, *tree-based*, or *spectral* according to the primitives used to represent formulae.

Text-Based Approaches. In text-based approaches, math expression trees are linearized before indexing and retrieval. Common normalizations include defining synonyms for symbols (e.g., function names), using canonical orderings for commutative operators and spatial relationships (e.g., to group `a+b` with `b+a` and `x_i^2` with `x^2_i`), enumerating variables, and replacing symbols by their mathematical type (e.g., numbers, variables, and classes of operators) [19,28].

Although linearization masks significant amounts of structural information, it allows text and math retrieval to be carried out efficiently by a single search engine. Most text-based methods use TF-IDF (term frequency-inverse document frequency) retrieval after linearizing expressions [14,19]. In an alternative approach, the largest common substring between the query formula and each indexed expression is used to retrieve LaTeX strings [17]. This captures more structural information, but also requires evaluating all expressions in the index using a quadratic algorithm.

Tree-Based Approaches. These methods represent formula appearance or semantics directly as trees. Expressions are indexed as complete trees, along with their subtrees to support partial matching. Tree indices may be compressed by storing identical subtrees uniquely [10]. In addition to exact matching of trees and/or subtrees, tree-edit distances with early stopping have been used for fast retrieval [11]. The *substitution tree* [6] has been used to create indices for operator trees, with each path representing a series of subexpression variable substitutions [12]. The NII group from Japan has devised a hashing scheme to represent operator trees by a set of interdependent codes for all subexpressions [2,18] (see Section 7 for results).

One method adapts TF-IDF retrieval for SLTs, using vectors of subexpressions, along with subexpressions where arguments are replaced by wildcards [13]. 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 [24].

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 [15]. 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 [9] 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-2* search engine uses the *relative* position of symbols to create a bag of symbol pairs [20], along with extensions to represent matrix and grid structures [16]. A *symbol pair* gives the location of symbol s_2 relative to an ancestor symbol s_1 in an SLT $(s_1, s_2, \delta x, \delta y)$, where $(\delta x, \delta y)$ are horizontal and vertical displacements along writing lines from s_1 to s_2. 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*).[2] A Dice coefficient variant incorporating symbol pair frequencies was found to perform similarly [20]. This technique combined with keyword retrieval in Lucene produced the highest Precision@5 result for the NTCIR-11 math retrieval task (92%) [2].

Limitations of Symbol Pair-Based Retrieval. In Tangent-2, symbol pair matches may be scattered throughout a formula, and exact matches for small subexpressions in candidates produce low similarity scores. Wildcards match individual symbols, and pairs of wildcards are ignored to avoid performance problems. Non-wildcard symbols are not unified: symbol pairs that are equivalent after renaming are missed (e.g., x^2 does not match a^2). The Tangent-2 formula structure model is also inconsistent, with different groupings for roots, matrices, vectors and parenthesized expressions, and retrieval is very slow (see Section 7).

In this paper, we make improvements to the Symbol Layout Tree model used in Tangent, along with the symbol pairs data model. These changes reduce index sizes substantially while improving retrieval results. We also define a new engine for fast retrieval of symbol pair matches, and address limitations in wildcard handling and locality of matching through an additional re-ranking step.

3. FORMULA RETRIEVAL MODEL

The Tangent-3 formula search engine (i.e., the newest version of Tangent presented in this paper) employs a two-stage cascading search [23] for fast retrieval and intuitive rankings (see Figure 1).[3] Queries are parsed into a Symbol Layout Tree, 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).

After parsing, the first 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 Section 5). Tuples with one wildcard are expanded, but tuples with two wildcards are ignored for efficiency. Iterator trees are used to process postings quickly. The initial ranking weighs matched vs. unmatched symbol pairs in the query and candidates. The second (*re-ranking*) stage re-scores matches using an approximate best matching subtree for the query in each candidate (see Section 6), addressing limitations of symbol pair-based retrieval described in the previous section.

Illustration. Table 1 shows queries processed using our two-stage method. For query 1, using MSS for re-ranking

[2]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 $\frac{2|F_q \cap F_c|}{|F_q|+|F_c|}$.

[3]Source code: http://www.cs.rit.edu/~dprl/Software.html.

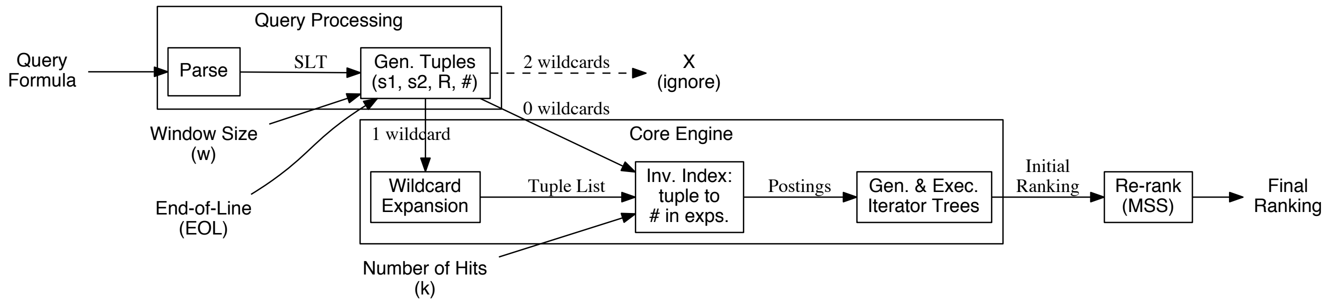

Figure 1: Tangent-3 Formula Retrieval Model. System parameters include maximum symbol pair distance (window size w), how end-of-line symbols are indexed (EOL), and the number of hits to return (k).

Table 1: Top-5 Results for Tangent-3 (k=100). Asterisks represent wildcards (e.g., * or *1*).

	Initial Ranking	Re-ranked (MSS)
Query 1: $f_*(z) = z^2 + c$		
1.	$f_c(z) = z^2 + c$	$f_c(z) = z^2 + c$
2.	$f_c(z) = z^2 + c.$	$\mathbf{P}_c(z) = z^2 + c$
3.	$f(z) = z^2 + c$	$f_c(\mathbf{x}) = \mathbf{x}^2 + c$
4.	$f_0(z) = z^2$	$f_c(z) = z^2 + c.$
5.	$f_c(z) = z * z + c$	$f(z) = z^2 + c$
Query 2: $\sum_{*2*}^{*1*} * = \sum_{*2*}^{*1*} *$		
1.	$E = \sum_i^N E_i$	$\sum_{i=1}^d a_i = \sum_{i=1}^d b_i$
2.	$G_{net} = \sum_i \sum_{i=1}^N$	$\sum_{i=1}^N d_i = \sum_{i=1}^N \lambda_i.$
3.	$\sum_i^{N_1} p_i = \sum_j^{N_2} p_j$	$\sum_{n=0}^\infty a_{\sigma(n)} = \sum_{n=0}^\infty a_n.$
4.	$\sum_{i=1}^n x_i k_i = \sum_{i=1}^n x_i$	$\sum_i^{N_1} p_i = \sum_j^{N_2} p_j$
5.	$= \sum_{k=1}^n a_k$	$\sum_{n=0}^\infty a_n = \sum_{n \in N} a_n.$

produces top-5 hits matching the query formula exactly after unifying identifiers (before re-ranking, only the top-3 match). For query 2, the numbered wildcards *1* and *2* should be identical when repeated. Before re-ranking only one hit matches the query exactly (rank 4), but after re-ranking the top-3 are exact matches for the query, and the remaining two hits are strong partial matches.

4. FORMULA STRUCTURE MODEL

The Tangent formula search engines use a Symbol Layout Tree (SLT) to represent formula appearance (see Figure 2). Whereas Tangent-2 based its encoding on a two-dimensional interpretation of formulas on a page, expressing symbol positions in terms of horizontal and vertical offsets (see Section 2), this revised SLT representation provides greater consistency and expressivity in representing relationships between symbols. In our representation, matrices are an integral part of formulas rather than auxiliary relational structures. Tangent-3 also includes a unified representation of all parenthesized subexpressions regardless of their interpretation (for example, as function arguments vs. parenthesized matrices). We also add a crude representation of type, which is critical for re-ranking using Maximum Subtree Similarity. We describe these in more detail below.

Node Labels and Types. Nodes in an SLT represent individual symbols and visually explicit aggregates, such as fractions, matrices, function arguments, and parenthesized expressions. More specifically, SLT nodes represent:

- typed mathematical symbols: numbers (N!n); identifiers such as variable names (V!v); text fragments, such as *lim*, *otherwise*, and *such that* (T!t)
- fractions (F!)
- container objects: radicals (R!); matrices, tabular structures, and parenthesized expressions (M!$frxc$)
- explicitly specified whitespace (W!)
- wildcard symbols (*w)
- mathematical operators

Because of their visual similarity, all tabular structures, including matrices, binomial coefficients, and piecewise defined functions are encoded using the matrix indicator M!. If a matrix-like structure is surrounded by fence characters, then those symbols are indicated after the exclamation mark. Finally, the indicator includes a pair of numbers separated by an x, indicating the number of rows and the number of columns in the structure. For example, M!2x3 represents a 2x3 table with no surrounding delimiters and M!()1x5 represents a 1x5 table surrounded by parentheses.

Importantly, *all* parenthesized subexpressions are treated as if they were 1x1 matrices surrounded by parentheses, and, in particular, the arguments for any n-ary function are represented as a 1xn matrix surrounded by parentheses.

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*. Using type V! for all identifiers simplies our SLT model, but sometimes leads to unexpected unifications (see Section 7).

Spatial Relationships. Labeled edges in the SLT capture the spatial relationships between objects represented by the nodes. With respect to a given object O, seven axes reflect the following relationships:

1. **next** ($\rightarrow$) references the adjacent object that appears to the right of O and on the same line
2. **within** ($\boxed{\cdot}$) references the radicand if O is a root or the first element appearing in row-major order in O if it is a structure represented by M!
3. **element** ($\multimap$) references the next element appearing after O in row-major order inside a structure represented by M!
4. **above** ($\uparrow$) references the leftmost object on a higher

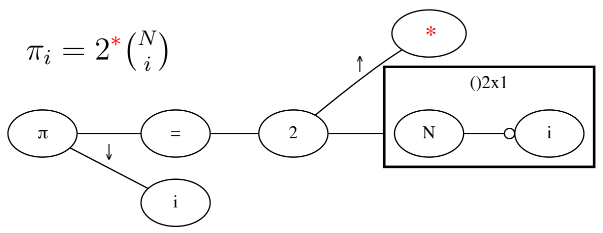

Figure 2: Query Formula with Corresponding SLT. The query has one wildcard, and a tabular structure.

line starting at the position above O (e.g., superscript, over symbol, fraction numerator, or index for a radical)

5. **below** (↓) references the leftmost object on a lower line starting at the position below O (e.g., subscript, under symbol, fraction denominator)
6. **pre-above** (⇑) references the leftmost object of a prescripted superscript of O
7. **pre-below** (⇓) references the leftmost object of a prescripted subscript of O

An SLT is rooted at the leftmost object on the main writing line of the formula it represents. Figure 2 shows an example of an SLT, where for simplicity, unlabeled edges represent the *next* relationship and the type prefixes are omitted. We do not distinguish vertical from scripted regions.[4] This causes $\hat{x}^2$ to be represented without an accent, but also represents limits uniformly: $\int_i^n$ and $\int\limits_i^n$ are equivalent.

Creating SLTs. SLTs can be created in linear time from Presentational MathML by a recursive descent parser. For other input formats, we assume that converters such as LaTeXML[5] exist to produce Presentational MathML.

In most circumstances, whitespace is not represented in an SLT. As a result, although unicode whitespace and related characters, such as "invisible times" (`U+2062`), occasionally appear as operators in Presentational MathML expressions, they are all ignored for the purpose of matching expressions in Tangent-3.

Tuple Representation for SLTs. An effective formula search engine must be able to find formulae that contain a query formula, appear within a query formula, or are in many ways only similar to a query formula. Thus high-quality search engines create indexes based on selected *features* of formulae found in a corpus and match queries based on those features. Previous versions of Tangent showed that pairs of symbols together with their relative inter-symbol distances in two dimensions are effective features to use [16], and we improve on this approach.

As described above, a node in an SLT can have up to seven labeled outgoing edges (with no edge label repeating for any node) corresponding to the seven defined axes. For a given SLT, Tangent-3 produces a set of tuples, each of which encodes the relationship between a pair of symbols occurring on some path from the root to a leaf. Given two nodes on such a path, we define the relative path between the nodes by the sequence of edge labels traversed from the ancestor node to the descendant. The features we use are tuples encoding pairs of labels occurring on ancestor and descendant nodes, together with their relative path and the number of times each such label-label-path triple occurs in an SLT. For example, the tuple ($V!x$,$N!2$,→↑→→,3) indicates that the corresponding formula includes three instances of a node representing the number 2 that appears "to the right, then above, and then twice to the right" with respect to some node representing variable x; for example, such a formula might include ...xy^{z+2}....

Tuple Generation Parameters. As seen in Figure 1, there are two parameters that control symbol pair tuple generation: the window size (w), and how symbols at the end of writing lines are included in the index (EOL).

To save both space and time, and following the practice of searching via n-grams [26], Tangent-3 extends the approach used in Tangent-2 by storing only those tuples for which the distance between symbols (measured by the number of edges separating them) is less than or equal to a specified window size w. For example, the tuple ($V!x$,$N!2$,→↑→→,3) will be included in the index only if $w \geq 4$.

In addition to symbol pairs, end-of-line information can be captured by introducing special tuples of the form (*symbol*, !0, →, *count*). Including these is likely to improve retrieval, particularly for very small expressions. However, wildcards as EOL symbols have very large wildcard expansions, increasing retrieval time. To alleviate this problem, we examine adding EOL symbols for small expressions with height two or less (*Sm-EOL*), adding all EOL symbols (*EOL*), and omitting EOL symbols (*No-EOL*).

5. CORE ENGINE

The core engine is the first retrieval stage in Tangent-3, quickly finding the top-k highly relevant matches for a formula query, which are later re-ranked (see Figure 1). The engine ranks these top-k formulae using a simple algorithm, along with a list of document locations for each formula.

Since runtime performance is a high priority, the core engine uses a customized inverted index data structure implemented in C++. 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.

The input to the indexer is a set of document names and the extracted mathematical formulae found in each document, and the input to the search engine is a single query formula. Each formula is converted to a set of tuples (see Sections 3 and 4) that serve as index and search "terms."

Index Data Structures. At index time, an inverted index is built over the given document-formula-tuple relationships, using two main data structures: *dictionaries (D_*)* convert objects (such as strings or tuples) into a compact 0-based range of internal identifiers (integers) and *postings lists (P_*)* are lists of integer tuples ordered by the first integer in the tuple. The data structures listed below can be combined to produce compression, ease of storage, and fast access speeds.

D_f: formula → formID
D_t: tuple → tupleID
D_d: document → docID
D_w: wildcardtuple → wildcardtupleID
P_t: tupleID → (formID, count)$^+$
P_f: formID → (docID, position)$^+$
P_w: wildcardtupleID → (tupleID)$^+$

The index includes postings lists P_t that map each tuple

[4]This was later changed for *non-operator* symbols.
[5]http://dlmf.nist.gov/LaTeXML/

to all formulae containing that tuple. A query can thus be implemented by combining the corresponding tuples' postings lists using an OR operator. We store these postings lists as ordered lists of formula identifiers (integers), so that the lists can be easily combined using a merge algorithm. Dictionary D_f defines a consistent assignment of a formula to its identifier, and another dictionary D_t is used for tuples, thus saving both space and time in the engine.

In order to return document information for query results, the engine stores postings lists P_f mapping each formula identifier to the identifiers of the documents containing those formulae, along with their positions in documents. Dictionary D_d is used for document names.

Wildcards. The core engine supports limited wildcard functionality. As illustrated in Figure 1, query tuples containing a single wildcard are implemented as iterator expansions. The engine stores postings lists P_w that map each wildcard tuple to the set of tuple identifiers that match. Assigning tuple identifiers using a dictionary D_w again gives some compression benefits. Implementing even this restricted wildcard functionality can be expensive, since the iterator expansion can be quite large.[6]

Searching. Query processing follows the architecture shown in Figure 1. 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. As optimizations, iterators may skip over some formulae based on thresholds and max-score calculations (*see below*). After the iterators are finished, matching formulae and scores are returned along with the associated document names.

The engine uses Dice's coefficient over tuples as a simple ranking algorithm, counting the number of tuples that overlap between the query and a candidate formula using the query iterators. The engine also stores the tuple count for each formula (the size of the formula) in an array A_s and uses these values in the ranking calculation:

$$A_s\colon \text{formID} \rightarrow \text{tuplecount}$$

Since wildcards can often match multiple tuples in a query and overlap with other wildcards, there could be multiple ways to count the tuples that overlap. The engine implements a greedy counting approach by simply assigning the matches for tuples when each of the iterators is advanced.

Optimizations. Even using simple dictionary and postings list implementations (i.e., std::maps and 32-bit arrays), the engine's data structures are small enough to be run in memory for the datasets being examined, so we do not consider compressing these data structures here. Nevertheless, query processing might still be slow, even though the data structures are in memory, ranking is fast, and using a dictionary avoids repeated processing of duplicate formulae. As a result, the techniques below are used to reduce query execution time (related results may be found in Section 7):

O_1: Avoid processing all postings by allowing skipping in query iterators. This functionality is implemented using doubling (galloping) search [3].

O_2: Skip formulae based on size thresholds. We use the current top-k candidate list to define a minimum score that defines minimum and maximum tuple size thresholds from the definition of Dice's coefficient. We also improve on the effectiveness of these thresholds by reordering formula identifiers: sort the formulae by size, split into quartiles $\{q_1, q_2, q_3, q_4\}$, and then reorder $\{q_2, \mathit{reverse}(q_1), q_3, q_4\}$.

O_3: Avoid formulae that match only wildcard tuples when the score threshold allows. This is similar to the max-score optimization [21], only at a coarser granularity.

O_4: Avoid processing all wildcard tuple expansions. If a tuple is matched to a wildcard for the next formula, do not process the remaining iterators for this wildcard.

O_5: Process iterators for large postings lists first. Evaluate the binary operator tree left-first and order tree operators descending by size when possible.

6. RE-RANKING BY MAXIMUM SUBTREE SIMILARITY

In this section we describe an alternative to Dice's coefficient that is particularly effective in ranking mathematical formulae. We first formalize matching of subtrees based on their structure and then on their consistent re-labelling, providing support for unification of identifiers and constants (which is absent in a direct application of Dice's coefficient to tuples). Next we define a metric based on such matched subtrees, and then apply the definition to score a candidate SLT against a query that may include wildcards.

Notation: The label on node n in SLT T is denoted $\lambda(n)$. The number of nodes in SLT T is denoted $|T|$. We also write $n \in T$ if n is a node in T and $(n_1, n_2) \in T$ if (n_1, n_2) is an edge in T.

Approximate matches of formulae might involve 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.[7]

Definition (*aligned SLTs*): 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 corresponding 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.) For N a subset of nodes in T_1, we define $f(N) = \{f(n) \mid n \in N\}$.

Approximate matches might also involve simple replacements of symbols in one SLT by alternative symbols (e.g., x for y or 3 for 2). Naturally, a wildcard symbol can be replaced by any symbol.

Definition (*unified nodes*): Node n_1 in SLT T_1 can be *unified* with node n_2 in SLT T_2, denoted $n_1 \dashrightarrow n_2$, if any of the following conditions holds:

- Both n_1 and n_2 have type *variable name* (V!),

[6]The engine does not try to enforce wildcard variable agreement between tuples (wildcard joins), and it ignores multi-wildcard tuples. An initial implementation handling multi-wildcard tuples and wildcard joins was found to be approximately one hundred times slower for a small dataset.

[7]Formally, a "part" of an SLT T will be a *pruned subtree*: any connected subset of labelled nodes from T together with the labelled edges connecting those nodes. Thus a pruned subtree of T is itself an SLT, but it need not extend to the leaves of T. Henceforth, we will use "subtree" to mean "pruned subtree."

- Both n_1 and n_2 have type *number* (N!),
- Both n_1 and n_2 have type *matrix* (M!),
- n_1 has type *wildcard* (*), or
- $\lambda(n_1) = \lambda(n_2)$.

Next, 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 start by identifying candidate sets of nodes in T_1 that can be consistently relabelled.

Definition (*alignment partition*): Given T_1 and T_2, two aligned SLTs with isomorphism f from T_1 to T_2, an *alignment partition* is a subset of nodes N in T_1 such that $(x \in N \wedge y \in N) \Rightarrow (\lambda(x) = \lambda(y) \wedge x \dashrightarrow f(x) \wedge y \dashrightarrow f(y) \wedge \lambda(f(x)) = \lambda(f(y)))$ (i.e., the nodes have identical labels and their unified images have identical labels or identical SLTs). For node $n \in T_1$, we define $P(n)$ to be the alignment partition containing n if it exists and $\emptyset$ otherwise. (Note that $n \in P(n) \Leftrightarrow n \dashrightarrow f(n)$.) For alignment partition A, $\lambda(A)$ denotes the label common to all nodes in A and $\lambda(f(A))$ denotes the label that is common to all nodes in $f(A)$.

We can now choose a set of partitions that are consistent in their relabelling of nodes.

Definition (*matched set of nodes*): Given aligned SLTs T_1 and T_2 with isomorphism f from T_1 to T_2 and the set of all corresponding alignment partitions, we define a *matched set of nodes* M as

$$M = \{n \in T_1 \mid n \in P(n) \wedge \forall n' \in M$$
$$([\lambda(n') = \lambda(n) \vee \lambda(f(n')) = \lambda(f(n))] \Rightarrow n' \in P(n))\}$$

In preparation to preferring matches of large *connected* parts of SLTs, let $E(M) = \{(n_1, n_2) \mid n_1 \in M \wedge n_2 \in M \wedge (n_1, n_2) \in T_1\}$, the set of edges induced by M.

Note that there may be many possible matched sets of nodes for a given alignment, depending on which alignment partitions are chosen to be included.

Because the 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.

We suggest the following properties for a scoring function, as illustrated in Figure 3: 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. We employ such a scoring function:

Definition (*SLT score*): Given a query SLT T_q, an SLT T_c for a candidate match, and two aligned SLTs T_1 and T_2 where T_1 is a subtree of T_q and T_2 is a subtree of T_c, let M be a matched set of nodes for T_1 and T_2. Let S be the harmonic mean of the fraction of nodes from T_q preserved by M and the fraction of edges preserved by $E(M)$, i.e., $S = \frac{2}{\frac{|T_q|}{|M|} + \frac{|T_q|-1}{max(|E(M)|,0.5)}}$ if $|M| > 0$, otherwise 0; this is a measure of the size of the consistent structural part of the match. The *score* of T_c with respect to T_q, T_1, T_2, and M is denoted $s(T_q, T_c; T_1, T_2, M)$ and defined as the tuple composed of the following parts:

1. *structural match*: S (favor large matches).
2. *unmatched*: the negation of the number of unmatched nodes in T_c, i.e., $|M| - |T_c|$.
3. *exact match*: $|\{n \in M \mid \lambda(n) = \lambda(f(n))\}|$.

Scores assigned to any two candidate matches are compared lexicographically to determine which candidate ranks higher.

For aligned SLTs T_1 and T_2 with isomorphism f from T_1 to T_2 and the set of all corresponding alignment partitions, we want to choose a matched set of nodes M that produces a high score, but evaluating all matched sets induced by an alignment is too expensive. Therefore we use a greedy algorithm to select which partitions to include in the matched set of nodes, based on the properties used for scoring.

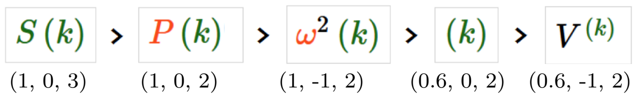
Figure 3: MSS Scoring for Query $S(k)$. Ranking triples contain MSS (1), and the number of candidate symbols that are unmatched (2) and match the query exactly (3). Parentheses count as one symbol.

Algorithm 1 Greedy selection of matching subtree M

```
1. Let $A_0$ be the alignment partition containing the most
   nodes; if multiple partitions have the most nodes, let $A_0$
   be one of those partitions where $\lambda(A_0) = \lambda(f(A_0))$ if it
   exists; otherwise let $A_0$ be any of the largest alignment
   partitions. Initialize $M$ to include all nodes in $A_0$.
2. Repeatedly identify the largest alignment partition $A_i$ such
   that $\lambda(A_i)$ does not label any node in $M$, and $\lambda(f(A_i))$ does
   not label any node unified with a node in $M$, choosing $A_i$
   where $\lambda(A_i) = \lambda(f(A_i))$ if it exists; replace $M$ by $M \cup A_i$.
3. Stop if no more alignment partitions can be added to $M$.
```

If hash tables are used to record which node labels have been included in M and in $f(M)$, checking for duplicate labels can be performed in $O(1)$ time. Partitions can be considered one by one in decreasing order of size, which requires $O(|T_q| \log(|T_q|))$ time to initialize and then $O(|T_q|)$ to enumerate since the number of partitions cannot exceed the number of nodes in T_q.

Finally, to compare a query SLT T_q against a candidate SLT T_c, we choose a pair of aligned subtrees that maximizes the score for the candidate with respect to the query. Thus for each node n in T_q, we consider the score of the largest subtree rooted at n that can be aligned with some subtree in T_c. We start with a formal definition of the largest subtree.

Definition (*alignable and maximally similar subtree*): Given SLTs T_q and T_c and aligned SLTs T_1 rooted at r_1 and T_2 rooted at r_2 with isomorphism f from T_1 to T_2, where T_1 is a subtree of T_q and T_2 is a subtree of T_c, let $m = |\{n \in T_1 \mid n \dashrightarrow f(n)\}|$. Let $\frac{2m}{|T_1|+|T_q|}$ be a measure of similarity of T_1 to T_q with respect to T_2 (Dice's coefficient). T_1 is then *alignable and maximally similar* to T_q with respect to T_2 if r_1 can be unified with r_2 and there is no other SLT T_1' rooted at r_1 and T_2' rooted at r_2 with corresponding measure m', where T_1' is a subtree of T_q, T_2' is a subtree of T_c, $|T_1'| > |T_1|$, and $\frac{2m'}{|T_1'|+|T_q|} > \frac{2m}{|T_1|+|T_q|}$.

Definition (*Maximum Subtree Similarity score*): Given SLTs T_q and T_c, consider pairs of nodes $n_{i_1} \in T_q$ and $n_{i_2} \in T_c$ such that n_{i_1} can be unified with n_{i_2}. Let T_{i_1}, rooted at n_{i_1}, be alignable and maximally similar to T_q with respect to tree T_{i_2}, rooted at n_{i_2}. The *Maximum Subtree Similarity* score $\text{MSS}(T_q, T_c)$ of T_c with respect to T_q is $\max_i s(T_q, T_c; T_{i_1}, T_{i_2}, M_i)$ over all such pairs, where M_i uses Algorithm 1 to chose matched sets of nodes.

THEOREM 1. *Computing Maximum Subtree Similarity for a candidate formula requires time* $O(|T_c||T_q|^2 \log(|T_q|))$.

PROOF. The number of pairs of aligned subtrees is at most $|T_q| * |T_c|$. For each pair, checking whether the roots can be unified takes $O(1)$ time, finding the largest alignable subtrees takes $O(|T_q|)$ time, and computing the score takes constant time plus time $O(|T_q| log(|T_q|))$ to choose M. □

MSS ranks more structurally similar subexpressions on a candidate higher, on the assumption they will be perceived as more relevant by users. Were this true, ranking by MSS would be consistent with the Probability Ranking Principle [22], that hits are ideally sorted by decreasing order of probable relevance, $P(c|q)$. In the next Section we show experimentally that the MSS similarity metric performs very well. In the future one might examine the correlation of MSS with $P(c|q)$ more directly.

7. EVALUATION

We now present experiments designed to observe the effect of system parameters on index size, retrieval time, and results, along with a human assessment of top-10 results.

Our main dataset is the NTCIR-11 Wikipedia collection with 30,000 articles totalling 2.5 GB and containing roughly 387,947 unique LaTeX expressions. In addition, we use the much larger NTCIR-11 arXiv collection to test the scalability of Tangent-3; this collection is 174 GB uncompressed, with 8,301,578 documents (arXiv article fragments) and 60 million formulae including isolated symbols.

7.1 Efficiency

Computational Resources and Parameters. 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 (as noted below), *all retrieval times are reported for single threaded processing.* Parallelization of query execution, and parallelizing re-rank scoring over matching formulas could be used to further speed up processing, and additional opportunities for decreasing runtimes are discussed below. All results reported for Tangent-3 in this Section were obtained using the top 100 formula from the core engine (i.e., $k = 100$).

Indexing. As seen in Table 2, index size increases roughly linearly from window sizes 1-4, with end-of-line tuples increasing storage by a constant amount. Adding EOL tuples just for small expressions (Sm-EOL; see Section 4) increases the index size modestly (by less than 500k for Wikipedia, and less than 10 MB for arXiv). The maximum Wikipedia index size is 503.1 MB on disk; in contrast, for the arXiv the maximum index size is 29 GB. For small window sizes storage is much smaller; for $w = 1$ with *No-EOL* and *Sm-EOL*, the index file is just over 64 MB for Wikipedia, and just under 5.4 GB for arXiv; these are much smaller than Tangent-2 (1.3 GB for Wikipedia, and roughly 36 GB for the arXiv dataset [16]). When these index files are loaded into memory, they consume 2 - 2.5 times their space on disk.

Table 2: Index Sizes for NTCIR-11 Collections.

	Index Sizes (*MB*)					
	WIKIPEDIA			ARXIV		
w	No-EOL	Sm-EOL	EOL	No-EOL	Sm-EOL	EOL
1	64.2	64.5	73.7	5,364	5,372	6,179
2	95.6	96.0	105.2	7,568	7,577	8,383
3	128.3	128.7	137.9	9,662	9,671	10,477
4	161.3	161.7	170.9	11,587	11,596	12,402
All	493.5	493.9	503.1	28,225	28,234	29,040

For the arXiv data, it took 43 hours to pre-process the documents (using 10 processes), and at most an additional 3.5 hours to generate the index (when $w = All$ and end-of-line tuples are included) using a single process. Wikipedia was much faster, requiring 260 seconds for preprocessing, and at most 95 seconds for index creation. As our document pre-processor is implemented in Python, we believe that a faster implementation (e.g., in C++) could reduce run times by a factor of 4-10 in both cases.

Retrieval Times. We ran the 100 NTCIR-11 Wikipedia formula queries over the large NTCIR-11 arXiv collection to test retrieval speed for the core engine. Larger window sizes increase index entries (see Table 2) and lead to longer query execution times. For example, when EOL tuples are ignored, retrieval times in milliseconds (given as (μ, σ, median)) increase from (372.95, 1649.24, 90.40) for $w = 1$ to (1932.41, 9332.54, 281.64) for $w = All$. Including EOL pairs has a much larger effect on performance: for $w = 1$, retrieval times increase to (4334.86, 16368.06, 584.94). Again using ($w = 1$, EOL), if we turn off the core optimizations O_1-O_5, the average retrieval time doubles and the standard deviation increases (9239.92, 23776.99, 681.79). Using the core optimizations again, for $w = 1$ with Sm-EOL retrieval times increase only a small amount over when no EOL pairs are used (435.55, 1626.48, 111.39).

When running the queries on the smaller NTCIR-11 Wikipedia collection, Tangent-2 requires 8 *minutes* to execute the 100 test queries using a parallelized index with nine sub-indices on Amazon Web Services [16]. In contrast, using a single process and the slowest configuration on our system ($w = All$, EOL), Tangent-3 requires only 8.48 *seconds* without re-ranking, and 106.14 seconds with re-ranking; in the fastest condition ($w = 1$, no EOL) this reduces to 0.57 seconds without re-ranking and 78.03 seconds with re-ranking.

Re-ranking times are consistent across w and *EOL* settings because the number of formulae re-ranked is fixed at $k = 100$. For the Wikipedia corpus, re-rank times are $(775, 3562, 72)$ milliseconds. The mean is skewed by a small number of outliers: for one query re-ranking takes 46 seconds, with retrieval from the core taking only 1.7 seconds ($w = All$, EOL). This query expression is very large (*Query 52*), with 16 wildcard symbols producing large candidates with many possible unifications. Re-ranking can be accelerated by recoding from Python to C++.

7.2 Effectiveness

The NTCIR-11 Wikipedia benchmark [18] includes 100 queries for measuring specific-item retrieval performance. Queries are associated with a single target formula in a specific document, ignoring identical formulae appearing within the same or different documents. These 100 queries are split

Table 3: NTCIR-11 Wikipedia Formula Retrieval Benchmark Results. 100 Queries: 65 Constant, 35 with wildcards (Variable). Metrics: % Recall@k for targets ($k = 10,000$) and Mean Reciprocal Rank (MRR in %).

	Recall@k						MRR					
	Documents			Formulae			Documents			Formulae		
System	Total	Const	Var	Total	Const	Var	Total	Const	Var	Total	Const	Var
TUW Vienna †	**97**	*100	91	**93**	98	83	**80**	80	79	***82**	86	75
NII Japan †	**97**	98	*94	**94**	97	89	**74**	79	74	**72**	*87	63
Tangent-2 ○	**88**	91	83	**78**	78	77	**70**	68	75	**67**	65	72
Tangent-3 Core ○												
w=1 No-EOL	**95**	97	91	**95**	97	91	**79**	80	77	**76**	76	76
Sm-EOL	**97**	*100	91	**97**	*100	91	**80**	*82	77	**77**	78	76
EOL	***98**	*100	*94	***98**	*100	*94	**81**	*82	79	**77**	78	76
w=All No-EOL	**95**	97	91	**95**	97	91	**79**	80	78	**76**	76	76
Sm-EOL	**97**	*100	91	**97**	*100	91	**80**	*82	78	**78**	78	76
EOL	**97**	*100	91	**96**	*100	89	**81**	*82	78	**77**	78	75
Tangent-3 Re-rank ○												
w=1 No-EOL	**95**	97	91	**95**	97	91	**80**	80	*82	**77**	76	*80
Sm-EOL	**97**	*100	91	**97**	*100	91	***82**	*82	*82	**79**	78	*80
EOL	***98**	*100	*94	***98**	*100	*94	***82**	*82	*82	**79**	78	*80
w=All No-EOL	**95**	97	91	**95**	97	91	**80**	80	80	**77**	76	77
Sm-EOL	**97**	*100	91	**97**	*100	91	**81**	*82	80	**78**	78	77
EOL	**97**	*100	91	**96**	*100	89	***82**	*82	*82	**78**	78	78

†: uses Operator Tree (OT) formula representation (Content MathML)
○: uses Symbol Layout Tree (SLT) formula representation (Presentation MathML)

into 65 *Constant* queries containing no wildcards and 35 *Variable* queries containing wildcards.

Search results are returned as a ranked list of (*documentId*, *formulaId*) pairs. *Document-centric* results are computed using the list of document identifiers in their order of appearance after removing duplicates. *Formula-centric* results are computed using the complete ranked list of matches.

Systems are evaluated using two metrics. First, by the *recall* (i.e., percentage of targets located at rank 10,000 or less) and second by the *mean reciprocal rank* (MRR) over all queries (assigning zero when not found). Note that a reciprocal rank of 50% indicates that on average, the target formula appears at rank two.

Summary of Results. At the top of Table 3, NTCIR-11 results from the two best systems and Tangent-2 are shown as fractions rounded to the nearest percentage [18]. A summary of participating systems is available [2]. The first two systems are *tree-based* approaches applied to Operator Trees encoded in Content MathML. As described earlier, Tangent-2 is a *spectral* approach, using symbol pair-based retrieval over Symbol Layout Trees encoded in Presentation MathML. TUW Vienna indexes individual symbols and linearized subexpressions. NII represents sub-expressions in Operator Trees by hash code sets (see Section 2).

Tangent-3 results are shown at the bottom of Table 3, split into core and reranked results. We include combinations of window sizes $w = \{1, 2, 3, 4, All\}$ (where *All* is all tuples) and End-of-Line symbol indexing settings (No-EOL, Small-EOL, EOL).

Using re-ranking, $w = 1$ and all EOL tuples, Tangent-3 obtains the highest document and formula recall (both 98% vs. 97% and 94%), the highest Variable (wildcard) formula MRR (80%) and the highest document MRR (82%) to date. In all Tangent-3 conditions, the mean rank of a target formula is just above the middle of ranks one and two (i.e., higher than an MRR of 75%). This is interesting, as previously the strongest results make use of Operator Trees rather than SLTs for retrieval. This supports the idea that choice of formula representation may be less important than the choice of primitives for retrieval. The higher formula recall obtained by Tangent-3 may be due to matching symbol pairs rather than complete subtrees (see Section 2).

Relative to Tangent-2, recall is increased 10% for documents, and 20% for formulae. MRR values are also 10% higher in Tangent-3 (all conditions) than Tangent-2.

Using *Sm-EOL* provides state-of-the-art performance comparable to *EOL*, but uses a smaller index size with faster retrieval times. With *Sm-EOL* we obtain document and formula recall of 97% (a reduction of only 1%) with unchanged MRR values. For the same parameter settings, formula MRR is 80% (the best reported value is 82%).

Window Size. Window size had little effect on performance, and so for space we show only results for $w = 1$ and $w = All$ in Table 3. Interestingly, recall in the top-10,000 actually decreases slightly when $w = All$ is used, missing two additional formula with wildcards (Var) and one document relative to $w = 1$. This may be because of noise introduced by the larger number of tuples, which may match anywhere in a candidate formula. Window size had no effect on MRR for queries without wildcards (*Const*), but we again see a small decrease for queries with wildcards for $w = All$.

End-of-Line Symbols. For $w = 1$, adding all EOL tuples increases the number of formulae retrieved by three (e.g., the query 's' produces no symbol pairs and thus requires an EOL tuple if it is to be matched). *Sm-EOL* gave fast queries and also retrieved two of the three formulae missed when EOL tuples are omitted. For $w = All$ using all EOL tuples decreases formula recall slightly relative to *Sm-EOL* (missing one query with wildcards), perhaps because of the numerous matches obtained using $w = All, EOL$. Adding the *Sm-EOL* tuples increases document and formula MRR slightly for non-variable queries.

Re-ranking. The re-ranker does not affect recall for Tangent-3, as the re-ranker only reorders hits returned by the core. Exact matches are represented well by the Dice coefficient over tuples, particularly for concrete queries without wildcards. However, the re-ranker does improve wildcard (variable) formula MRR by 4% for queries with wildcards

Table 4: Likert Rating $\mu(\sigma)$ for Top-10 NTCIR-11 Wikipedia Hits (21 participants, 10 queries).

	Rank/Position in Top-10 Hits									
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
$w = 1$	4.54 (0.78)	3.79 (1.16)	3.48 (1.31)	3.23 (1.30)	2.83 (1.25)	2.94 (1.22)	2.65 (1.19)	2.78 (1.21)	2.78 (1.21)	2.85 (1.25)
$w = 2$	4.54 (0.78)	3.71 (1.22)	3.48 (1.30)	3.16 (1.28)	2.90 (1.26)	2.93 (1.20)	2.85 (1.25)	2.57 (1.18)	2.74 (1.22)	2.80 (1.13)
$w = All$	4.54 (0.78)	3.78 (1.19)	3.59 (1.23)	3.27 (1.16)	2.98 (1.25)	2.92 (1.23)	2.80 (1.17)	2.98 (1.24)	2.92 (1.21)	2.87 (1.17)
best	4.54 (0.79)	4.03 (1.07)	3.75 (1.12)	3.49 (1.13)	3.31 (1.20)	3.15 (1.16)	3.02 (1.17)	2.94 (1.17)	2.85 (1.19)	2.77 (1.19)

for $w = 1$, *Sm-EOL*. (80% with re-ranking, vs. 76% without). This is 5% higher than the best previous MRR results for formula MRR (75%).

On average, only 4.16 of the top-10 results (with standard deviation of 2.61) remain in the top-10 after re-ranking by MSS. MSS will not re-order exact or very nearly exact matches, hence constant queries have the same MRR values as the core engine, but the reranker is able to improve MRR values for variable queries where the core implements approximate wildcard handling.

Without re-ranking, the core engine produces comparable results using $k = 100$. For the core engine, increasing k produces perfect recall (100%) without substantially increasing query time. The reranker, however, cannot process a large number of results in a reasonable amount of time. The core engine also produces higher document-centric MRR values than the best published results, although limited handling of wildcards causes variable queries to underperform.

We now consider how well MSS-based rankings relate to human perceptions of formula similarity, using the top-10 results from 10 of the NTCIR-11 Benchmark queries.

Human Evaluation: Data. 10 queries were selected using random sampling from the Wikipedia queries. Five contained wildcards, and the other five did not. Some queries were replaced using a new randomly selected query to ensure a diverse set of expression sizes and structures.

To limit the number of hits for participants to evaluate, we chose to consider only $w = 1, 2$, and *All*. We used *No-EOL*, as none of the queries were a single symbol (the smallest query, Q8 is '$\alpha(x)$'), and we wished to observe the effect of window size using actual symbol pairs, without EOL tuples. Using the Wikipedia collection, for the three versions of the core compared ($w = \{1, 2, All\}$, *No-EOL*), we applied reranking to the top-100 hits, and then collected the top-10 hits returned by each query for rating.

Evaluation Protocol. Participants completed the study alone in a private, quiet room with a desktop computer running the evaluation interface in a web browser. The web pages provided an overview, followed by a demographic questionnaire, instructions on evaluating hits, and then familiarization trials (10 hits; 5 for each of two queries). After familiarization, participants evaluated hits for the 10 queries, and finally completed an exit questionnaire. Participants were paid \$10 each at the end of their sessions.

Participants rated the similarity of queries to results using a five-point Likert scale (Very Dissimilar, Dissimilar, Neutral, Similar, Very Similar). It has been shown that presenting search results in an ordered list influences relevance assessments [7]. Instead we presented queries along with each hit in isolation, with query presentation order randomized, and the presentation order for hits was also randomized.

Demographics and Exit Questionnaire. 21 participants (5 female, 16 male) were recruited from the Computing and Science colleges at RIT. Their age distribution was: 18-24 (8), 25-34 (9), 35-44 (1), 45-54 (1), 55-64 (1) and 65-74 (1). Their highest levels of education completed were: Bachelor's (9), Master's (9), PhD (2), and Professional (1). Their reported areas of specialization were: Computer Science (13), Electrical Engineering (2), Psychology (1), Sociology (1), Mechanical Engineering (1), Computer Engineering (1), Math (1) and Professional Studies (1).

In the post-questionnaire, participants rated the evaluation task as Very Difficult (3), Somewhat Difficult (10), Neutral (6), Somewhat Easy (2) or Very Easy (0). They reported different approaches to assessing similarity. Many considered whether operations and operands were of the same type or if two expressions would evaluate to the same result. Others reported considering similarity primarily based on similar symbols, and shared structure between expressions.

Similarity Rating Results. As seen in Table 4, Likert ratings are similar in all conditions. Average ratings increase from the 5th to 1st hits, and are close to the best obtained in the result pools ('best' in Table 4). On average Top-4 hits have some similarity with the query, with average ratings higher than 'Neutral' (3). For large formulae, strong matches for small subexpressions were sometimes perceived as dissimilar, and *exact* matches to the query were often rated as 'Similar' (4). Reading large formulae is difficult and may reduce perceived similarity; we wonder if highlighting matches would have increased these ratings.

The top-5 hits are largely identical across conditions (at least 4/5 match) with the exception of query 60:

$$p = \frac{-x \pm \sqrt{x^{*} - 4(*)(\frac{*}{*} - y)}}{2(\frac{-gx^2}{*})}$$

$w = All$ returns the best rated hits in the top-5, but $w = \{1, 2\}$ return only one of the best-rated matches in the top-5. $w = All$ generates more tuples for matching, compensating for wildcard pairs not being indexed by matching relationships between more distant symbols.

We produce ideal rankings by pooling top-10 hits and then sorting them by average Likert rating. Relative to this ideal, average nDCG@10 values increase and standard deviations decrease with window size ($w = 1$: 0.87 (0.1); $w = 2$: 0.88 (0.08); $w = All$: 0.90 (0.03)). Selecting the best window size for each query improves metrics further (*oracle*: 0.92 (0.02)). Interestingly, the highest nDCG values are obtained for six queries using w = 1; the additional symbol pair matches considered for larger window sizes sometimes lead to very scattered matches being returned in the top-k from the core engine, which $w = 1$ avoids.

Unification works well when most of a candidate matches the query (e.g., as in Table 1). In Tangent-3 all identifiers are unifiable, including greek letters, latin letters, and function names. At times high MSS scores are obtained for candidates with weak Likert ratings due to a large number of unifications, or unifying symbols of different types (e.g., x with π). Unifying or matching symbols associated with operators of different types appears to contradict some ratings (e.g., $a + b$ and $a \leq b$ may be perceived as dissimilar, while $a + b$ and $a - b$ perceived as similar).

To improve unification and MSS scores, one could expand the set of symbol and operator types in the SLT model, annotate SLTs with semantic information [24], or modify MSS to be computed using matches in both Operator Tree and SLT formula representations. It should be straight-forward to adapt Tangent-3 to work with Operator Trees.

8. CONCLUSION

We have presented a new two-stage cascaded retrieval model for appearance-based formula retrieval. The Symbol Layout Tree and symbol pair retrieval models used in the first stage out-performs earlier versions, being more efficient in space and time. The second stage re-ranks the top-k matches by Maximum Subtree Similarity (MSS), producing state-of-the-art results for the NTCIR-11 Wikipedia formula retrieval task. Human similarity ratings agree substantially with formula rankings produced using MSS.

For future work, more human experiments are needed to identify features that affect the perception of formula similarity for mathematical experts and non-experts. Retrieval efficiency may be improved by compressing dictionaries and postings lists, and by using an implementation of weak-AND [4] or a more fine-grained implementation of max-score [21]. Retrieval effectiveness can be improved through changes to the SLT, unification models, and the MSS function and scoring vector. Additional opportunities include allowing wildcards to match subexpressions rather than single symbols, implementing additional query functionality in the engine, incorporating textual features and context [24], and integrating Tangent-3 with keyword search.

Acknowledgements

This material is based upon work supported by the National Science Foundation (USA) under Grant Nos. IIS-1016815 and HCC-121880. Financial support from the Natural Sciences and Engineering Research Council of Canada under Grant No. 9292/2010, Mitacs, and the University of Waterloo is also gratefully acknowledged.

9. REFERENCES

[1] A. Aizawa, M. Kohlhase, and I. Ounis. NTCIR-10 math pilot task overview. In *NTCIR*, 2013.

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

[3] J. L. Bentley. An almost optimal algorithm for unbounded searching. *Inf. Process. Lett.*, 5(3):82–87, 1976.

[4] A. Z. Broder, D. Carmel, M. Herscovici, A. Soffer, and J. Zien. Efficient query evaluation using a two-level retrieval process. In *CIKM*, 2003.

[5] T. W. Cole, I. Debauchies, K. M. Carley, J. L. Klavans, Y. LeCun, M. Lesk, C. A. Lynch, P. Olver, J. Pitman, and Z. J. Xia. *Developing a 21st Century Global Library for Mathematics Research.* The National Academies Press, Washington, DC, 2014.

[6] P. Graf. Substitution tree indexing. In *RTA*, 1995.

[7] Z. Guan and E. Cutrell. An eye tracking study of the effect of target rank on web search. In *SIGCHI*, 2007.

[8] F. Guidi and C. S. Coen. A survey on retrieval of mathematical knowledge. In *CICM*, 2015.

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

[10] S. Kamali and F. W. Tompa. A new mathematics retrieval system. In *CIKM*, 2010.

[11] S. Kamali and F. W. Tompa. Structural similarity search for mathematics retrieval. In *CICM*. 2013.

[12] M. Kohlhase and I. Sucan. A search engine for mathematical formulae. In *AISC*, 2006.

[13] X. Lin, L. Gao, X. Hu, Z. Tang, Y. Xiao, and X. Liu. A mathematics retrieval system for formulae in layout presentations. In *SIGIR*, 2014.

[14] B. R. Miller and A. Youssef. Technical aspects of the digital library of mathematical functions. *Ann. Math. Artif. Intell.*, 38(1):121–136, 2003.

[15] 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.

[16] 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*, 2014.

[17] P. Pavan Kumar, A. Agarwal, and C. Bhagvati. A structure based approach for mathematical expression retrieval. In *MIWAI*. 2012.

[18] M. Schubotz. Challenges of mathematical information retrieval in the NTCIR-11 Math Wikipedia Task. In *SIGIR*, 2015.

[19] P. Sojka and M. Líška. Indexing and searching mathematics in digital libraries. In *CICM*, 2011.

[20] D. Stalnaker and R. Zanibbi. Math expression retrieval using an inverted index over symbol pairs. In *DRR*, 2015.

[21] H. Turtle and J. Flood. Query evaluation: strategies and optimizations. *Inf. Process. Manage.*, 31(6):831–850, 1995.

[22] C. J. van Rijsbergen. *Information Retrieval.* Butterworths, 2nd edition, 1979.

[23] L. Wang, J. Lin, and D. Metzler. A cascade ranking model for efficient ranked retrieval. In *SIGIR*, 2011.

[24] 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.

[25] K. Wangari, R. Zanibbi, and A. Agarwal. Discovering real-world use cases for a multimodal math search interface. In *SIGIR*, 2014.

[26] P. Willett. Document retrieval experiments using vocabularies of varying size. II. Hashing, truncation, digram and trigram encoding of index terms. *J. Documentation*, 35:296–305, 1979.

[27] A. Youssef. Roles of math search in mathematics. In *Proc. Math. Knowl. Manage.*, pages 2–16. 2006.

[28] R. Zanibbi and D. Blostein. Recognition and retrieval of mathematical expressions. *Int. J. Document Anal. Recognit.*, 15(4):331–357, 2012.