Structural Similarity Search for Formulas using Leaf-Root Paths in Operator Subtrees

Wei Zhong ✉ and Richard Zanibbi

wxz8033@rit.edu, rlaz@cs.rit.edu
Rochester Institute of Technology

Abstract. We present a new search method for mathematical formulas based on Operator Trees (OPTs) representing the application of operators to operands. Our method provides (1) a simple indexing scheme using OPT leaf-root paths, (2) practical matching of the K largest common subexpressions, and (3) scoring matched OPT subtrees by counting nodes corresponding to visible symbols, weighting operators lower than operands. Using the largest common subexpression (K=1), we outperform existing formula search engines for non-wildcard queries on the NTCIR-12 Wikipedia Formula Browsing Task. Stronger results are obtained when using additional subexpressions for scoring. Without parallelization or pruning, our system has practical execution times with low variance when compared to other state-of-the-art formula search engines.

Keywords: Mathematical Information Retrieval · formula search · similarity search · subexpression matching

1 Introduction

Mathematical Information Retrieval (MIR [5, 21]) requires specialized tasks including detecting and recognizing math in documents, math computation and knowledge search (e.g., in Wolfram Alpha), and similarity search for math expressions. Formula search engines are useful for looking up unfamiliar notation and math question answering.

Traditional text search engines are unaware of many basic characteristics of math formulas. Key problems in math formula similarity search include:

- How do we represent math formulas for search?
- How do we measure math formula similarity?
  - Structural similarity: Common subexpression(s), operator commutativity and operator associativity.
  - Symbol set similarity: Being aware of unifiable/interchangeable elements (e.g., $(1+1/n)^n$ and $(1+1/x)^x$), while still distinguishing $e = mc^2$ from $y = ax^2$; weighting identical symbols appropriately.
  - Semantic similarity of mathematical formulas, including equivalent formulas (e.g., x^{-1} and $1/x$).
- What is a good trade-off between feature-based matching and costly structure matching, to identify similar formulas efficiently and effectively?

We present a new formula search engine based on Operator Trees (OPTs). OPTs represent the semantics of a formula, in terms of the application of operators to operands in an expression. We adapt the leaf-root path indexing of all subtrees used in MCAT [8], where these paths act as the retrieval units or "keywords" for formulas. MCAT uses additional encodings (e.g., Presentation MathML) that we do not consider in this work. Our scoring function generalizes subtree scoring methods which only consider single best matched tree such as the Maximum Subtree Similarity (MSS) of Tangent [22]. To the best of our knowledge, our model is the first using multiple common subexpressions to score formula hits. Our approach has achieved usable execution times using a single process without any dynamic pruning applied so far, and produces state-of-the-art results for non-wildcard queries in the NTCIR-12 Wikipedia Formula Browsing Task [20]. Our system is available for download.[1]

2 Related Work

There are two major approaches to math representation and indexing, *Text-based* and *Tree-based* [21]. Text-based approaches apply traditional text search engines, converting formulas to canonically ordered text strings with index augmentation [11,12,14] to deal with operator commutativity, operator associativity, and subexpression matching. Tree-based approaches index formulas directly from hierarchical representations of appearance and semantics. In the recent NTCIR-12 MIR tasks [20], tree-based MIR systems achieve the best accuracy.

Tree representations are primarily divided into SLTs (Symbol Layout Trees) and OPTs (Operator Trees). SLTs capture appearance based on the arrangement of symbols on writing lines (i.e., topology). OPTs represent semantics: internal nodes represent operators, and leaves represent operands. SLTs capture appearance with few ambiguities, and require few spatial relationships to represent structure. However, they cannot capture semantic equivalences, operator commutativity, or operator associativity. By representing operations explicitly, visually distinct but mathematically equivalent formulas have identical OPTs (e.g., $\frac{1}{x}$ and $1/x$) and operator commutativity is captured explicitly (e.g., allowing us to determine that $1 + x^2$ and $x^2 + 1$ are equivalent). OPT construction requires an expression grammar, which for real-world data needs to accommodate ambiguous and malformed expressions (e.g., unpaired parentheses). In our work, we parse LaTeX formulas into OPTs.

Measuring similarity using both SLTs and OPTs, Gao et al. [9] uses *sibling patterns* extracted from semi-OPTs (which do not identify implicit multiplication). They extract "level content" from OPTs, identifying the depth at which a sibling pattern appears. Arguments are represented as wildcards. For example, at level one, $(x + y)z$ is represented by $(*) \times *$, at level two by $(*)$, and then recursively $* + *$. Extracted $(pattern, level)$ tuples are used for search.

Some systems use leaf-root paths extracted from formula trees (*vertical paths*). Hijikata et al. [6] use leaf-root paths in Content MathML (a form of OPT), but

[1] Source code: https://github.com/approach0/search-engine/tree/ecir2019

to be considered as a match, a candidate path and query path must be identical. Similarly, OPMES [23,24] requires complete leaf-root query paths to match some *prefix* of candidate leaf-root paths. This allows for retrieving partially matched candidate expressions, but still requires a complete match in the query expression. Yokoi and Aizawa [19] adopt a more flexible approach, using all possible *subpaths* of leaf-root paths, and do not require all leaf-root paths in the query to be matched. The MCAT system of Kristianto et al. [8] combines path features (both ordered paths and unordered paths) generated from leaf-root paths, and also uses sibling patterns for search.

Stalnaker et al. [15] use *symbol pairs* extracted from SLTs, where node to ancestor symbol pairs along with their relative position in an SLT are used for search. Later Davila et al. [4,22] use labeled paths between symbols and generate symbol pairs falling within a given maximum path length (window size). In their approach, expressions are a candidate as long as they share one symbol pair with the query. This method has high recall due to low granularity in the search unit; however, it produces a large candidate set with few structural constraints, thus Davila et al. [22] introduce a second stage to rerank by structural similarity. They find an alignment between query and candidate formulas maximizing a similarity score with $O(|T_d||T_q|^2 \log T_q)$ time complexity, where T_q and T_d are query and candidate trees. Later they apply similar techniques in both SLTs and OPTs, and combine results to obtain better results in the *Tangent-S* system [4].

There are also techniques that capture structural similarity more precisely, e.g., Kamali et al. [7] use tree edit distance to measure differences between MathML DOM trees for formula similarity (in SLTs), however, the computation has non-linear time complexity in terms of expression size. How to determine the costs of edit operations to reflect similarity remains an open problem. There have been studies on similarity distance metrics that do not depend on edit operations, and subgraph-based graph similarity metrics have been explored for a long time in the pattern recognition literature [2].

3 Methodology

In our context, matching subexpressions means finding subtrees that are structurally identical and the matched nodes have the same tokens (we will use uppercase words to indicate tokens, e.g. variables x, y will both be VAR tokens after tokenization). To formally define our structure matching approach, we incorporate the graph/subtree isomorphism definition [13] and add a few definitions based on the *formula subtree* [23]. In addition to general subtree isomorphism, a formula subtree (indicated by $\preceq_l$) requires leaves in a subtree to be also mapped to leaves in the other tree.

Definition 1. *A common formula subtree of two formula trees T_q and T_d consists of two corresponding formula subtrees $\hat{T}_q$ of T_q and $\hat{T}_d$ of T_d where they are isomorphic and they are subgraphs of T_q and T_d respectively. Let CFS(T_q, T_d) denote the set of all such common formula subtrees of T_q, T_d, i.e.,*

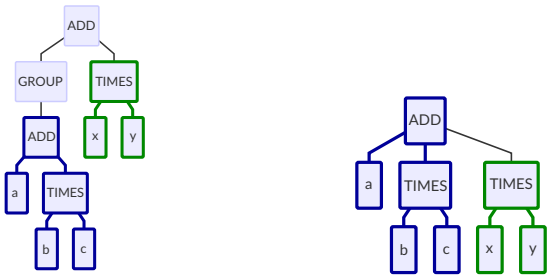


Fig. 1. Common formula forest in OPT. Left to right: $(a + bc) + xy$ and $a + bc + xy$.

$CFS(T_q, T_d) = \{\hat{T}_q, \hat{T}_d : \hat{T}_q \preceq_l T_q, \hat{T}_d \preceq_l T_d, \hat{T}_q \cong \hat{T}_d, \hat{T}_q \subseteq T_q, \hat{T}_d \subseteq T_d\}$ where "$\cong$" and "$\subseteq$" indicate graph isomorphism and subgraph relation respectively.

Similar to common forest definitions [18], we define a form of disjoint common subtrees to describe multiple subexpression matches. Figure 1 illustrates two matching common subexpressions ($a+bc$ and xy), with the matches highlighted in blue and green. We call these matches a *common formula forest*. It consists of common formula subtree(s) identified by $(\hat{T}_q^i, \hat{T}_d^i)$ as defined below.

Definition 2. *A set of common formula subtrees π is called a common formula forest of two formula trees T_q and T_d,*

$$\pi = \{(\hat{T}_q^1, \hat{T}_d^1), (\hat{T}_q^2, \hat{T}_d^2), ...(\hat{T}_q^n, \hat{T}_d^n)\} \in \Pi(T_q, T_d) \tag{1}$$

iff for $i = 1, 2, ...n$:
(1) $\hat{T}_q^i, \hat{T}_d^i \in CFS(T_q, T_d)$
(2) $\hat{T}_q^1, \hat{T}_q^2, ...\hat{T}_q^n$ are disconnected, and $\hat{T}_d^1, \hat{T}_d^2, ...\hat{T}_d^n$ are disconnected.
where $\Pi(T_q, T_d)$ denote all possible common formula forests of T_q and T_d.

For our structural similarity metric, we want to find the "largest" common formula forest to represent the most similar parts of two math expressions. In order to define "large" generally, our similarity scoring formula between two formula trees is parameterized by some scoring function γ of $\pi \in \Pi(T_q, T_d)$.

Definition 3 (General multi-tree structure similarity). *The formula tree similarity of T_q and T_d given scoring function γ is*

$$\Gamma_\gamma(T_q, T_d) = \max_{\pi \in \Pi(T_q, T_d)} \gamma(\pi) \tag{2}$$

Intuitively, we choose the number of matched tree nodes to measure matched "size". Since the similarity contribution of different nodes (i.e. operands and operators) may be non-uniform, we propose using the similarity scoring function γ defined by

$$\gamma(\pi) = \sum_{(\hat{T}_q^i, \hat{T}_d^i) \in \pi} \beta_i \cdot \Big(\alpha \cdot \text{internals}(\hat{T}_d^i) + (1 - \alpha) \cdot \text{leaves}(\hat{T}_d^i)\Big) \tag{3}$$

where internals(T) is the number of internal nodes/operators in T, leaves(T) is the number of leaves/operands in T, and $\alpha \in [0, 1]$ defines the contribution weight of operators. $\beta_i \geq 0$ are the contribution weights for different matched subexpressions. For the convenience of later discussion, we refer to trees in equation (3) indexed by i, e.g. $\hat{T}_d^i$, as the *i-th widest* match (in terms of number of matched leaves) in π. We set $\beta_1 \geq \beta_2 \geq ... \geq \beta_n$ in order to weight "wider" subexpressions higher. And in practice, it is wasteful to compute all terms in equation (3), if we assume the largest K matched subexpressions cover most of the total matched size, we obtain an approximate scoring function where only a subset of terms in equation (3) are computed by fixing $\beta_i = 0$ for $i \geq \min(n, K)$.

3.1 Subexpression matching

Valiente [17] has shown $O(m+n)$ time complexity for computing similar multiple-tree similarity, but this requires matching vertex out degree (i.e., complete subtrees), which is too strict for retrieval, e.g., $a+b$ will not match $a+b+c$ because the operand number does not agree. To practically compute formula tree similarity, we propose an greedy algorithm.

Using paths as units, it is easier to count matched operands than operators (each matched operand is identified by a matched path), so we first greedily find a common formula forest π^* that consists of the widest common formula subtree (in terms of matched operands), and then calculate the corresponding number of matched operators. In order for π^* to be the optimizer in equation (2), it requires the following assumption.

Assumption 1 *If* $\pi^* = \{(\hat{T}_q^{1*}, \hat{T}_d^{1*}), (\hat{T}_q^{2*}, \hat{T}_d^{2*})...(\hat{T}_q^{n*}, \hat{T}_d^{n*})\} \in \Pi(T_q, T_d)$ *is the maximizer in equation (2) for* $\alpha = 0$ *and* $\beta_1 \gg \beta_2 \gg ... \gg \beta_n$*, then we assume* π^* *is also maximizer in equation (2) for all* $\alpha \neq 0$ *and all* $\beta_1 \geq \beta_2 \geq ... \geq \beta_n$*.*

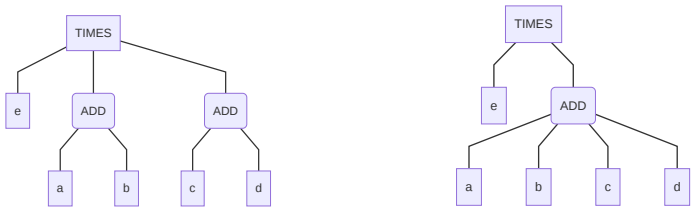
Under this assumption, finding the widest matched subtrees in order will yield our defined formula tree similarity, while in reality, greedily finding widest matched subtrees may not maximize equation (3).

We also want to use paths to test identical structures efficiently. Let $\mathcal{P}(T)$ be all leaf-root paths from rooted tree T, and a matching between path sets S_1, S_2 is defined as bipartite graph $M(S_1, S_2, E)$ where E is edges representing assigned matches. In our context, two paths match if they are identical after tokenization (e.g. The OPTs represent $a+b$ and $x+y$ have the same set of tokenized leaf-root path "VAR/ADD"). To compare structure efficiently, we also assume that two subtrees are structurally identical if only their leaf-root paths match:

Assumption 2 *For any tree* T_q, T_d*, let* $S_q = \mathcal{P}(T_q), S_d = \mathcal{P}(T_d)$*, if there exists perfect matching* $M(S_q, S_d, E)$*, then we assume* $T_q \cong T_d$*.*

This assumption does not always hold true (see Figure 2), nevertheless, we expect this to be relatively rare in practice, and it allows us to design a practical algorithm for computing formula tree similarity.

Under Assumptions 1 and 2, it can be shown that if $\text{CFS}(T_q, T_d) \neq \emptyset$, and $S_q^m, S_d^m = \arg\max |E|$ for any matching $M(S_q^m \subseteq S_q, S_d^m \subseteq S_d, E)$, where



$(a+b)\times(c+d)\times e$ $\quad$ $(a+b+c+d)\times e$

Fig. 2. Formulas with identical leaf-root paths, but different structure

$S_q = \mathcal{P}(\hat{T}_q), S_d = \mathcal{P}(\hat{T}_d)$, $\hat{T}_q, \hat{T}_d \in \text{CFS}(T_q, T_d)$ then leaves$(\hat{T}_q^{1*})$ = leaves$(\hat{T}_d^{1*})$ = $|S_q^m| = |S_d^m|$. In other words, we can use leaf-root paths from query and document OPT subtrees to get the number of leaves of the widest matched tree in a common formula forest π^* that maximizes scoring function γ in equation (2). After leaves$(\hat{T}_d^{1*})$ is obtained, we can exclude already matched paths and similarly compute other leaves$(\hat{T}_d^{i*}), i = 2, 3...k$. The process of matching, i.e., finding S_q^m, S_d^m in any $M(S_q, S_d, E)$, can be implemented using bit masks and the output value $|S_q^m|$ does not depend on input order (matching order).

In scoring function (3), we also want the number of operators associated with matched leaves. Adding the number of matched operators in equation (3) helps better assess similarity when assumption 2 fails. Consider the example in Figure 2: only one of the two "ADD" operators on the left tree can match the "ADD" operator on the right. If we count the matched operators correctly, we can differentiate the two expressions in Figure 2. To calculate the number of operators, assume we have found a common formula forest π^* that maximizes function γ, then we go through all subtree pairs (T_q^x, T_d^y) rooted at $x \in T_q, y \in T_d$, and examine if it joins with any pair of matched trees in π^* by looking at whether their leaves intersect. If true, we will count x, y as matched operators if both of them are not marked as matched yet.

Algorithm 1 describes our matching procedure in detail. In experiments, we found that counting only visible operators improves results in most cases. This is because some internal OPT nodes do not appear in the rendered expression, so counting them will bias the similarity measurement in our model. In particular, we do not count SUBSCRIPT and SUPERSCRIPT operator nodes. Algorithm 1 avoids counting those nodes by consulting a pre-built "visibility" mapping for operators (i.e., the VISIBLE function).

3.2 Indexing and retrieval

At the indexing stage, every math expression in the corpus is parsed into an OPT T_d. For all internal (operator) nodes n in T_d, we extract all leaf-root paths of T_d^n rooted at n. This path set $S = \bigcup_{n \in T_d} \mathcal{P}(T_d^n)$ is *tokenized* (e.g. operand

Algorithm 1 Formula tree matching algorithm

Let (S_q^m, S_d^m) be the maximum matching path set of given path set (S_q, S_d).
Define $\ell(S)$ to be all the leaf nodes (equivalently, path IDs) for path set S.
function operandMatch(Q^m, D^m, L, k, leavesCounter)
  $Q^X := \{\}, D^X := \{\}$ ▷ Excluded path set
  for $i < k$ **do**
    $Q^{\max}$, $D^{\max}$, max := 0 ▷ Best matched tree records
    for (S_q, S_d) from L **do**
      if $Q^X \cap \ell(S_q) = \emptyset$ and $D^X \cap \ell(S_d) = \emptyset$ **then** ▷ Disjoint tree pairs
        if $|S_q^m| >$ max **then** ▷ Greedily find widest matches
          max := $|S_q^m|$
          $Q^{\max}, D^{\max} := \ell(S_q^m),\ \ell(S_d^m)$
    if max > 0 **then**
      $Q^X := Q^X \cup Q^{\max}$
      $D^X := D^X \cup D^{\max}$
      $Q_i^m, D_i^m := Q^{\max},\ D^{\max}$
      leavesCounter[i] = max
    else ▷ No more possible operand matchings
      break
  return Q^m, D^m, leavesCounter
function operatorMatch(Q^m, D^m, L, k, operatorsCounter)
  Let Q^{map}, D^{map} be maps of matched internal nodes, initially empty.
  for (S_q, S_d) from L **do**
    for $i < k$ **do**
      if $Q_i^m \cap \ell(S_q) \neq \emptyset$ and $D_i^m \cap \ell(S_d) \neq \emptyset$ **then** ▷ Joint tree pairs
        Let n_q, n_d be the root-end nodes of S_q, S_d respectively.
        if $Q^{\text{map}}[n_q], D^{\text{map}}[n_d]$ are both empty **then**
          $Q^{\text{map}}[n_q], D^{\text{map}}[n_d] := n_d, n_q$
          if visible(n_q) **then**
            operatorsCounter[i] := operatorsCounter[i] + 1
        break
  return operatorsCounter
function formulaTreeMatch(T_q, T_d, k)
  for $i < k$ **do**
    $Q_i^m := \{\}, D_i^m := \{\}$ ▷ Matched path set for *i-th* largest matched tree
    leavesCounter[i] := 0
    operatorsCounter[i] := 0
  L := List of (S_q, S_d) where $S_q, S_d \in \mathcal{P}(T_q^x), \mathcal{P}(T_d^y)$ for each node $x \in T_q, y \in T_d$.
  Q^m, D^m, leavesCounter := operandMatch(Q^m, D^m, L, k, leavesCounter)
  operatorsCounter := operatorMatch(Q^m, D^m, L, k, operatorsCounter)
  return leavesCounter[i], operatorsCounter[i] for $i = 1, 2, ...k$

symbols a, b, c are tokenized into VAR, operators fraction and division ($\div$) are tokenized into FRAC) by pre-defined OPT parser rules[2] to allow results from unification/substitution and boost recall. Each unique tokenized path is associated with a posting list, where the IDs of expressions containing the path are stored. The IDs of endpoint nodes (leaf and operator) of each path are also stored in the posting lists. This allows the structure of matched subexpressions to be recovered from hit paths at the posting list merge stage.

During query processing, a query expression tree T_q is decomposed in the same way. Posting lists associated to its tokenized path set are retrieved and merged. During merging, we examine the matched paths from a document expression one at a time, input as list L in Algorithm 1 and compute the structural matching. Then we compute the overall similarity score (considering both struc-

[2] Our expression grammar has roughly 100 grammar rules and 50 token types.

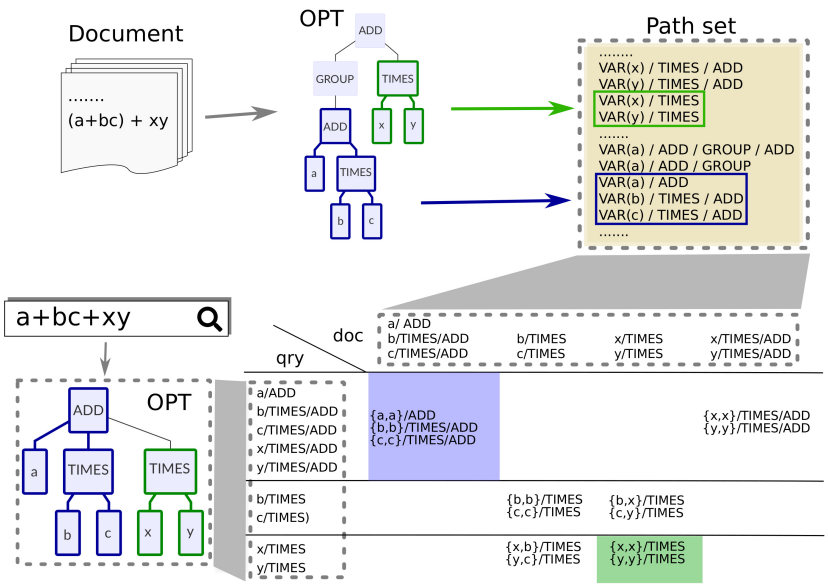


Fig. 3. Illustration of path retrieval and subexpression matching. After matching the largest common subexpression, i.e., $a + bc$ (in blue), the remaining largest disjoint common subexpression is xy (in green).

tural and symbolic similarity) as follows:

$$\frac{S_{\text{st}} S_{\text{sy}}}{S_{\text{st}} + S_{\text{sy}}} \left[(1-\theta) + \theta \frac{1}{\log(1 + \text{leaves}(T_d))} \right], \qquad \theta \in [0,1] \tag{4}$$

where structure similarity S_{st} is normalized formula tree similarity

$$S_{\text{st}} = \begin{cases} \frac{\Gamma_\gamma(T_q, T_d)}{\text{leaves}(T_q)} & \text{if } \alpha = 0 \\ \\ \frac{\Gamma_\gamma(T_q, T_d)}{\text{leaves}(T_q) + \text{internals}(T_q)} & \text{if } \alpha \neq 0 \end{cases} \tag{5}$$

and S_{sy} is the normalized *operand* symbol set similarity score $y \in [0,1]$ produced from the Mark-and-Cross algorithm [23] which scores exact symbol matches higher than unified symbol matches:

$$S_{\text{sy}} = \frac{1}{1 + (1-y)^2} \tag{6}$$

The final scoring function (4) is a F-measure form of structure similarity and symbol set similarity combination, partially (θ) penalized by document math formula size measured by total number of its operands, i.e. leaves(T_d).

We can calculate the maximum matchings using bit operations if we assume the number of operands and the number of subexpressions that one math expression can have are less than a constant. And because the number of elements in L is $|T_q| \times |T_d|$, after maximum matchings are obtained, Algorithm 1 has overall time complexity $O(k|T_q||T_d|)$.

Figure 3 illustrates the path retrieval and subexpression matching process. Notice that the operands/leaves are not shown as tokenized in some places, so that we can identify which paths are matched. Algorithm 1 can be visualized using a table (as shown at bottom right), where pairs of matched {query, document} paths are inserted into corresponding cells when we merge posting lists (e.g. b/TIMES and x/TIMES are matched, indicated by {b, x}/TIMES). Each table cell represents an element of input list L in Algorithm 1. At the end of the algorithm, we obtain the highlighted cells with the largest number of matched leaves. Then the matched operators are counted for highlighted cells. Finally, we calculate the structural similarity score S_{st} from the number of operators and operands associated with each matched subexpression, and the symbol set similarity score S_{sy} from matched operands symbolic differences, then plug these into equation (4) to obtain the final similarity score for ranking.

4 Evaluation

We evaluate our system using the NTCIR-12 MathIR Wikipedia Formula Browsing Task (in the following, use *NTCIR-12* for short), which is the most current benchmark for isolated formula retrieval. The dataset contains over 590,000 math expressions taken from English Wikipedia. We consider all the 20 non-wildcards queries in NTCIR-12. During the task, pooled hits from participating systems were each evaluated by two human assessors. Assessors score a hit from highly relevant to irrelevant using 2, 1, or 0. The final hit relevance rating is the sum of the two assessor scores (between 0 and 4), with scores of 3 or higher considered *fully relevant* and other scores of 1 and higher considered *partially relevant*. We use *bpref* [1] on top-1000 results as our primary effectiveness metric because our system does not contribute to pooling, and bpref is computed over only judged hits. In addition to *standard* Precision@K values, we compute Precision@K metrics using only judged hits (*condensed*), and provide *upper bound* values by treating unjudged hits as relevant [10].

First, we explored the impact of different parameter values using up to 3-tree matching ($K = 3$). The θ parameter for penalizing overly large formulas is fixed at 0.05. Figure 4 shows representative parameter values that we have tried. We started with a single tree match (first four rows in table at left), finding that weighting operator symbol matches slightly lower than operands ($\alpha = 0.4$) have produced the best results (we tried α in $[0, 1]$ using an increment of 0.1). We then fixed $\alpha = 0.4$, $\sum_i^n \beta_i = 1$ and tried uniform weights (rows 5–7) and non-uniform weights for two trees (rows 8–10) and three trees (rows 11–13). We examined uniform β weights for multiple matches from K = 1 to 3. For non-uniform weights in two-trees, we consider β_1 in $[0.5, 0.99]$ using increments

Run	Parameters				Bpref score	
	α	β_1	β_2	β_3	Partial	Full
opt-only	1.0	1.00			0.5304	0.6448
opd-opt-a6	0.6	1.00			0.5416	0.6498
opd-opt-a4	0.4	1.00			0.5899	0.6662
opd-only	0.0	1.00			0.5153	0.6586
uni-beta-1	0.4	1.00			0.5899	0.6662
uni-beta-2	0.4	0.50	0.50		0.5642	0.6481
uni-beta-3	0.4	0.34	0.33	0.33	0.5188	0.6423
2-beta-98	0.4	0.98	0.02		**0.5951**	0.6696
2-beta-80	0.4	0.80	0.20		0.5888	0.6671
2-beta-60	0.4	0.60	0.40		0.5856	0.6583
3-beta-90-4	0.4	0.90	0.06	0.04	0.5950	**0.6726**
3-beta-75-2	0.4	0.75	0.15	0.10	0.5879	0.6695
3-beta-60-3	0.4	0.60	0.25	0.15	0.5900	0.6655

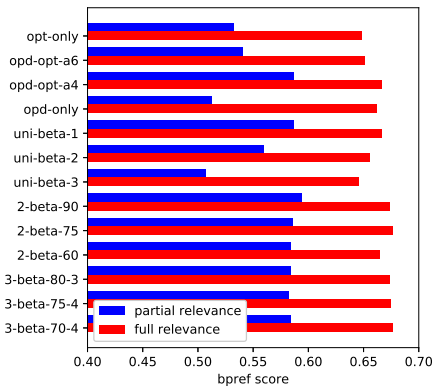


Fig. 4. Relevance results from representative parameter values (table and bar graph).

of 0.05 or 0.01; for three-tree matching, we considered β_1 in $[0.5, 0.95]$ and β_2 in $[0.05, 0.45]$ using increments of 0.05. Figure 4 shows uniform weights generally yield worse results than non-uniform ones. And two runs from non-uniform weights when K=2 and 3 obtain the best partial and full relevance scores respectively. This observation is intuitive because our setting non-uniform weights emphasizes larger subexpressions, which arguably have more visual impact.

Second, to illustrate the effect of matching multiple subexpressions, Figure 5 shows changes in fully relevant bpref scores for different queries, when changing the maximum number of matched trees (K) with uniform weights. Figure 5 omits queries whose score remains unchanged or differs negligibly across values of K. We can observe that different queries have different behaviours as K increases, e.g., introducing secondary matching into queries 4 and 6 improves results, while multi-tree matching hurts performance noticeably in queries 16, 18 and 20. Looking at the queries in Figure 6, due to the differences in their structural complexity, extracting partial components in queries 4 and 6 produces better similarity than matching partial components in more complex queries (e.g 16 and 18). This makes Queries 4 and 6 benefit from multiple-tree scoring while queries 16 and 18 perform better using a single tree.

Table 1 compares our system with two other state-of-the-art formula search engines. Our model is able to outperform both of them in bpref full relevance and partial relevance. We compare our best runs for $K = 1, 2, 3$ (uni-beta-1, 2-beta-98, 3-beta-90-4) with the *Tangent-S* system[3] and the best performing system at NTCIR-12, *MCAT* [20]. Using only one subexpression match (uni-beta-1), we outperform the other systems in bpref score. Although lower bound Precision@k values are lower for our system and Tangent-S partly due to some relevant hits being unjudged (all MCAT results are judged), we can achieve equal or better

[3] Tangent-S is an improved version of the Tangent system [3] that participated in NTCIR-12.

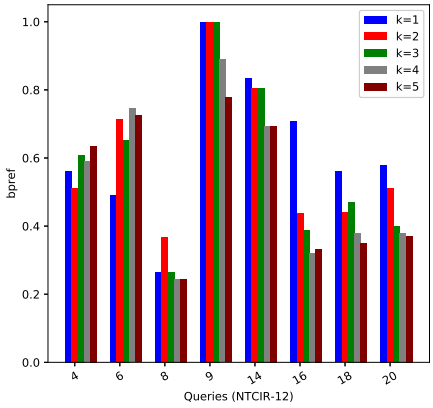


Fig. 5. NTCIR-12 Full relevance scores for matching uniformly-weighted subtrees (1 to 5 trees).

No.	Query Formula
4)	$\nabla \times \mathbf{B} = \mu_0 \mathbf{J} + \underbrace{\mu_0 \epsilon_0 \frac{\partial}{\partial t} \mathbf{E}}_{\text{Maxwell's term}}$
6)	${}^{238}_{92}\mathrm{U} + {}^{64}_{28}\mathrm{Ni} \rightarrow {}^{302}_{120}\mathrm{Ubn}^{*} \rightarrow \ \ldots$
16)	$\tau_{\mathrm{rms}} = \sqrt{\frac{\int_0^\infty (\tau-\overline{\tau})^2 A_c(\tau) d\tau}{\int_0^\infty A_c(\tau) d\tau}}$
18)	$P_i^x = \frac{N!}{n_x!(N-n_x)!} p_x^{n_x} (1-p_x)^{N-n_x}$

Fig. 6. A few example queries in Fig 5.

Table 1. NTCIR-12 Wikpiedia Formula Browsing Task Results (top-1000 hits). $k-\beta$ represents our best run using k matched subtrees in scoring.

Metrics		Fully Relevant					Partially Relevant				
		$1-\beta$	$2-\beta$	$3-\beta$	MCAT	Tangent-S	$1-\beta$	$2-\beta$	$3-\beta$	MCAT	Tangent-S
Bpref		0.6662	0.6696	**0.6726**	0.5678	0.6361	0.5899	**0.5951**	0.5950	0.5698	0.5872
P@5	standard	0.4000	0.4000	0.4000	**0.4800**	**0.4800**	0.5300	0.5300	0.5300	**0.9500**	0.7900
	condensed	0.5400	0.5400	**0.5400**	0.4800	0.5200	0.8900	0.9000	0.9000	**0.9500**	0.9300
	upper bound	**0.8400**	**0.8400**	**0.8400**	0.4800	0.6500	**0.9700**	**0.9700**	**0.9700**	0.9500	0.9600
P@10	standard	0.2850	0.2800	0.2900	**0.3550**	0.3500	0.4600	0.4650	0.4650	**0.8650**	0.7000
	condensed	0.4050	0.4050	**0.4150**	0.3550	**0.4150**	0.8600	0.8650	0.8600	0.8650	**0.9200**
	upper bound	**0.7850**	0.7750	**0.7850**	0.3550	0.5850	**0.9600**	**0.9600**	**0.9600**	0.8650	0.9350
P@15	standard	0.2200	0.2233	0.2233	0.2867	**0.2900**	0.3967	0.4067	0.4100	**0.8333**	0.6433
	condensed	0.3367	0.3433	**0.3467**	0.2867	0.3233	0.8233	0.8333	0.8333	0.8333	**0.8633**
	upper bound	**0.7833**	0.7800	0.7767	0.2867	0.5600	0.9600	**0.9633**	**0.9633**	0.8333	0.9133
P@20	standard	0.1950	0.1950	0.1900	**0.2450**	0.2300	0.3775	0.3850	0.3800	**0.8100**	0.6050
	condensed	0.3125	**0.3175**	**0.3175**	0.2450	0.2825	0.8000	0.7950	0.7925	0.8100	**0.8350**
	upper bound	**0.7800**	**0.7800**	**0.7800**	0.2450	0.5350	0.9625	**0.9700**	**0.9700**	0.8100	0.9100

condensed scores in all the full relevance evaluations, and potentially can have higher precision than the other two systems according to upper bound values.

In terms of efficiency, MCAT reportedly has a median query execution time of 25 seconds, using a server machine and multi-threading [8]. Figure 7 shows query run times for our system and Tangent-S in the same environment using a single thread (Intel Core i5 CPU @ 3.8 GHz each core, DDR4 32GB RAM, 256 GB SSD drive). We compare our most effective run for full-relevance bpref scores (3-beta-90-4), and the most efficient run opd-only which only matches the single largest subtree, counting only leaves for structure scoring. Both of our runs have two versions, one with posting lists read from disk, and another where posting lists are cached in memory. Tangent has two substantial outlier queries

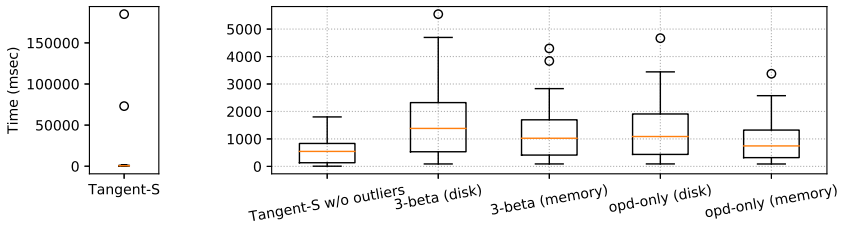


Fig. 7. Query processing times in milliseconds for the 20 NTCIR-12 queries

(due to the non-linear complexity of its structure alignment algorithm), although it is faster in general. However, our execution times are more consistent, with a median time of about 1.5 seconds or less. Our higher typical run time is likely caused by the large number of query "keywords", as the query path set contains all leaf-root paths in all subtrees. Our in-memory posting lists are compressed by Frame-Of-Reference variances [25]. The in-memory version reduces the variance in run times, but the relatively small shift in median times suggests our system is more computation-bound than IO-bound. Our on-disk path index is stored as a naive file-system directory hierarchy where each posting list is a single uncompressed file. The on-disk index takes about 0.8 GB in a reiserFS partition.

5 Conclusion and Future Work

We have introduced a math formula search engine that obtains state-of-the-art results using simple and consistent path-based indexing. Our system uses a novel structural matching scheme that incorporates multiple subtree matches. It achieves better results when considering only visible symbols, and giving greater weight to operands than operators. Our algorithm allows trading-off between effectiveness and efficiency by changing the maximum number of matched subexpressions, or choosing to count matched operators or not. Because the current system examines *all* hits and merges posting lists without any skipping, and our query path set is typically large, there may be great potential in single-process efficiency if we can skip documents and avoid unnecessary computations (e.g. by applying dynamic pruning techniques such as MaxScore [16]). In the future we will extend our retrieval model to support query expansion of math synonyms to improve recall (e.g. expand $1/x$ for x^{-1}), and provide support for wildcard symbols in queries.

References

1. Buckley, C., Voorhees, E.M.: Retrieval evaluation with incomplete information. In: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval. pp. 25–32. ACM (2004)

2. Bunke, H., Shearer, K.: A graph distance metric based on the maximal common subgraph. Pattern Recognition Letters **19**(3-4), 255–259 (1998)
3. Davila, K.: Tangent-3 at the NTCIR-12 MathIR Task. (2016), http://research.nii.ac.jp/ntcir/workshop/OnlineProceedings12/pdf/ntcir/MathIR/06-NTCIR12-MathIR-DavilaK.pdf
4. Davila, K., Zanibbi, R.: Layout and semantics: Combining representations for mathematical formula search. In: Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 1165–1168. ACM (2017)
5. Guidi, F., Sacerdoti Coen, C.: A survey on retrieval of mathematical knowledge. In: Proceedings of the International Conference on Intelligent Computer Mathematics - Volume 9150. pp. 296–315. Springer-Verlag New York, Inc., New York, NY, USA (2015)
6. Hijikata, Y., Hashimoto, H., Nishida, S.: An investigation of index formats for the search of mathml objects. In: 2007 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology. pp. 244–248 (Nov 2007)
7. Kamali, S., Tompa, F.W.: Structural similarity search for mathematics retrieval. In: Carette, J., Aspinall, D., Lange, C., Sojka, P., Windsteiger, W. (eds.) Intelligent Computer Mathematics. pp. 246–262. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
8. Kristianto, G., Topic, G., Aizawa, A.: MCAT math retrieval system for NTCIR-12 MathIR task (06 2016)
9. Lin, X., Gao, L., Hu, X., Tang, Z., Xiao, Y., Liu, X.: A mathematics retrieval system for formulae in layout presentations. In: Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval. SIGIR '14, ACM, New York, NY, USA (2014)
10. Lu, X., Moffat, A., Culpepper, J.S.: The effect of pooling and evaluation depth on ir metrics. Inf. Retr. **19**(4), 416–445 (Aug 2016). https://doi.org/10.1007/s10791-016-9282-6, http://dx.doi.org/10.1007/s10791-016-9282-6
11. Miller, B.R., Youssef, A.: Technical Aspects of the Digital Library of Mathematical Functions. Annals of Mathematics and Artificial Intelligence **38**(1-3), 121–136 (May 2003), https://link.springer.com/article/10.1023/A:1022967814992
12. Misutka, J., Galambos, L.: Extending full text search engine for mathematical content pp. 55–67 (01 2008)
13. Shamir, R., Tsur, D.: Faster subtree isomorphism. Journal of Algorithms **33**(2), 267–280 (Nov 1999)
14. Sojka, P., Líška, M.: Indexing and searching mathematics in digital libraries. In: International Conference on Intelligent Computer Mathematics. pp. 228–243. Springer (2011)
15. Stalnaker, D., Zanibbi, R.: Math expression retrieval using an inverted index over symbol pairs. In: Document recognition and retrieval XXII. vol. 9402, p. 940207. International Society for Optics and Photonics (2015)
16. Turtle, H., Flood, J.: Query evaluation: Strategies and optimizations. Inf. Process. Manage. **31**(6), 831–850 (Nov 1995). https://doi.org/10.1016/0306-4573(95)00020-H, http://dx.doi.org/10.1016/0306-4573(95)00020-H
17. Valiente, G.: An efficient bottom-up distance between trees. In: Proceedings Eighth Symposium on String Processing and Information Retrieval. pp. 212–219 (Nov 2001)
18. Valiente Feruglio, G.A.: Simple and efficient tree comparison (2001)

19. Yokoi, K., Aizawa, A.: An approach to similarity search for mathematical expressions using mathml. Towards a Digital Mathematics Library. Grand Bend, Ontario, Canada, July 8-9th, 2009 pp. 27–35 (2009)
20. Zanibbi, R., Aizawa, A., Kohlhase, M., Ounis, I., Topic, G., Davila, K.: NTCIR-12 MathIR task overview. In: NTCIR (2016)
21. Zanibbi, R., Blostein, D.: Recognition and retrieval of mathematical expressions. Int. J. Doc. Anal. Recognit. **15**(4), 331–357 (Dec 2012)
22. Zanibbi, R., Davila, K., Kane, A., Tompa, F.W.: Multi-stage math formula search: Using appearance-based similarity metrics at scale. In: Proceedings of the 39th International ACM SIGIR Conference on Research & Development in Information Retrieval. SIGIR '16, ACM, NY, USA (2016)
23. Zhong, W., Fang, H.: A novel similarity-search method for mathematical content in LaTeX markup and its implementation. Master's thesis, University of Delaware (2015)
24. Zhong, W., Fang, H.: Opmes: A similarity search engine for mathematical content. In: European Conference on Information Retrieval (ECIR 2016). pp. 849–852. Springer (2016)
25. Zukowski, M., Heman, S., Nes, N., Boncz, P.: Super-scalar ram-cpu cache compression. In: Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on. pp. 59–59. IEEE (2006)