A Novel Similarity-Search method for Mathematical Content in *LaTeX* Markup

ABSTRACT

Mathematical content are widely contained by digital document, search engine should provide us the ability to better search these rich structural content. We propose a similarity-search method to advance information retrieval in LaTeX math content. Our approach uses an intermediate tree representation to capture structural information of mathematical expression, and based on a previous idea, we index math expressions by tree leaf-root paths. A search method to limit search set for possible subexpression isomorphism is provided. We rank search results by intuitive query/document similarity in both structural and symbolic point of view, with consideration of their α-equivalence. For the purpose of evaluation, we also implement a search engine based on our idea.

Categories and Subject Descriptors

H.3 [**Information Search and Retrieval**]: Miscellaneous

General Terms

Algorithms

Keywords

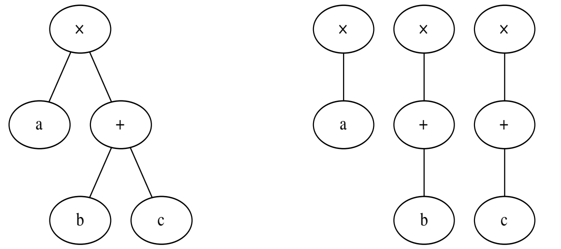
mathematical searching, language processing, search engine

1. INTRODUCTION

With *MathJax* becoming popular, more and more LaTeX markup can be crawled directly from many websites. In order to search those mathematical language in LaTeX markup, a search method that can respect the semantic properties of math expression needs to be developed. Although many researches have been conducted to retrieve information in structured mathematical content (e.g. *MathML*), information retrieval on LaTeX math content is still not well-studied or exhaustively covered by mainstream math IR research.

Unlike general text content, mathematical language, by its nature, has many differences from other textual document, there are a number of new problems in measuring mathematical expression similarity. Among these problems, we know one math expression can be transformed to alternative forms, e.g. $\frac{a+b}{c}$ and $\frac{a}{c} + \frac{b}{c}$ should be considered as semantically the same. To identify those variations requires search engine to apply mathematical transformation rules to a query in order to obtain all forms of relevant expressions. Further, math expressions with the same evaluated value may also be considered relevant, in this case, $1/\sqrt{2}$ is equivalent to $\sin(\pi/4)$. Some computational search engines (e.g. *Symbolab* and *WolframAlpha*) are aware of these problems. But sometimes we need to rely on conventions and context to distinguish expressions such as $f(a+b)$ and $c(a+b)$, because the symbol f in the former expression is likely to represent function instead of a variable, in addition, expression such as g^{-1} can either be reciprocal or an inverse function. Moreover, a higher level of understanding of mathematic knowledge may be required for math-aware search engine to find the results for queries such as "find an article related to the four color theorem" [1].

Yet the problems addressed above are not considered in this paper, instead, we are focusing on the aspects which does not require a "good understanding" of mathematics, we target our research domain to be the following: The first aspect is structural similarity, for example, $ax + (b + c)$ is not equivalent to $(a + b)x + c$ although they have the same set of symbols, this is because their structural difference. However, as the position of operands in math expression can be commutative in some cases, structural similarity is often measured by substructure isomorphism if we use *operation tree* [2] to represent math expressions. The second aspect is symbolic similarity, with concern of α-equivalence. We know that symbols can be used interchangeably in each math formula to express the same meaning, e.g. $a^2 + b^2 = c^2$ and $x^2 + y^2 = z^2$. Nevertheless, we still weight symbolic similarity sometimes, for instance, $E = mc^2$ is considered more meaningful when exact symbols are used rather than just being structurally identical with $y = ax^2$. On the other hand, we should value α-equivalent expressions because changes of symbols in expression preserve more syntactic similarity when changes are made by substitution, e.g. for query $x(1 + x)$, expression $a(1 + a)$ are considered more relevant than $a(1 + b)$. Because the "bond variable" x and a here are at the same positions and both are supposed to represent the same value. All the points addressed here makes transitional IR methods (e.g. bag of words model and tf-idf weighting) deficient to handle mathematical language.



$a(b+c)$ in operation tree   Generated leaf-root paths

Figure 1: Leaf-root path example

2. RELATED WORK

Similarity/boolean search for mathematical content is not a new topic, conference in this topic is getting increasingly research attention and the proposed systems have progressed considerably [3]. DLMF project from NIST [4, 5] and MIaS system [6, 7, 8], notably, use text-based approaches and utilize existing models to deal with math content on top of existing IR tools (such as *Apache Lucene*). They are commonly using augmentation and normalization (by ordering the subexpressions) to enumerate and represent all possible sequences of commutative subexpressions or operands. MWS [9, 10, 11] takes an *automatic theorem proving* approach and uses *term indexing* to minimize the cost of unification algorithm which is able to find if two expressions are equivalent, however, their index relies on RAM memory [11] and needs to include all sub-terms of a formula [9]. A *symbol layout tree* or *presentation tree* [12, 13] is introduced to describe geometric layouts of symbols in a formula. [13] uses two templates to parse LaTeX markup with two typical operator terms: explicit ones ("\frac", "\sqrt", etc.) and implicit ones ("+", "÷", etc.) to form a presentation tree, then extracts original terms and generalized terms from normalized presentation tree, to provide the flexibility of both fuzzy and exact search. [12] uses symbol layout tree as a kind of substitution tree, while [14, 15] have developed a *symbol pairs* idea to capture relative position information between symbols in an expression, this idea enables key-value lookup to speed search. Tree edit distance is adopted by [16] in which they try to overcome the bad time complexity of original algorithm by summarizing and using a compromised edit distance algorithm, then in [17], they improves the speed further by applying with an early termination algorithm along with a distance cache. There are also efforts using image-based approaches [18, 19] and lattice-based approach [20] to measure math formula similarity in a different perspective.

3. METHODOLOGY

Our method can be seen as an approach built upon the idea of leaf-root path or sub-path [21, 22, 23, 24] from an operation tree, to capture structure and semantic information of math expression. Figure 1 is an example of generated leaf-root paths for math expression $a(b+c)$. The intuition behind this idea is that an operation tree, no matter how operands are ordered, uniquely determines the leaf-node paths decomposed from the tree. This makes leaf-root path a good fit for representing mathematical expression because commutative operands are exhaustively used in mathematical language. Besides, by going bottom-up from leaves of an operation tree, we are essentially traversing to an expression from its subexpression for every level. So we can index the leaf-root paths and search an expression by going through and beyond the leaf-root paths from its subexpression.

We develop these ideas to simultaneously search along the way of all leaf-root paths from a given query operation tree, so that we are essentially pruning indexes which does not share the common postfixes beyond the root of the query tree. To better describe this idea and further ideas based upon this, we will put it in a formal way.

3.1 Formal Definition

Here we clarify some notations used throughout this paper, a path p is a sequence of numbers given by $p = p_0p_1 \dots p_n$ where $n \geq 0$, $p_i \in \mathbf{R}$ and $\mathbf{P}$ is the set of all paths. A path with only one element can be explicitly wrapped by a bracket, e.g. $p = (p_0)$, to avoid confusion. Any function $f(\cdot) = y \in \mathbf{R}$ applied on path p is mapped to a path too: $f(p) = f(p_0)f(p_1)\dots f(p_n)$. And we name a *concatenation* of two paths ${}^1p = p_0p_1 \dots p_n$ and ${}^2p = p_np_{n+1}\dots p_m$ where $m \geq n$, to be a new path denoted as ${}^1p \cdot {}^2p = p_0p_1 \dots p_np_{n+1}\dots p_m$, and the concatenation of a path p on a set $S = \{s_1, s_2 \dots s_n\}$ is defined as $S \cdot p = \{s_1 \cdot p,\ s_2 \cdot p\ \dots\ s_n \cdot p\}$. Furthermore, *the longest common postfix* path p^* between two path p_1 and p_2 is mapped by the function named lcp, which is defined by $p^* = \mathrm{lcp}(p_1, p_2) = \mathrm{lcp}(p_2, p_1)$.

We introduce a *formula tree* to represent a mathematical expression, in which each node is associated with a label to represent the unified token (e.g. same value for token $+$, $\oplus$ and $\pm$) and each leaf node is associated with a symbol to identify math expression operand. Besides, a *formula subtree* relation is also defined to address the sub-structure relation between two mathematical expressions.

3.1.1 Formula tree

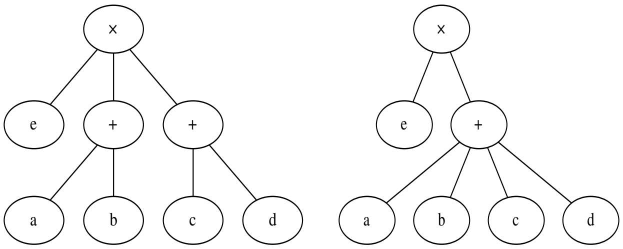
A *formula tree* is a labeled rooted tree $T = T(V, E, r)$ with root r, where each vertices $v \in V(T)$ is associated with a label (not necessarily unique in the same tree) $\ell_T(v) \in \mathbf{R}$ mapped by label function ℓ_T, and each leaf $l \in V(T)$ is also associated with a symbol $\mathcal{S}_T(v) \in \mathbf{R}$ mapped by symbol function $\mathcal{S}_T$. For convenience, we will write ℓ and $\mathcal{S}$ as short names which refer to the tree implied by the context, also we use function $\mathcal{S}(p)$ to indicate the symbol of the leaf in a leaf-root path p.

3.1.2 Formula subtree

Given formula tree S and T, we say S is a *formula subtree* of T if there exists an injective mapping $\phi : V(S) \to V(T)$ satisfying:

1. $\forall\ (v_1, v_2) \in E(S)$, we have $(\phi(v_1), \phi(v_2)) \in E(T)$;
2. $\forall\ v \in V(S)$, we have $\ell(v) = \ell(\phi(v))$;
3. If $v \in V(S)$ is a leaf vertices in S, then $\phi(v)$ is also a leaf in T.

Such a mapping ϕ is called a formula subtree isomorphic embedding (or formula embedding) for $S \to T$. If satisfied, we denote $S \preceq_l T$ on Φ, where Φ ($\Phi \neq \emptyset$) is the set of all the possible formula embeddings for $S \to T$.



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

Figure 2: Leaf-root paths with different structure

3.1.3 Leaf-root path set

A *leaf-root path set* generated by tree T is a set of all the leaf-root paths from tree T, mapped by a function $g(T)$.

3.1.4 Index

An *index* Π is a set of trees such that $\forall\ T \in \Pi$, we have $T \in \mathcal{I}_\Pi(a)$ for any $a \in \ell(g(T))$, we say T is indexed in Π and $\mathcal{I}_\Pi$ is called an index look-up function for index Π.

3.2 Search Method

For a collection of document expressions, we will index them by merging all the leaf-root paths from each document formula tree into a large "inverted" index tree, in which each node at path a stores the information of all the indexed formula trees in $\mathcal{I}_\Pi(a)$.

Through searching all sub-paths at the same time, we are able to limit the set of possible formula trees being structurally matching (in formula subtree relation) with a query formula tree, to only a subset of our index. This is illustrated as follows.

Given an index Π and a formula tree T_q, $\forall\ T_d \in \Pi$: If $T_q \preceq_l T_d$ on Φ, then $\exists\ \hat{a} \in \mathbf{P},\ s.t.$

$$T_d \in \bigcap_{a\in L} \mathcal{I}_\Pi(a)$$

where $L = \ell(g(T_q)) \cdot \hat{a}$.

Justification. Denote the root of T_q and T_d as r and s respectively. Let $\hat{p}$ be the path determined by vertices from $t = \phi(r)$ to s in T_d, and ${}^1p,\ {}^2p \ldots {}^np,\ n \geq 1$ be all the leaf-node paths in T_q. Then $\hat{a} = \ell(\hat{p})$, this is because: $L = \ell(\{{}^1p,\ {}^2p \ldots {}^np\}) \cdot \hat{a} = \ell(\{\phi({}^1p), \phi({}^2p) \ldots \phi({}^np)\}) \cdot \ell(\hat{p}) = \{\ell(\phi({}^1p)\cdot\hat{p}), \ell(\phi({}^2p)\cdot\hat{p}) \ldots \ell(\phi({}^np)\cdot\hat{p})\}$. According to definition 3.1.2 and $t = \phi(r)$, we have $\phi({}^ip)\cdot\hat{p} \in g(T_d),\ 1 \leq i \leq n$. Since $T_d \in \Pi$, T_d is indexed in Π with respect to each of the elements in L, that is to say $\forall\ a \in L,\ T_d \in \mathcal{I}_\Pi(a)$.

In a nutshell, we search the index by intersecting the indexed formula trees from all the generated leaf-node paths at the same time, then further possible search path $\hat{a}$ is only possible when paths along the generated leaf-node paths in the index have a common postfix. Therefore we can "merge" the paths ahead and prune those paths not in common. Level by level, we are always able to find the structurally matched formula tree as long as it is indexed in Π.

```
procedure removeCandidate(d, Q, C)
    for a ∈ Q do
        if C_a = ∅ then
            return ∅
        else
            C_a := C_a − {d}
    return C

procedure match(a, a′, Q, C)
    for b ∈ Q do
        t := lcp(a, b)
        Q_t := Q_t ∪ {b}
        P := P ∪ {t}
    for t ∈ P do
        for b ∈ Q_t do
            for b′ ∈ C_b do
                if t ≠ lcp(a′, b′) then
                    C := removeCandidate(b′, Q_t, C)
                    if |C| = 0 then
                        return FAIL
        if decomposeAndMatch(Q_t, C) = FAIL then
            return FAIL
    return SUCC

procedure decomposeAndMatch(Q, C)
    if Q = ∅ then return SUCC
    a := OnePathIn(Q)    ▷ Choose a reference path in Q
    Q_new := Q − {a}
    for a′ ∈ C_a do
        C_new := removeCandidate(a′, Q_new, C)
        if C_new = ∅ then return FAIL
        if match(a, a′, Q_new, C_new) then return SUCC
    return FAIL
```

Figure 3: The decompose-and-match algorithm

3.3 Substructure Matching

However, query formula tree will not necessarily be formula subtree of all the document (indexed) formula trees in our search set $\bigcap_{a\in L} \mathcal{I}_\Pi(a)$, even if their generated leaf-root paths are identical. One supporting example for this point is shown in figure 2. To address this problem, we propose an algorithm described in in figure 3, to test the document formula trees in our search set to see if they are in formula subtree relation with query formula tree. This algorithm is inspired from the following observations.

3.3.1 Observation 1

For two formula trees which satisfy $T_q \preceq_l T_d$ on Φ, then $\forall\ \phi \in \Phi$, $p \in g(T_q)$, also any vertices v along path p, the following properties are obtained:

$$\deg(v) \leq \deg(\phi(v)) \tag{1}$$

$$\ell(p) = \ell(\phi(p)) \tag{2}$$

$$|g(T_q)| \leq |g(T_d)| \tag{3}$$

Justification. Because $\forall\ w \in V(T_q)\ s.t.\ (v,w) \in E(T_q)$, there exists $(\phi(v), \phi(w)) \in E(T_d)$. And for any (if exists) two different edges $(v,w_1),(v,w_2) \in E(T_q)$, $w_1 \neq w_2 \in V(T_q)$, we know $(\phi(v),\phi(w_1)) \neq (\phi(v),\phi(w_2))$ by definition 3.1.2. Therefore any different edge from v is associated with a distinct edge from $\phi(v)$, thus we can get (1). Given the fact that every non-empty path p can be decomposed into a series of edges $(p_0, p_1), (p_1, p_2) \dots (p_{n-1}, p_n)$, $n > 0$, property (2) is trivial. Because there is exact one path between every two nodes in a tree, the leaf-root path is uniquely determined by a leaf node in a tree. Hence the rationale of (3) can be obtained in a similar manner with that of (1), expect neighbor edges are replaced by leaf-node paths.

3.3.2 Observation 2

Given two formula trees T_q and T_d, if $|g(T_q)| = 1$ and $\ell(g(T_q)) \subseteq \ell(g(T_d))$, then $T_q \preceq_l T_d$.

Justification. Obviously there is only single one leaf-root path in T_q because $|g(T_q)| = 1$. Denote the path as $p = p_0 \dots p_n$, $n \geq 0$ where p_n is the leaf, and let $a = \ell(p)$. Since $a \subseteq \ell(g(T_d))$, we know that there must exist a path $p' = p'_0 \dots p'_n \in g(T_d)$ such that $a = \ell(p')$. Without loss of generality, suppose p'_n is the leaf of T_d. Now the injective function $\phi : p_i \to p'_i$, $0 \leq i \leq n$ satisfies all the requirements for T_q as a formula subtree of T_d.

3.3.3 Observation 3

For two formula trees T_q and T_d, if $T_q = T(V, E, r) \preceq_l T_d$ on Φ, $\forall a, b \in g(T_q)$ and a mapping $\phi \in \Phi$. Let $T'_d = {}^tT_d$ where $t = \phi(r)$ and $a' = \phi(a)$, $\forall\, b' \in g(T'_d)$, it follows that:

$$b' = \phi(b) \quad \Rightarrow \quad |\text{lcp}(a, b)| = |\text{lcp}(a', b')|$$

Furthermore, $\forall\ c \in g(T_q)$ *s.t.* $|\text{lcp}(a, b)| \neq |\text{lcp}(a, c)|$, we have

$$|\text{lcp}(a, b)| = |\text{lcp}(a', b')| \quad \Rightarrow \quad b' \neq \phi(c)$$

Justification. Because $a, b \in g(T_q)$, thus $a_0 = b_0 = r$, and we can also make sure $\text{lcp}(a, b) \geq 1$. Denote the path of $a = a_0 \dots a_n a_{n+1} \dots a_{l-1}$, similarly denote the path of b as $b = b_0 \dots b_n b_{n+1} \dots b_{m-1}$, where the length of each $l, m \geq 1$ and $a_i = b_i$, $0 \leq i \leq n \leq \min(l-1, m-1)$ while $a_{n+1} \neq b_{n+1}$ if $l, m > 1$. On the other hand $a' = \phi(a)$ and $b' \in g({}^tT_d)$, therefore $a'_0 = \phi(a_0) = \phi(r) = t = b'_0$. For the first conclusion, if $b' = \phi(b)$, there are two cases. If any of $|a|$ or $|b|$ is equal to one then $|\text{lcp}(a, b)| = |(r)| = |(t)| = |\text{lcp}(a', b')| = 1$; Otherwise if $l, m > 1$, path $a_0 \dots a_n = b_0 \dots b_n$ and $a_{n+1} \neq b_{n+1}$ follow that $\phi(a_0 \dots a_n) = \phi(b_0 \dots b_n)$ and $\phi(a_{n+1}) \neq \phi(b_{n+1})$ by definition. Because edge $(\phi(a_n), \phi(a_{n+1}))$ and $(\phi(b_n), \phi(b_{n+1}))$ are also in $E(T'_d)$, we have $|\text{lcp}(a, b)| = |\text{lcp}(a', b')| = n$. For the second conclusion, we prove by contradiction. Assume $b' = \phi(c)$, by the first conclusion we know $|\text{lcp}(a, c)| = |\text{lcp}(a', b')|$. On the other hand, because $|\text{lcp}(a, c)| \neq |\text{lcp}(a, b)| = |\text{lcp}(a', b')|$, thus $|\text{lcp}(a, c)| \neq |\text{lcp}(a', b')|$ which is impossible.

For a query formula tree T_q and a document formula tree T_d, observation 1 offers us some constrains for finding initial possible isomorphic paths in T_d (what we call *candidates*) for a given query path in T_q. And observation 2 is a sufficient condition to test substructure relation, but the query tree has to consist only one leaf-root path to be tested. While observation 3 states two necessary conditions for one formula tree to be a formula subtree of another and implies that a group of query leaf-root paths can only be isomorphic to someones in another group of leaf-root paths in document formula tree. This leads to the idea to decompose the formula tree and divide the problem into subproblems by ruling out impossible candidates using observation 3, until at some point we can apply observation 2 or observe trivial cases to solve all the sub-matching problems.

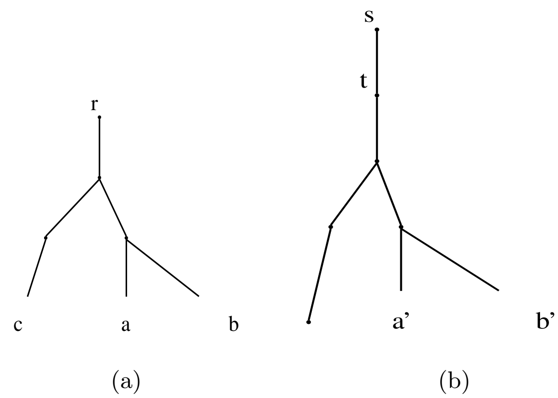


Figure 4: Formula subtree matching

To illustrate our DECOMPOSEANDMATCH algorithm, figure 4 gives a general case in which query tree in (a) is trying to match a document tree in (b). Initially every leaf-root path in (a) should be associated with a set of candidate paths in (b) which satisfy the constrains in 3.3.1. Then we arbitrarily choose a path in (a) as a *reference path* (heuristically a *heavy path* [25]), for each of the paths in its candidate set, we choose it as reference path in (b), and suppose we choose a' here. At this time we can apply the two constrains from 3.3.3 and ruling out some impossible isomorphic paths in candidate set of each path in (a) and divide the problems further. For example, because $|\text{lcp}(a, b)| = |\text{lcp}(a', b')|$, we know b' is still in candidate set of b; while b' is not in candidate of c because $|\text{lcp}(a, b)| \neq |\text{lcp}(a, c)|$. After going through these eliminations for each leaf-node path (except the reference path a) in (a), we now have two similar subproblems: c as a subtree along with its candidate set, and b as a subtree along with its candidate set. We can apply this algorithm recursively until a trivial subproblem is reached (e.g. the case in 3.3.2). During this process, if we find any candidate set to be empty, we stop the subproblem process and change to another reference path or stop the algorithm completely if every possible reference path is tried. The argument Q and C is the set of leaf-root paths in query tree and the candidate sets associated with all query tree leaf-root paths respectively. The procedure returns SUCC if a matching possibility is found, otherwise FAIL is returned indicating the formula tree in (a) cannot not be a formula subtree of that in (b).

DECOMPOSEANDMATCH algorithm offers a way to "double-check" structure isomorphism, because for any two formula tree T_q and T_d and $\forall\ \hat{a} \in \mathbf{P}$, if $\ell(g(T_q)) \cdot \hat{a} \subseteq \ell(g(T_d))$, it is not sufficient to imply $T_q \preceq_l T_d$. Nevertheless, we think the cases which makes the above statement insufficient are fairly rare in common mathematical content, and the complexity introduced from this algorithm will offset the benefit to identify the structure isomorphism. Thus a compromised search, for efficiency reason, would assume all the document formula trees in search set $\bigcap_{a \in L} \mathcal{I}_{\Pi}(a)$ is structurally matching the given query formula tree T_q.

3.4 Structural Similarity

After searching structurally matched document expressions in a boolean manner, we use two factors to measure their structural similarity degree.

The first factor is *matching depth*. As it is addressed in [6], the deeper sub-formulae in in mathematical expression will make it less important to the overall formula. For example, given query formula $\sqrt{a}$, expression $\sqrt{x}$ would score higher than $\sqrt{\sqrt{x}}$ does. To reflect the depth where two expressions match, we define *matching depth factor* $f(d)$ to be a function value in negative correlation with matching depth $d = |\hat{a}|$, e.g. $f(d) = 1/(1+d)$ in our method. The second factor is *matching ratio*. According to the property (2) in 3.3.1, we have $|g(T_q)|/|g(T_d)| \leq 1$, and the ratio r on left-hand is defined as *matching-ratio*, which characterises the structural coverage for the matching part in an expression. Intuition behind this is, for example, given query $\alpha y + \beta$, document expression $ax + b$ should precede $x^2 + ax + b$ simply because the query matches more "area" of the former expression than that of the latter.

3.5 Symbolic Similarity

As we have discussed in section 1, besides structural similarity, symbolic similarity is also essential to be considered. Here our scoring goal for symbolic similarity can be summarized in two points:

- Given two formula trees $T_q \preceq_l T_d$ on Φ, if path $a \in g(T_q)$ is isomorphic to path $a' \in g(T_d)$, that is to say, $\phi(a) = a'$, where $\phi \in \Phi$, then if their symbol matches, i.e. $\mathcal{S}(a) = \mathcal{S}(a')$, we score them higher than those do not match symbolically. And the more symbolic matches there are, the higher symbolic relevance degree two expressions have.
- α-equivalent expressions have more symbolic relevance degree than those are not, and the more bond variables (at the structurally matching positions) two expressions match, the more symbolically relevant they are considered to be.

The two are illustrated as follows. Let the rank of a structural relevant document expression d be $r(d)$, and given query $\sqrt{a}(a-b)$ for instance. Then the first goal is essentially saying

$$r\big(\sqrt{a}(a-b)\big) > r\big(\sqrt{a}(a-x)\big) > r\big(\sqrt{x}(x-y)\big)$$

here the second one has two symbols matching while the third one has no symbolic match at all.

By the second goal, we know

$$r\big(\sqrt{x}(x-b)\big) > r\big(\sqrt{x}(y-b)\big)$$

because here the second one does not have the same number of bond-variable matches as the first one does.

However, sometimes the two goals can be conflicting. Given document expression $\sqrt{a}(x-b)$ and $\sqrt{x}(x-b)$ for instance, the former has two symbolic matches (i.e. "a" and "b") while it does not have bond-variable match. On the other hand, the latter has bond-variable match while it only has one

```
procedure markAndCross(D, Q, C)
    score := 0
    if D = ∅ then
        return 0
    for a′ ∈ D do
        T_a′ := unmark
    for v ∈ 𝒱(D) do
        B_v := 0
    QList := sortBySymbolAndOccur(Q)
    for a in QList do
        for v ∈ 𝒱(D) do
            m := −∞
            m_p := ∅
            for a′ ∈ C_a ∩ {y | y = v, y ∈ 𝒱(D)} do
                if T_a′ = unmark and sim(a, a′) > m then
                    m := sim(a, a′)
                    m_p := a′
            if m_p ≠ ∅ then
                T_m_p := mark
                B_v := B_v + m
            else                              ▷ Exhausted all candidates
                return 0
        if 𝒮(a) changed or  last iteration of a then
            m := −∞
            m_v := ∅
            for v ∈ 𝒱(D) do
                if B_v > m then
                    m := B_v
                    m_v := v
                B_v := 0
            score := score + m
            for v ∈ 𝒱(D) do
                if v = m_v then
                    nextState := unmark
                else
                    nextState := cross
                for a′ ∈ C_a ∩ {y | y = v, y ∈ 𝒱(D)} do
                    if T_a′ = mark then
                        T_a′ := nextState
    return score
```

Figure 5: The mark-and-cross algorithm

symbolic match (i.e. "b"). We nevertheless score the latter higher because it does not lose any mathematic semantics.

To meet the goals above, intuitively, we first take the bond variable with greatest number of occurrence in its expression in the query, try to match as many symbols as possible with bond variables from document expression. The bond variable with most matches would be chosen (named *best-matching*) to contribute to final symbolic relevance score (proportionally to the number of matches in that bond variable), and we exclude its paths from matching query paths in future iterations. In the next iteration, we choose the bond variable with the second number of occurrence and repeat this process until all the query bond variable are iterated.

We describe our algorithm in figure 5 which measures symbolic similarity given two expressions. The markAndCross algorithm takes three arguments, the set of leaf-root paths D and Q in document expression and query expression respectively, and the candidate sets C associated with all leaf-root paths in query. The bond variables in D is addressed by the set $\mathcal{V}(D) = \{x \mid \mathcal{S}(x),\ x \in D\}$, which contains all the leaf node symbols from document expression. Function sortBySymbolAndOccur takes the elements from a set of leaf-root paths and returns a list containing all the paths. The list is sorted by the tuple $(\mathcal{S}(p), O_p)$, where O_p is the number of occurrence of path p in the list. Each document path a' is associated with a tag $T_{a'}$ which has three possible states: marked, unmarked and crossed. And bond variable $v \in \mathcal{V}(D)$ can be given a score B_v which represents the similarity degree between current evaluating query/document bond variables. The function $\text{sim}(a, a')$ measures the symbolic similarity degree between two leaf-root paths a and a'. Intuitively, we set the similarity function

$$\text{sim}(a, a') = \begin{cases} 1 & \text{if } \mathcal{S}(a) = \mathcal{S}(a') \\ \alpha < 1 & \text{otherwise} \end{cases}$$

to give more weights to leaf-root paths with exact symbol match.

Let us determine the proper value for α. Consider the conflicting cases stated in this section by using another example here, given query expression $a + \frac{1}{a} + \sqrt{a}$ and document expression $a + \frac{1}{a} + b + \frac{1}{b} + \sqrt{b}$, we consider bound variable matching

$$\boxed{\text{a}} + \frac{1}{\boxed{\text{a}}} + \sqrt{\boxed{\text{a}}}$$

with

$$a + \frac{1}{a} + \boxed{\text{b}} + \frac{1}{\boxed{\text{b}}} + \sqrt{\boxed{\text{b}}}$$

weighted more than exact symbol matching

$$\boxed{\text{a}} + \frac{1}{\boxed{\text{a}}} + \sqrt{a}$$

with

$$\boxed{\text{a}} + \frac{1}{\boxed{\text{a}}} + b + \frac{1}{b} + \sqrt{b}$$

(expressions surrounded by a box here indicates the matching part)

Because the former matching has more variables involved even if they are not identical symbolic matches compared with its counterpart of the latter. That is to say, given a document bound variable matching k variables with that in query, we need α to satisfy $k\alpha > (k-1) \times 1 = k - 1$ and $\alpha < 1$. Therefore, we set α to a value close to 1 (e.g. 0.9 in our practice).

By sorting the query paths in Q, the algorithm is able to take out paths from same bound variable in maximum-occurrence-first order from QList. Each query path a tries to match a path a' in each document bond variable v by selecting the unmarked path m_p with maximum $\text{sim}(a, a')$ value, and accumulate the value on B_v indicating the similarity between currently evaluating query bond variable and the bond variable v. In addition, mark the tag T_{m_p} associated with the document path m_p. Once a query bond variable has been iterated completely (line 23), find the d cument variable m_v with greatest B_v value m, and regard it as the best-matching bond variable in $\mathcal{V}(D)$ with the query bond variable just iterated, then add m to the result score. Before iterating a new query bond variable, we will cross all the document paths of variable m_v to indicate they are confirmed been matched, and unmark the tags of those marked paths that are not variable m_v. We continue doing so until all the query path is iterated, and finally return the result score indicating the symbolic similarity between the two expressions.

4. IMPLEMENTATION

In order to evaluate the performance of our method, we have implemented a proof-of-concept search engine [1] as well as a parser for parsing LaTeX markup content directly into operation trees.

4.1 Parser

We are tokenizing math content using lexer generator *flex* and have implemented a LALR parser generated from a set of *GNU bison* grammar rules we defined specifically for MathJax content in a subset of LaTeX (those related to mathematics). Our parser transforms a math formula into an in-memory operation tree (representing formula tree), as an intermediate step to extract the path labels, path ID, and degree numbers associated with the path for every leaf-root path from the tree. Our lexer omits all LaTeX control sequences not matching any pattern of our defined tokens, most of them are considered unrelated to math formula semantics (environment statement, color, mbox etc.).

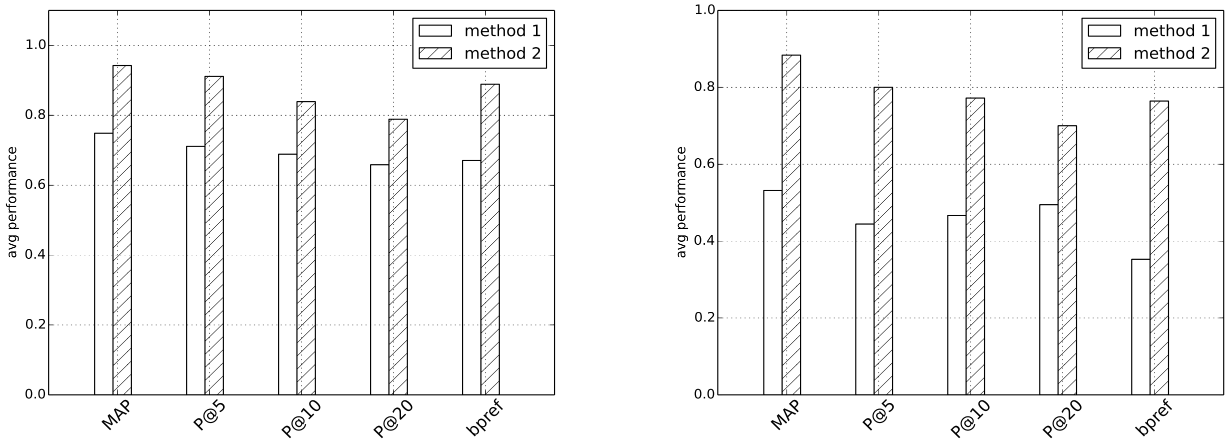
4.2 Index

The indexer writes the information extracted from parser into disk. There are two parts in our index, the first part uses native file system (for the sake of implementation simplicity) to store leaf-root path labels in directories from which our search engine can go level by level. Path ID, degree numbers, and also the formula ID generating that leaf-root path (we refer to these three as *branch word*) are stored in a "posting" file at the directory corresponding to that branch word labels, e.g. the example in figure 1 will index into two directories: ./VAR/TIMES and ./VAR/ADD/TIMES. All the indexed branch word with path labels corresponding to directory ./VAR/TIMES/ are stored in the posting file of that directory, locating at: ./VAR/TIMES/posting.bin. Branch words in a posting file are ordered by their formula IDs to speed merge search. The second part of our index is a key-value database (using *Kyoto Cabinet*) to map a formula ID to additional information for that formula (e.g. original markup, number of leaf-root paths $|T_d|$ and the URL on which the formula is crawled).

4.3 Searching and Ranking

The compromised substructure searching in indexed expressions (described in section 3.2) is used to filter out likely isomorphic expressions in our index. Searching is performed by simultaneously going from all the directories corresponding to the generated leaf-root paths from query formula tree, to all their merged subdirectories. We go on traversing and intersect the branch words in the posting files of each searching directory by their formula IDs. All the intersected formula IDs actually represent the indexed formula trees in

[1] demo page: http://infolab.ece.udel.edu:8912/cowpie/



Partial relevance (relevance score ≥ 1)   Relevance (relevance score ≥ 3)

Figure 6: Effectiveness Comparison

our search set $\bigcap_{a \in L} \mathcal{I}_{\Pi}(a)$. Every document formula identified in the search set is considered as a hit, we then apply MARKANDCROSS algorithm to get its symbolic similarity score with query formula. Denote this score to be s, and $f(d)$ being the matching depth factor, r being the matching ratio for one pair of query/document. We will use them together to indicate the overall similarity and rank items in search results, measured by tuple $(s, f(d), r)$. To know if one should ranked higher than another, first compare s, if equal, compare $f(d)$ and then r. In addition, we use a min-heap to keep the top-k scored items in our search results (by replacing the lowest scored items if we find a newer hit with higher score), where k is the maximum number of items a search results may contain. We also place a valve on the number of branch words can be searched for one query at a time, exceeding this limit will result in a stop of searching and immediately returning the search results we have so far. Because some query can potentially have a very long posting list, doing so would make our search response no more than a certain time.

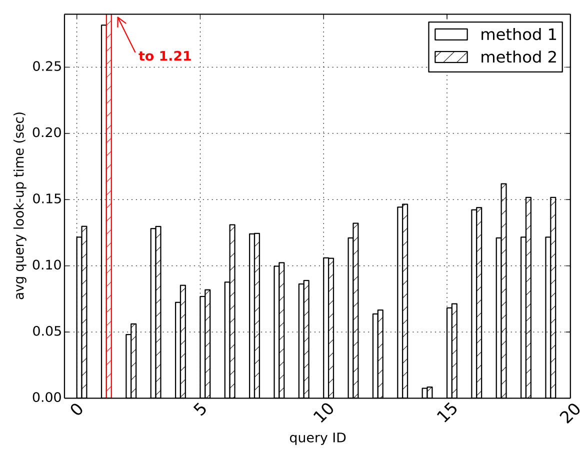


Figure 7: Efficiency Performance

5. EVALUATION

The popular evaluation dataset in this research domain, the NTCIR Math Task dataset, is in MathML/XML format, and original LaTeX information is not always preserved in their dataset. Because we are parsing LaTeX directly, although converting MathML/XML formula back into LaTeX is possible, we fail to convert all the document correctly (using *pandoc*). Furthermore, supporting wildcard query is a default requirement in NTCIR Math Task, while our approach does not find a way to enable wildcard feature so far. Nevertheless, we choose to create our own dataset to evaluate our method. We have crawled pure LaTeX content from the posts of nearly entire (27180 pages of questions) Math Stack Exchange website before March 2015 (roughly 60MB *bzip2* compressed raw data [2] in plain text format with one LaTeX math mode content per line, and one file for each post). The resulting index has over 8 million formula IDs, with 892 MB key-value database file and roughly 9.4GB directories and posting files. Our test query set [3] consists queries mostly from [1] and [14], some of them are excluded here because we are not able to find similar formula in our own dataset. Table 1 shows our complete test queries used in our evaluation.

There are four levels of relevance scored from 0 to 4 in our evaluation. The criteria considers both structural similarity and symbolic similarity. Structural similarity is measured by 0, 1 (mostly similar) and 2 (complete matching), and symbolic similarity is measured by 0, 1 (mostly identical symbols for the matching parts) and 2 (identical symbols for the matching parts). The level of relevance is simply the sum of the two scores. We have evaluated our method (method 2) and a baseline method (method 1) without considering symbolic similarity. Table 2 shows the distribution

[2]http://infolab.ece.udel.edu:8912/cowpie/math-stack-exchange-march-2015.tar.bz2

[3]http://infolab.ece.udel.edu:8912/cowpie/test-queries.tex

of hits and relevance level for each run. The performance and comparison of these two method are shown in figure 7 and figure 6. Query look-up time is one part of total response time, it is the time consumed to search directories and posting files. The evaluation results show that our symbolic similarity, if used, can boost search effectiveness in all the five metres evaluated, and consumes a reasonable of time on top of the baseline method.

6. CONCLUSION AND FUTURE WORK

Our method tries to measure mathematical expressions similarity through their structures and operand symbols. We search relevant expressions only in a subset of index and have achieved rather satisfactory effectiveness, meanwhile, our index does not need augmentation thus has less redundancy. In the next stage, it is desired to integrate text search ability into our math-only search method. Additionally, the manner to break math formula into branch words and index them through posting list makes it easy to parallelize and distribute the searching process, there is a large potential for future efforts to improve efficiency.

7. REFERENCES

[1] Topics for the ntcir-10 math task full-text search queries. http://ntcir-math.nii.ac.jp/wp-content/blogs.dir/13/files/2014/02/NTCIR10-math-topics.pdf.

[2] Richard Zanibbi and Dorothea Blostein. Recognition and retrieval of mathematical expressions. *International Journal on Document Analysis and Recognition (IJDAR)*, 15(4):331–357, 2012.

[3] Akiko Aizawa, Michael Kohlhase, and Iadh Ounis. Ntcir-11 math-2 task overview. *The 11th NTCIR Conference*, 2014.

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

[5] Youssef A. Information search and retrieval of mathematical contents: Issues and methods. *The ISCA 14th IntâĂŹl Conf. on Intelligent and Adaptive Systems and Software Engineering (IASSE 2005)*, 2005.

[6] Petr Sojka and Martin L. Indexing and searching mathematics in digital libraries. *Intelligent Computer Mathematics*, 6824:228–243, 2011.

[7] Petr Sojka and Martin L. The art of mathematics retrieval. *ACM Conference on Document Engineering, DocEng 2011*, 2011.

[8] Martin L. Evaluation of mathematics retrieval. Master's thesis, Masarykova University, 2013.

[9] Michael Kohlhase and Ioan A. SÂÿucan. A search engine for mathematical formulae. In *Proc. of Artificial Intelligence and Symbolic Computation, number 4120 in LNAI*, pages 241–253. Springer, 2006.

[10] Michael Kohlhase. Mathwebsearch 0.4 a semantic search engine for mathematics.

[11] Michael Kohlhase. Mathwebsearch 0.5: Scaling an open formula search engine.

[12] Thomas Schellenberg, Bo Yuan, and Richard Zanibbi. Layout-based substitution tree indexing and retrieval for mathematical expressions. *Proc. SPIE 8297, Document Recognition and Retrieval XIX, 82970I*, 2012.

[13] Xuan Hu, Liangcai Gao, Xiaoyan Lin, Zhi Tang, Xiaofan Lin, and Josef B. Baker. Wikimirs: A mathematical information retrieval system for wikipedia. *Proceedings of the 13th ACM/IEEE-CS joint conference on Digital libraries. Pages 11-20*, 2013.

[14] David Stalnaker and Richard Zanibbi. Math expression retrieval using an inverted index over symbol pairs in math expressions: The tangent math search engine at ntcir 2014. *Proc. SPIE 9402, Document Recognition and Retrieval XXII, 940207*, 2015.

[15] David Stalnaker and Richard Zanibbi. Math expression retrieval using an inverted index over symbol pairs. *Proc. SPIE 9402, Document Recognition and Retrieval XXII, 940207*, 2015.

[16] Cyril Laitang, Mohand Boughanem, and Karen Pinel-Sauvagnat. Xml information retrieval through tree edit distance and structural summaries. In *Information Retrieval Technology*, volume 7097 of *Lecture Notes in Computer Science*, pages 73–83. Springer Berlin Heidelberg, 2011.

[17] Shahab Kamali and FrankWm. Tompa. Structural similarity search for mathematics retrieval. In *Intelligent Computer Mathematics*, volume 7961 of *Lecture Notes in Computer Science*, pages 246–262. Springer Berlin Heidelberg, 2013.

[18] Richard Zanibbi and Bo Yuan. Keyword and image-based retrieval for mathematical expressions. *Multi-disciplinary Trends in Artificial Intelligence. 6th International Workshop*, pages 23–34, 2011.

[19] Li Yu and Richard Zanibbi. Math spotting: Retrieving math in technical documents using handwritten query images. *Document Analysis and Recognition (ICDAR)*, pages 446 – 451, 2009.

[20] T. Nguyen, S. Hui, and K. Chang. A lattice-based approach for mathematical search using formal concept analysis. *Expert Systems with Applications*, 2012.

[21] Hiroshi Ichikawa, Taiichi Hashimoto, Takenobu Tokunaga, and Hozumi Tanaka. New methods of retrieve sentences based on syntactic similarity. *IPSJ SIG Technical Reports*, pages 39–46, 2005.

[22] Yoshinori Hijikata, Hideki Hashimoto, and Shogo Nishida. An investigation of index formats for the search of mathml objects. In *Web Intelligence/IAT Workshops*, pages 244–248. IEEE, 2007.

[23] Yokoi Keisuke and Aizawa Akiko. An approach to similarity search for mathematical expressions using mathml. *Towards a Digital Mathematics Library. Grand Bend, Ontario, Canada*, pages 27–35, 2009.

[24] Yoshinori Hijikata, Hideki Hashimoto, and Shogo Nishida. Search mathematical formulas by mathematical formulas. *Human Interface and the Management of Information. Designing Information, Symposium on Human Interface*, pages 404–411, 2009.

[25] PhilipN. Klein. Computing the edit-distance between unrooted ordered trees. volume 1461 of *Lecture Notes in Computer Science*, pages 91–102. Springer Berlin Heidelberg, 1998.

ID	formula	ID	formula
1	$\int_0^\infty dx \int_x^\infty F(x,y)dy = \int_0^\infty dy \int_0^y F(x,y)dx$	2	$X(i\omega)$
3	$x^n + y^n = z^n$	4	$\int_{-\infty}^{\infty} e^{-x^2} dx$
5	$\frac{f(x+h)-f(x)}{h}$	6	$\frac{\sin x}{x}$
7	$ax^2 + bx + c$	8	$\frac{e^x+y}{z}$
9	$O(n \log n)$	10	$H^n(X) = Z^n(X)/B^n(X)$
11	$A_n = \frac{1}{\pi} \int_{-\pi}^{\pi} F(x) \cos(nx) dx$	12	$\lim_{x\to\infty}(1+\dfrac{1}{x})^x$
13	$f(x) = f(0) + f'(0)x + \frac{f''(0)}{2!}x^2 + \dots$	14	$f(a) = \frac{1}{2\pi i} \oint_r \frac{f(z)}{z-a}\, \mathrm{d}z$
15	$x^2 + 2xy + y^2 = \|x\|^2 + 2\|x\|\|y\| + \|y\|^2$	16	$\int_a^b f(x)\, \mathrm{d}x = F(b) - F(a)$
17	$\frac{n!}{r_1! \cdot r_2! \cdots r_k!}$	18	$-b \pm \sqrt{b^2 - 4ac}$
19	$1 + \tan^2 \theta = \sec^2 \theta$	20	$\bar{u} = (x, y, z)$

Table 1: Test query set

Query ID	Relevance Score					Total judged
	0	1	2	3	4	
1	15	2	2	0	1	20
2	19	0	0	0	0	19
3	15	4	1	0	0	20
4	0	0	0	6	14	20
5	0	0	0	8	12	20
6	0	2	5	2	10	19
7	1	4	3	3	9	20
8	5	1	1	1	0	8
9	0	0	4	11	5	20
10	0	0	1	16	3	20
11	0	0	0	1	6	7
12	0	0	4	13	3	20
13	14	3	1	1	1	20
14	0	2	5	8	5	20
15	0	0	0	15	5	20
16	8	6	2	2	2	20
17	0	0	5	13	2	20
18	19	1	0	0	0	20
19	19	1	0	0	0	20
20	19	1	0	0	0	20

Method 1

Query ID	Relevance Score					Total judged
	0	1	2	3	4	
1	13	2	2	1	1	19
2	15	1	0	3	1	20
3	0	0	0	0	20	20
4	0	0	0	1	19	20
5	0	0	0	0	20	20
6	1	0	0	0	19	20
7	0	0	0	0	20	20
8	4	0	1	2	0	7
9	0	2	5	8	5	20
10	0	0	0	9	11	20
11	0	0	0	2	5	7
12	0	0	12	3	5	20
13	15	1	2	1	1	20
14	0	0	0	0	20	20
15	0	0	0	13	6	19
16	0	0	2	0	18	20
17	0	0	0	0	20	20
18	0	0	2	7	11	20
19	0	0	2	7	11	20
20	0	0	2	7	11	20

Method 2

Table 2: Relevance score distribution