A NOVEL SIMILARITY-SEARCH METHOD FOR MATHEMATICAL CONTENT IN LaTeX MARKUP AND ITS IMPLEMENTATION

by

Wei Zhong

A thesis submitted to the Faculty of the University of Delaware in partial fulfillment of the requirements for the degree of Master of Science in Electrical and Computer Engineering

Spring 2015

A NOVEL SIMILARITY-SEARCH METHOD FOR MATHEMATICAL CONTENT IN LaTeX MARKUP AND ITS IMPLEMENTATION

by

Wei Zhong

Approved: ______________________________
Fouad E. Kiamilev, Ph.D.
Professor in charge of thesis on behalf of the Advisory Committee

Approved: ______________________________
Kenneth E. Barner, Ph.D.
Chair of the Department of Electrical and Computer Engineering

Approved: ______________________________
Babatunde A. Ogunnaike, Ph.D.
Dean of the College of Engineering

Approved: ______________________________
James G. Richards, Ph.D.
Vice Provost for Graduate and Professional Education

ACKNOWLEDGMENTS

Thank you to my family for their support from every perspective through out my graduate academic education. Thank you to my advisor Hui Fang who offers me the opportunity to develop my idea further and supports me in many other ways. I am also grateful to all InfoLab members for their kind help. And thanks to those not previously mentioned, who have influenced me or helped along the way.

TABLE OF CONTENTS

List of Tables

List of Figures

ABSTRACT

In this paper, we have addressed the problems of searching content in mathematical language, particularly measuring the similarity degree (in terms of structural and semantical) between mathematical expressions, summarized some general properties from mathematical semantics, that a search engine should be aware of. To better deal with these problems in an efficient way, we propose some ideas including: (1) A list of grammar rules to parse mathematical content (particularly in LaTeX markup) into a tree representation in order to preserve as much as information from mathematical expressions; (2) An index approach to break down the tree representation into what we call branch words to enable fast search in a similar fashion with inverted index, with parallelism potential; (3) A search method to capture some level of query-document subgraph isomorphism, combined with two pruning methods to both speed search and improve effectiveness. We also build our own proof-of-concept prototype search engine to demonstrate these ideas, and thus are able to present some evaluation results through this paper.

Chapter 1

BACKGROUND

Apart from general text content, structured information is also widely contained by digital document. Among these, a lot of mathematical content (including documents on Internet), are represented using markups like LaTeX , MathML [1] or OpenMath [2], which is in a rich structural way. Information Retrieval on those structured data in mathematics language is not that well-studied or exhaustively covered by mainstream IR research, compared to that with general text. Thus it can be challenging yet very helpful given the contribution and importance of mathematics to our science.

However, the structured sense of mathematical language, as well as many its semantic properties (see section 1.2), makes general text retrieval models deficient to provide good search results. Through this paper, we have made our efforts to tackle some of these problems. Some of the ideas used in this paper deals with "tree structured" data in a general way, have the potential to be applied by other fields of structured data retrieval besides that from mathematical language.

1.1 Math IR Domains

Mathematical information involves a wide spectrum of topics, we are, of cause, not focusing on every aspects in mathematical information retrieval. It is good to clarify our concentration in this paper here, by first listing a set of concentrations that a mathematical information retrieval topics may be classified into, and define our target field of study.

[1] http://www.w3.org/Math/

[2] http://www.openmath.org/

Listed here, are considered four possible concentrations of topic for mathematical information retrieval:

1. Boolean or Similarity Search
2. Math Detection and Recognition
3. Evaluation, Derivation and Calculation
4. Other topics

The first one is doing mathematical information retrieval by searching, and finding the most relevant context of documents that match the query, very similar to the most common ways that other general text search engines will do, by boolean or similarity search. The only difference is, the query may contain mathematical expressions. Instances (examples online are SearchOnMath [3], Uniquation [4] and Tangent [5]) of such search engine can be useful in many ways, for example, student may utilize it to know which identity can be applied to a formulae in order to give a proof of that formulae. This is the area where we focus in this paper. Specifically, we are proposing a series of methods for similarity search of math content. And our method is using query only in LaTeX markup (some math-aware search engines [6] support queries in mathematical formulae and normal text together), and return documents ordered by score which indicates the similarity degree.

Digital mathematical content document can also be in an image format (e.g. generated by a handwritten query), thus to retrieve these information involves detection or recognition. Inspired by the advances from deep learning, we may foresee a large potential to be explored on topics related to this.

---

[3] http://searchonmath.com

[4] http://uniquation.com/en

[5] http://saskatoon.cs.rit.edu/tangent/random

[6] WolframAlpha: https://www.wolframalpha.com/ and Zentralblatt math from MathWebSearch: http://search.mathweb.org/zbl/

Because the nature of mathematical language, a query (e.g. an algebra expression) can be evaluated and potentially derived into an alternate form, or calculated. The result value of evaluation or derived form may also be considered being relevant to that query. These potentially require a system to handle symbolic or value calculation, or even a good knowledge of derivation rules implied by different mathematical expression (e.g. computational engine *Symbolab* [7] and WolframAlpha).

Besides the first three concentrations, there are many other topics. Knowledge mining, for example, will need deeper level of understanding on math content. A typical goal of this topic is to give a solution or answer based on information retrieved from math content. e.g. "Find an article related to the *Four Color Theorem*" [1].

These concentrations somehow overlap in some cases, for example, some derivation can be used to better assess the similarity between math formulae, e.g. $\frac{a+b}{c}$ and $\frac{a}{c}+\frac{b}{c}$ should be considered as relevant. Or, mathematical knowledge being used be know the same meaning (thus high similarity) between $\binom{n}{1}$ and C_n^1. Therefore even boolean or similarity search possibly involves certain level of understanding of mathematics. In terms of similarity, however, we only address the measurement for structural and symbol differences in this paper, without considering further topics lured from measuring math content similarity, such as evaluation, derivation or knowledge inference.

As supplementary, [2] gives a comprehensive review on mathematical IR researches and covers many topics across different domains.

1.2 Issues in Measuring Similarity

Unlike general text content, mathematical language, by its nature, has many differences from other textual documents, there are a number of new problems in measuring mathematical expression similarity. Among these, we select and focus on those regarding to structural similarity and symbolic differences between expressions.

[7] Symbolab Web Search: http://www.symbolab.com

At the same time trying to respect the semantical information inferred from structure or symbols in mathematical expressions. But even without caring about the possible derivations and high level knowledge inference, there are still many new problems.

Firstly, differences of symbols, structure and possible semantic rules in mathematics should be captured, and not one by one, but in an cooperative manner to measure similarity. To illustrate this point, we know that only respecting symbolic information is of course not sufficient in mathematical language. e.g. $ax + (b + c)$ in most cases is not equivalent to $(a + b)x + c$ (although they have the same set of symbols). And the order of tokens in math expression can be commutative in some cases but not always. For example, commutative property in math makes $a + b = b + c$ for addition operation, but on the other hand $\frac{a}{b}$ is most likely not equivalent to $\frac{b}{a}$. These make many general text search methods (e.g. *bag of words* model, *tf-idf* weighting) inadequate or inevitability less storage-efficient. Moreover, symbols can be used interchangeably to represent the same meaning, e.g. $a^2+b^2 = c^2$ and $x^2+y^2 = z^2$. However, interchangeability comes with some constrains to maintain the same semantical meaning, that is, 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 $a(1 + b)$.

Secondly, how we evaluate structural similarity between expressions is a question. A complete query may structurally be a part of a document, or only some parts of a query match somewhere in a document expression. In cases when a set of matches occur within some measure of "distance", we may consider them to contribute similarity as a whole, but when matches occur "far away" for a query expression, then under the semantic implication of mathematics, they probably will not contribute the similarity degree in any way. We need metrics to score these similarity under certain criteria and set up standard and rules for relevance assessments.

Lastly, trying to capture semantic information from expressions will help measure similarity but introduce ambiguity. Apart from the cases covered in [3], semantic incorrect written markups, which is somehow common in many online documents, e.g.

writing “sin” in LaTeX markup instead of macro “\sin”, will make it difficult to tell whether it is a product of three symbols or a *sine* function, thus need to disambiguate. And depending on what level of semantical meaning we want to capture, ambiguity cases can be different. Consider $f(2x + 1)$, if we want to know if f is a function rather than a variable, the only possibility is looking for implicit contexts, but we can nevertheless always think of it as a product without losing the possibility to search similar expression like $f(1+2y)$, the same way goes reciprocal a^{-1} and inverse function f^{-1}. Most often, even if no semantic ambiguity occurs, efforts are needed to capture some semantical meanings. e.g. In $\int f(x)\frac{\mathrm{d}x}{\sin x}$ and $\sin 2\pi$, it is not easy to figure out, without a little knowledge on integral or trigonometric function, that integral is applied to $\frac{f(x)}{\sin x}$ and the scope applied by sine function is 2π, if we want to capture the subordinative relationship information.

1.3 Related Work

Boolean or similarity 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 [4]. And a variety of approaches have already emerged in an early timeline [5]. But there are a limited number of main ideas, from different angle, to deal with mathematical structured data. [6, 7, 8] use the same way to classify them into text-based and tree-based (structure-based). Here we follow the same classification (long with approaches different from this two) and give a recap and an overview on their core ideas.

1.3.1 Text-based methods

Many researchers are utilizing existing models to deal with mathematical search, and use texted-based approaches to capture structural information on top of matured text search engine and tools (such as *Apache Lucene*).

DLMF project from NIST [9] uses “flattening process” to convert math to textualized terms, and normalize them into *sorted parse tree normal form* which creates a unique form for all possible orders of nodes (e.g. in a associative or commutative operator). Then further introduces serialization and scoping to stack terms [10], trying to capture structure information by using text-IR based systems that supports phrase search. Similar idea is also used by [11], expressions are also augmented for various possible representations, but variables are also replaced and normalized, but they are using postfix notation, allows to search subexpressions without knowing the operator between them. MIaS system [12, 13, 14], like the methods above, are also trying to re-order commutative operations, normalize variable and constance into unified symbols, doing augmentation in a similar fashion. It indexes expressions and subexpressions from all depth levels, and system is able to discriminate assign different weight based on their generalization level, and place emphasis in which a match in a complex expression is assigned higher weight [14].

Augmentation usually consists a storage demand for combination of both symbols (e.g. a and b) and unified items (*id, const*) in different levels, in order to capture both symbolic information and structure information. Thus implies complex expressions with many commutative operators will cost a lot of storage space, the benefits of capturing expression variances will be overshadowed.

Although named as structured-based approach, [15] is using *longest common subsequence* algorithm to capture structure information (in a unified *preprocessed string* and a *level string*). The method takes $O(n^2)$ complexity for comparing a pair of formulae, and no index method is proposed. Therefore is not feasible to efficiently apply to a large collection.

The Mathdex search engine [16], from another perspective, uses query likelihood approach [17] to estimate how likely the document will generate the query expression by a n-gram from root expression to sub-expression and tokens.

Math GO! [18] is another system advances some transitional method to better search math content. It tries to find all the symbols and map formula pattern to pattern name keywords (like *matrix* or *root*), and proposes to replace term frequency by co-occurrence of a term with other terms.

1.3.2 Structure-based methods

What text-based methods share in common is they are converting math language symbols to bags of searchable words, the intrinsic defect when using a bag of words to replace structured information will make conversion process lose considerable information or cause storage-inefficiency. In order to cope with the problems from text-based approach, structure-based methods generally generate intermediate tree-variance structure, and use these information to index or search.

Term Indexing

Whelp [19] and MathWebSearch (MWS) directed by Kohlhase [20, 21, 22], derived from *automatic theorem proving* and *unification theory* [23]. The system of MWS uses *term indexing* [24] in a *substitution tree index* [24] to to minimize access time and storage. Because the subexpression is not easy to search using substitution tree, MWS indexes all sub-terms, but the increased index size remains manageable [20]. However, their index relies on RAM memory, even scaling can accommodate nearly entire arXiv site (72% paper on arXiv), the RAM usage will be 170 GB [22], which already needs a considerable hardware resources.

Leaf-root path

[25] uses leaf to root XML path in a MathML object to represent math formula. When efficiency is considered, it only indexes the first and deepest path (to indicate how a formula is started and presumably the most characteristic part of a formula); when user wants to obtain the perfect-match result, it indexes all the MathML object leaf-root path. The boolean search is performed by giving all the paths match with those of the search query. [26] further develops with incorporation of previous method using breath-first search, to add sibling nodes information into index and have achieved better effectiveness.

Very similar idea is proposed by [27] and used in [28]. The authors of latter transform MathML to an "apply free" markup from which the leaf-root path are extracted. Leaf-root path is also used to evaluate similarity between MathML formula.

Symbol layout tree

A *symbol layout tree* [29] (SLT) or *presentation tree* [6] describes geometric layouts of symbols in a formula. WikiMirs [6] 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. Term level, and df-idf idea of factors are used in scoring. [29] uses symbol layout tree as a kind of substitution tree, each branch in the tree represents a binding chain for variables, and every child node is an instance of its parent for a generalized term. They introduce *baseline size* to help group similar expressions together in their substitution SLT in order to decrease tree branch factor, however, this makes a single substitution variable not able to match multiple terms along the baseline. Also their implementation makes it unable to index certain formula (e.g. a left-side superscript) and have to generate many queries (e.g. all possible format variations and sub-expressions etc.) for a single query at the time of search. Furthermore, to differentiate similarity on boolean results, they use bag of words model in ranking. Later [7, 30] have developed a *symbol pairs* idea to capture a relative position information between symbol pairs. Due to the many possible combinations of symbol pairs in a complex math expression, the storage cost is intrinsically large. However, the key-value storage style will be suitable for fast lookup (e.g. HASH).

Other structure-based methods

A novel indexing scheme and lookup algorithm is proposed by [31], its index has hash signature for each subtree because they have observed a lot of common subtree structure occur in math formula collection. This idea will result in a slower index growth. Their lookup algorithm supports wildcards, and performs a boolean match test. Although their lookup time is generally below 700ms, the index size where query lookup time is tested is unclear, but presumably no greater than 70,000 expressions. By constructing a DOM tree, [32] extracts semantic keywords, structure description to indicate subordinative relationship in a string format. The similarity is calculated using normalized tf-idf vector (trained by clustering algorithms) by dot product. Although the final index is generated from text, promising results have been achieved. Tree edit distance is adopted by [33], it tries to overcome the bad time complexity of original algorithm by summarizing and using a structure-preserving compromised edit distance algorithm using heavy path. Although the result shows query processing time is long but it is reduced to average 0.8 seconds by applying with an early termination algorithm along with a distance cache [34].

1.3.3 Other related work

There are a number of articles trying use image to assess similarity. [35] compares their image-based approach using connected component-based feature vector with a proposed text-based method, reported precision@k values are low, but the potential for this method to be combined with shape representations or other features will potentially improve it and make it valuable for measuring similarity for image mathematical expressions. [36] uses X-Y tree to cuts the page in vertical and horizontal directions alternatively, in order to retrieve math symbols from images, then sub-image matching is performed, this method is intuitive, yet too expensive for regular document with markup language.

A lattice-based approach [37] build formal concept based on selected feature sets of each formula. The ranking is calculated by the distances from query in the lattice map when the query is inserted.

1.3.4 Performance Review

In the review of many related past research in section 1.3, we find the combination of state-of-art general text search engine (i.e. *Apache Lucene*) with the efforts to augment expressions by permutation and unification to satisfy the needs of mathematical search have achieved a good result in different metrics of evaluation: The text-based system of MIaS over-performs those of structure-based and ranked the first in four out of six measurement in NTCIR-11 conference [38]. However, structure-based method has its potential and merits too. Among all the structure-based methods, [7, 30] which use symbol pairs idea on layout tree is very promising and get a competitive result [38].

Chapter 2

METHODOLOGY

Our method can be seen as an approach built upon the idea of leaf-root path or sub-path (section 1.3.2), in an operation tree [2]. But we have developed this idea further in many ways. Our index is composed from leaf-root paths from mathematical formulae operation tree. The search method is traversing a “reversed” sub-path tree, coming along with a pruning method and a proposed sub-structure test algorithm, which are utilizing some observed properties from our indexed tree. Apart from these, we also offer several rules of constrains to measure symbolic similarity. This chapter gives a summary on the method intuition and the core ideas behind these.

The methods in a nutshell is, for a document expression, construct operation tree and break it down into sub-paths, index those paths by inserting them into a tree-structured index by their reversed order. For a search query, traversal index tree as the same way of going through the reversed sub-paths of that query (search path), get the search results along the merged ways from different search paths. Finally apply symbol similarity measurement algorithm or the sub-structure test algorithm to rank results.

2.1 Intuitions

First it is beneficial to document our intuitions on using operation tree as our intermediate representation and our idea to index it in a way of reversed sub-path tree, and also explain in abstract why this way helps reduce index space and boost search speed. We will give an illustrative example to describe these processes further in section 2.4.4.

2.1.1 Commutative immunity

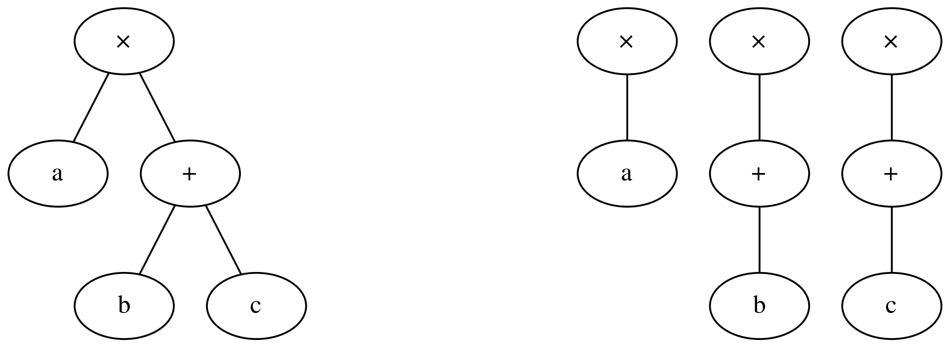
Operators with semantic implication of commutative property (e.g. addition and multiplication) are exhaustively used in mathematical language. The ability to identify the identical equations for any permutation is very essential for a mathematical similarity search engine. Given this as a start point, the leaf-root paths have the advantage to cope this so that we do not need to generate different order of patterns to match formulae with commutative operator. To illustrate this, we know that a leaf-root path from an operation tree (see figure 2.1) is generated through traversing in a bottom-up (or top-down) fashion from a tree, thus path string is independent with the relative position of operands from same father node. In another word, an operation tree uniquely determines the leaf-node paths decomposed from the tree, no matter how operands are ordered.

2.1.2 Sub-structure query ability

On the other hand, the structure of operation tree also makes it easy to represent sub-expression relation with a formula, because a sub-expression in a formula is usually (depending on the way we construct an operation tree) also a subtree in an operation tree. And by going up from leaves of operation tree, we are essentially traversing to an expression from its subexpression for every level. By making all the leaf-root paths as an index, we can search an expression by going through and beyond the leaf-root paths from its subexpression. This makes operation tree better in terms of searching an expression given a sub-expression in query. And it avoids information augmentation on index as some other structure-based methods need to do (e.g. index all sub-terms of an expression in MWS [20]). Therefore it helps save storage space.

2.1.3 Index and search properties

Additionally, some properties from the "inverted" of sub-paths (we will illustrate this in section 2.4.4) from an operation tree suggest some reduce of space and pruning possibilities in search process. First the sub-paths themselves can be indexed into a



$a(b+c)$ in operation tree

Generated leaf-root paths

Figure 2.1: Leaf-root path example

tree so that we can search a sub-path by traversing a sub-path tree, instead of hashing it to find a corresponding value as the symbol-pair search engine (i.e. *Tangent* [7]) does. This allows us to save a lot space as the reverted sub-paths of a large collection will have a great percentage of level sharing a common string with each other. Also the way to search in a tree structure with a limited branch factor does not lose much efficiency compared to the HASH methods used in *Tangent*, while also offer great storage efficiency. Second, by searching from all the "reverted" sub-paths of a query expression in our proposed index, and apply an intersection on the results from different sub-paths, we will find all the expressions have that query as subexpression (number four observed property from section 2.2.2). And during this search process, upon going further from the query expression root in the "reverted" sub-path, we can merge the next search directories by pruning all the entries that are not shared in common among all the "reverted" search path directories. Further more, multiple index search in different path level are independent with each other, put in another way, if a given indexed formula has been found in one search path level, then its other relevant sub-paths (in terms of the current query) will most likely be found at the same search level too, thus some implementation strategies can be applied to speed search further (i.e. distributed search to quick search massive), which we would address in the next chapter.

2.2 Structure Similarity

The basic ideas used in our approach, to test whether a mathematical expression is an sub-structure of another, to prune and to constrain search process are the foundation work in our research. It is desired to give a description in a formal language so that we can deliver these ideas in the most precise way. Some important observations as well as brief justifications are provided after definitions.

2.2.1 Definitions

For the second issue addressed in section 1.2, specifically, to assess the structural similarity. Previous formal definitions [39] have been given on this, providing a quantified n-similarity relation to address the similarity degree, which is determined by the max-weight common subtree between two formulae. The subtree, by their definition, includes all descendants from a node. Nevertheless, we are going to use the subtree definition in graph theory here to describe the sub-structure relation. To be explicit, given a rooted tree T, the connected graph whose edges are also in T is defined as the subtree of T.

Also we need clarify some conventional notations here. Through this paper, a path $p \in \mathbf{P}$ is a sequence of numbers given by $p = p_0 p_1 \dots p_n$ where $n \geq 0$, $p_i \in \mathbf{R}$, $0 \leq i \leq n$ and $\mathbf{P}$ is the set of all paths, including empty path. Any function $f(\cdot) = y \in \mathbf{R}$ applied on p is mapped to a path too: $f(p) = f(p_0)f(p_1)\dots f(p_n)$. Furthermore, for two paths ${}^1p = p_0 p_1 \dots p_n$ and ${}^2p = p_n p_{n+1} \dots p_m$ where $m \geq n$, a concatenation ${}^1p \cdot {}^2p$ is defined as $p_0 p_1 \dots p_n p_{n+1} \dots p_m$ (said as concatenation of ${}^2p$ on ${}^1p$), and the concatenation of a path p on a set $S = \{s_1, s_2 \dots s_n\}$ is defined as $S \cdot p = \{s_1 \cdot p,\ s_2 \cdot p\ \dots\ s_n \cdot p\}$. Usually a path with only one element is explicitly stated and wrapped by a bracket, e.g. $p = (p_0)$. Lastly, *the longest common postfix* path p^* between two path p_1 and p_2 is mapped by the function lcp: $p^* = \mathrm{lcp}(p_1, p_2) = \mathrm{lcp}(p_2, p_1)$.

Based upon this, we introduce a *formula subtree* relation to address the substructure relation between two mathematical expressions. The formula tree is associated with a label (labels are not required to be distinct here) in each node to represent a mathematical operator, variable, constance etc., also a symbol value in each leaf node to represent a symbolic instance of that constance or variable (e.g. "123", β, x, y etc.). Below are our formal definitions.

2.2.1.1 Formula tree

A *formula tree* is a labeled rooted non-empty tree $T = T(V, E, r)$ with root r, where each vertices $v \in V(T)$ is associated with a label $\ell_T(v) \in \mathbf{R}$ 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 function names which refer to the tree implied by the context, also use function $\mathcal{S}(p)$ to indicate the symbol of the leaf in a leaf-root path p.

2.2.1.2 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 $S \to T$.

2.2.1.3 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 $a = \ell(p),\ p \in g(T)$, we say T is indexed in Π with respect to a. Where $\mathcal{I}_\Pi$ is called an index look-up function.

2.2.1.4 Leaf-root path set

Lastly, 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 $g(T)$. Therefore we have $p \in g(T)$ for any leaf-root path p of tree T.

2.2.2 Observations

We have concluded the following observations from which the interpretation can be implied that gives us some insights into the structural properties of formula tree. Our structural isomorphism test algorithm and search method are based upon these.

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:

$$\deg(v) \leq \deg(\phi(v)) \tag{2.1}$$

$$\ell(p) = \ell(\phi(p)) \tag{2.2}$$

$$|g(T_q)| \leq |g(T_d)| \tag{2.3}$$

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 2.2.1.2. Therefore any different edge from v is associated with a distinct edge from $\phi(v)$, thus we can get (2.1). Given the fact that every non-empty path p can be decomposed into a series of edges $(p_0, p_1), (p_1, p_2) \ldots (p_{n-1}, p_n)$, $n > 0$, property (2.2) 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 (2.3) can be obtained in a similar manner with that of (2.1), expect neighbor edges are replaced by leaf-node paths.

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.

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 $a_0 = b_0 = r$, we 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 the path of $b = b_0 \dots b_n b_{n+1} \dots b_{m-1}$, 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 see $|\text{lcp}(a, b)| = |\text{lcp}(a', b')| = n$. 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.

Observation #4

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 2.2.1.2 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 $\forall\ a \in L,\ T_d \in \mathcal{I}_\Pi(a)$.

2.2.3 Interpretation

The observations above offer some insights on how to test a substructure of a mathematical expression and how to search for an indexed mathematical expression.

First we give some explanations on the definition. A formula subtree relation defined in 2.2.1.2 describes not only a sub-structure relation between two math expressions, it also requires a label similarity and leaf inclusion. Because structure shape (subtree isomorphic) is not only one factor to determine whether a math formula is a subexpression of another. Given expression in figure 2.1 as an example, where $b + c$ is an subexpression of $a \times (b + c)$, and we consider "similar" between the two. However, if expressions with different symbols but in similar semantics are given, e.g. $b \oplus c$ or $b \pm c$, they should also be considered as similar to $a \times (b + c)$ because both the operations has the similar semantical meaning as "add". These operations should be labeled the value of which all the similar operation tokens are the same. Also, operation tree

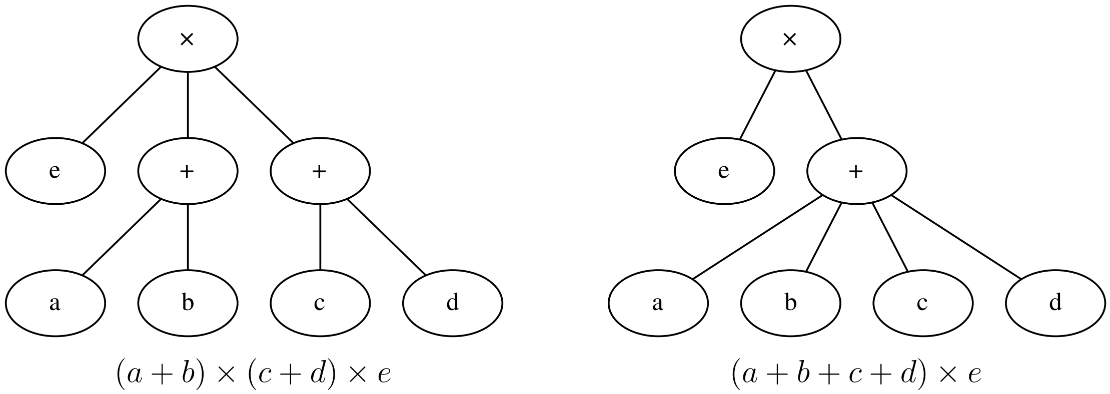


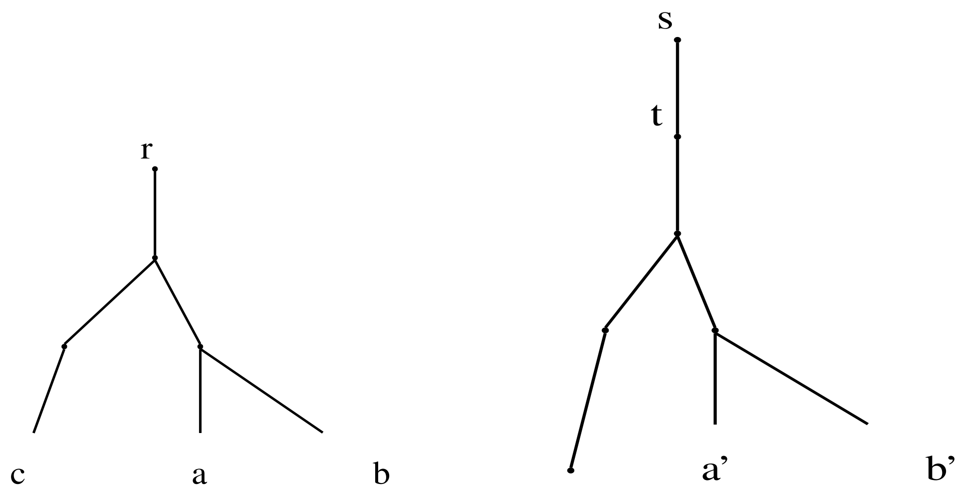
Figure 2.2: Leaf-root paths with different structure

representation generally puts operator in the intermediate nodes and operands in the leaves, so it is not common to address a sub-structure without leaves, like "$a \times +$". So a structure-similarity relation of two should also contain their leaves.

Now that we have defined our structure similarity rule as whether two trees T_q and T_d can satisfy: $T_q \preceq_l T_d$. We break down a formula tree into leaf-root paths p and index the label of each path $\ell(p)$. So if given a "similar" path q, we can further find the previous trees that also have $\ell(q)$ as its labeled path.

In section 2.2.2, the first observation gives some constrains to test if two leaf-node paths are similar without knowing the complete tree from which they are generated. However, comparing all the paths from the index one by one would be very inefficient. Observation #4 suggests if we search the index by all the generated leaf-node paths from a tree at the same time, then we may just need to look into an intersected region instead of the whole collection. Because every tree indexed ($T_d \in \Pi$) and matched by the query will be found at the intersection of index with respect to paths starting from each query leaf-root, furthermore, the search paths from these start points (indicated by $\ell(g(T_q))$ set), is the same (indicated by $\hat{a}$). Therefore we can "merge" the paths ahead and prune those paths not in common. Level by level, we will finally find the matched tree.

However, knowing the matched tree in in a set does not necessarily mean all the



(a) (b)

Figure 2.3: Formula subtree matching

tree in the set match with query. Figure 2.2 gives one case where two set of leaf-root paths are identical while the structures from which they are generated are different, and not in any sub-tree relation. If left figure is the query, then we can certainly find the tree in right figure as long as it is indexed, indicated by observation #4 in section 2.2.2. Although leaf-root paths offer some desired properties, whether the trees found through searching sub-paths of a query are also structure isomorphic with the query tree is still unknown.

Observations #2 and #3 in section 2.2.2 offer the way to test structure isomorphic. The former is a sufficient condition to test structure isomorphic, but the tree must first have only one leaf-root path. The latter states two necessary conditions to be a formula subtree of another. This leads to an idea to decompose the tree and divide the problem into subtree matching problems by ruling out impossible matches between leaf-root paths using observation #3, until it is obvious to conclude the structure isomorphic in a sub-problem by using observation #2.

2.2.4 Structure match

Here we propose and describe an algorithm for formula subtree matching based on the interpretation in section 2.2.3. Figure 2.3 illustrates a general case in which query tree (a) is trying to match a document tree (b).

Initially every leaf-root path in (a) should be associated with a set of leaf-root paths in (b) that are possible (by the constrains of observation #1 in 2.2.2) to be isomorphic, we call this set candidate set. For example the candidate set of path a in (a) probably is $\{a',\ b'\}$ in (b) if the nodes are assigned universally the same label. Then we arbitrarily choose a path in (a) as a reference path (heuristically a *heavy path* [40]), 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 observation #3 and ruling out some impossible isomorphic paths in candidate set of each path in (a) and divide the problems further. For example, because $|\mathrm{lcp}(a, b)| = |\mathrm{lcp}(a', b')|$, we know b' is still in candidate set of b; while b' is not in candidate of c anymore because $|\mathrm{lcp}(a, b)| \neq |\mathrm{lcp}(a, c)|$. 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 very simple subproblem is encountered, that can be solved by observation #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 evaluated.

The detailed algorithm is described in figure 2.4. The main procedure is *decomposeAndMatch* where the argument Q and C is the set of leaf-root paths in query tree and the candidate sets associated with all leaf-root paths respectively. The procedure returns SUCC if a match is found, otherwise FAIL is returned indicating no possible match.

```
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 2.4: The decompose-and-match algorithm

2.3 Symbolic Similarity

Until now, we have not addressed symbolic similarity yet. Although mathematical expression often use symbols interchangeably, symbolic matches is a good way to differentiate similarity, and most importantly, measure some semantic similarity in mathematical language.

Firstly, structure similarity of math expression is either boolean match (subtree or not) or measured with the similarity degree only depend on the subtree depth where expressions are matched. Symbolic similarity will introduce more factors to further distinguish similarity among structural identical math expressions. Also it is essential to give those with symbolic matches a higher rank because they may imply more semantic similarity. For example, $E = mc^2$ is considered more meaningful when exact symbols are used rather than just being structure identical with $y = ax^2$.

Secondly, as illustrated in section 1.2, same mathematical symbols in an expression (or bind variables) usually can only maintain semantical equality if the changes are made by substitutions. (similar to the notion of α-equality [41]). This is an important semantic information that we need to capture and certainly it involves comparison of symbols.

Yet there is one thing to notice here, in many mathematical search systems, a query may be specified with wildcards and thus will match any document with an expression substitution to that wildcard. And a query symbol not specified by wildcard is expecting an exact symbolic occurrence in document. We are not considering wildcards here with the limitation of our substructure matching method. And in terms of symbolic wildcard, we also doubt the its demand in mathematical search as it is not common to expect an exact symbol occurrence when we query in mathematical language (also addressed in [12]).

2.3.1 Ranking constrains

As we have discussed, symbolic similarity is essential to be captured, in order to further rank document expressions. Here we propose two constrains to addressed all the considerations, they can be summarized as:

- A document expression with both structure and symbol matches (not necessarily all the symbols) is considered more relevant than that with only structure matches. And the more symbol matches, the more relevant it is.
- Expressions with identical symbols at the same place (i.e. bind variable) should be considered more similar than those with different symbols at the same position.

In this paper, we use these two constrains as the rule to rank retrieval results in terms of symbolic similarity. And in cases where both constrains can be applied, we prioritize the second constrain. This is because, intuitively, as long as the semantic meaning of two expressions is the same, using different set of symbols does not make a difference. However, bind variable match is more important because an mathematical expression with more than one identical symbols most often imply that those symbols represents the same variable. Missing one or more symbols being identical will lose semantics in a certain degree.

The constrains and idea above are illustrated by the following example. Let the rank of a document math expression d be $r(d)$, and given query $\sqrt{a}(a-b)$ for instance. It is easy to see under the first constrain:

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

The one with the highest rank here is an exact match, with three symbols matching in total. The second one has two symbols match while the third one has no symbolic match at all.

In the same manner, by the second constrain we have:

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

The first one uses bind variable x but it preserves the same semantics except for the symbol of bind variable is different with that of query. The second one does not have bind variable match, in another word, it uses different symbols (i.e. "x, y") at positions where query expression uses the same symbols (i.e. "a").

One common pattern the first example follows is they all have the bind variable match. And for the second example, they have the same number of symbol matches (only "b" is matched in a symbolic way). So it is easy to follow only one of the two constrain. However, sometimes both constrains can be applied where conflict may occur and we have to choose only one to follow. Given document expression $\sqrt{a}(x-b)$ and $\sqrt{x}(x-b)$ for instance, the former has two symbolic matches (i.e. "$a,\ b$") while does not have bind variable match. The latter, on the other hand, has bind variable match while only has one symbolic match (i.e. "b"). We nevertheless score the latter higher because it does not lose any semantics.

2.3.2 An algorithm

Here we propose an algorithm to score symbolic similarity between query and document expressions. To follow all the constrains and issues addressed, intuitively, we first take the bind variable with greatest number of occurrence, e.g. "a" with three occurrences in $\left(b \cdot \dfrac{a+b}{a+c} + a\right)$, to match as many as symbols of each bind variable in a document expression in a greedy way. Whenever a symbol in document expression is matched, we exclude it from matching candidates in future iterations. In the next iteration, we choose the bind variable with the second number of occurrence and repeat this process until all the query bind variable are looped.

The algorithm is described in figure 2.5. It 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. The bind variables in D is defined by a set $V(D) = \{x \mid \mathcal{S}(x),\ x \in D\}$, which contains all the leaf node symbols from document expression. Function *sortBySymbolAndOccur* takes the elements from a set of leaf-root paths and return a list of all the leaf-root paths, each path p is ordered by the tuple $(\mathcal{S}(p), O_p)$ in the list, where O_p is the number of occurrence of path p in the set. Take the example expression $\left(b \cdot \dfrac{a+b}{a+c} + a\right)$ again, the result list returned by *sortBySymbolAndOccur* is a sequence of leaf-root path of a, a, a, b, b, c respectively. Each document path a' is associated with a tag $T_{a'}$ which has three possible state:

```
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
        if S(a) has 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 | y = v, y ∈ V(D)} do
                    if T_{a′} = mark then
                        T_{a′} := nextState
    return score
```

Figure 2.5: The mark-and-cross algorithm

marked, unmarked and crossed. And bind variable $v \in V(D)$ can be given a score B_v which represents the similarity degree between two bind variables. The symbolic similarity function $\mathrm{sim}(a, a')$ measures the similarity degree between two leaf-root paths a and a'. Intuitively, we set the similarity function:

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

to give greater (double) weight to leaf-root paths with exact symbol match than those without symbol match but in a candidate set (this function is called only when a document path is in candidate set, see figure 2.5). Furthermore, denote the number of variables in a bind variable x in query expression that best matches the bind variable y in document expression as $n_{x,y}$, then for the bind variable a in query expression and two bind variable a and b in document expression, if $n_{a,b} > n_{a,a}$ then we weight the bind variable matching of a in query with b in document more than that of a in query with a in document, even if the latter has exact symbol matches. For example, for query expression $a + \frac{1}{a} + \sqrt{a}$ and document expression $a + \frac{1}{a} + b + \frac{1}{b} + \sqrt{b}$, we have bind variable matching $\boxed{\mathrm{a}} + \frac{1}{\boxed{\mathrm{a}}} + \sqrt{\boxed{\mathrm{a}}}$ with $a + \frac{1}{a} + \boxed{\mathrm{b}} + \frac{1}{\boxed{\mathrm{b}}} + \sqrt{\boxed{\mathrm{b}}}$ weighted more than exact symbol matching $\boxed{\mathrm{a}} + \frac{1}{\boxed{\mathrm{a}}} + \sqrt{a}$ with $\boxed{\mathrm{a}} + \frac{1}{\boxed{\mathrm{a}}} + b + \frac{1}{b} + \sqrt{b}$ (expressions surrounded by a box indicate the matching part), because the former matching has more variables involved even if they are not symbolic identical with its counterpart. That is to say, given a bind variable matching with k variables of that bind variable in query, we need α to satisfy $k\alpha > k - 1$ and $\alpha < 1$. In our practice, we set α to a value close to 1, like 0.9 for instance.

By sorting the query paths Q, the algorithm is able to take out paths from same bind 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 $\mathrm{sim}(a, a')$ 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 which has maximum similarity value. Once a query bond variable has been iterated (line 21), we find the document variable m_v with greatest B_v value m, and regard it as the best match bond variable in document for the bond variable just iterated, and use m to contribute total 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 rollback those marked paths that are not variable m_v to be unmarked. We continue doing so until all the query path is iterated, and finally return the score indicating the symbolic similarity between query and document expression.

2.4 Combine the Two

We have already discussed and proposed the problem and algorithms to both test structure isomorphic and measure symbolic similarity between mathematical expressions. The two algorithms, however, do not cooperate in an unified way. To illustrate this point, assume we first use decompose-and-match algorithm and have concluded one is a formula subtree of another, but we only get one possible substructure match. There are very likely other possible positions where this tree can also be formula subtree of another, because the candidate sets are not unique. Thus we need to exhaust all possibilities and apply mark-and-cross algorithm to each of them, in order to find the maximum symbolic similarity pair. Analogously, assume we first want to get the symbolic similarity degree, then we are uncertain about the paths we have specified in candidate set are really isomorphic paths to the query path. We will not guarantee substructure relation before using decompose-and-match algorithm.

One possible way to both test strict structure isomorphism and measure symbolic similarity is to decompose the tree trying to match all the substructures but at the same time heuristically choose reference paths to achieve best symbolic similarity in a greedy way, if no possible substructure can be matched isomorphically, we have to rollback and try other candidates using backtracking. Because the mark-and-cross algorithm has an worst case time complexity of $O\big(|Q| \times |D|\big)$, trying to find the maximum point while introduce more parameters (we try to find the one with not only the most symbolic similarity but also who satisfies structure isomorphism) would lead to even more time complexity.

2.4.1 Relaxed structure match

The complexity introduced to combine the two methods will make our approach infeasible to efficiently deal with large data set. Here we choose to relax our constrain on strict structure isomorphism. As figure 2.2 has illustrated, we know that the labels of a leaf-node path set being a subset of that of another does not necessarily mean the tree generating the former path set is a formula subtree of that generating the latter. To further generalize it, we say 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 considering those cases will offset the benefit to strictly identify the structure isomorphism. Therefore, we loose our constrain on structure similarity so that any $T_d \in \bigcap_{a \in L} \mathcal{I}_\Pi(a)$ is considered structurally relevant to query formula tree T_q in observation #4 of section 2.2.2, and we say T_d is *searchable* by T_q in index Π. Optionally, we can apply constrain 2.1 in observation #1 to further eliminate cases such as the one in figure 2.2 thus less query/document expressions that are not in formula subtree relation would be considered relevant. We name this constrain as *fan-number constrain*.

The revised method will collect all the "structure similar" document formula trees that satisfy the fan-number constrain in the set $\bigcap_{a \in L} \mathcal{I}_\Pi(a)$. Then use mark-and-cross algorithm to finally rank the collected set by symbolic similarity.

2.4.2 Matching-depth

On the other hand, we may introduce a *matching-depth factor* into symbolic similarity measurement algorithm, to make it also consider some structurally matching depth. As it is addressed in [12], the deeper level at where two formula matches, the lower similarity weight would it be, since 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 math expressions matches, we introduce the matching-depth factor $f(d)$ and modify the similarity function in the mark-and-cross algorithm:

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

where the factor value $f(d)$ is in a negative correlation with matching depth $d = |\hat{a}|$ (See observation #4 of section 2.2.2). Many possible functions may be adopted, we use $f(d) = \dfrac{1}{1+d}$ in our method.

The idea behind this is because we accumulate $\text{sim}(a, a')$ value to contribute the overall symbolic similarity score returned by algorithm 2.5. In order to incorporate mark-and-cross algorithm, by distributivity, we can distribute the factor of matching depth for two mathematical expressions, over all the path pairs generated from the two expressions.

2.4.3 Matching-ratio

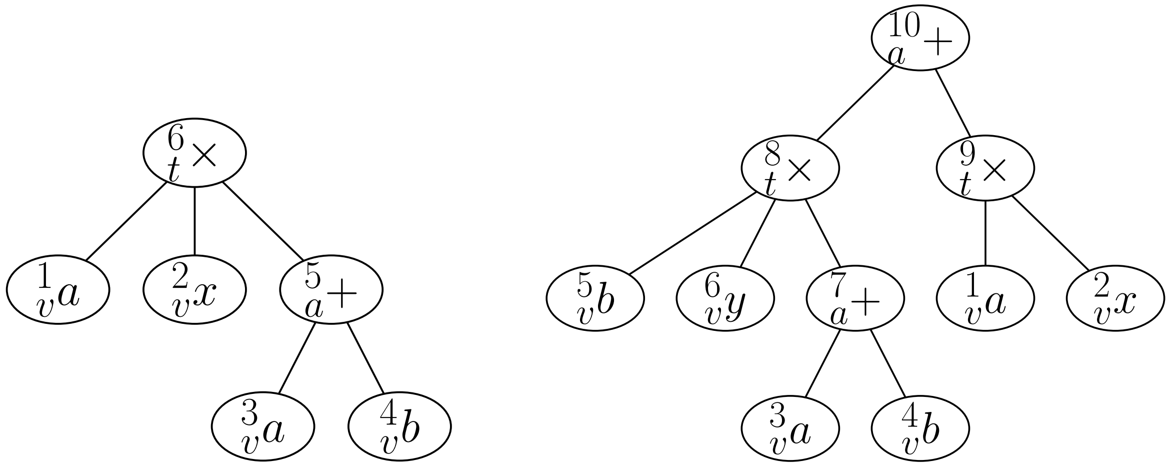
Before matching-depth is introduced, if a formula subtree relation is inferred between two formula trees, they are considered structurally matching in a boolean manner. There is no similarity degree between a complete match (or a sub-formula) and structurally irrelevant. In another word, structural differences is considered in a way to filter indexed expressions, for further symbolic similarity assessment. The final output ranking is determined by symbolic similarity degree, without considering structural similarity. Here we introduce another metres that measures the structural similarity, combined with symbolic similarity to rank final search results.

According to the property 2.3 in section 2.2.2 (observation #1), for two expression in formula subtree relation, we have $\frac{|g(T_q)|}{|g(T_d)|} \leq 1$, and the ratio of left-hand is named as *matching-ratio*, which characterises the structural coverage for a matching query in an expression. Consider the scenario where a query expression is structural isomorphic to two document expressions, and the symbolic similarity between them are the same. However, the different ratio of the document expression "area" to the "area" which the matching query covers essentially makes them scored differently. For example, for query $ax + b$, document expression $ax + b$ should precede $x^2 + ax + b$ although the query matches both two document expressions with same symbolic score. Therefore, in addition to the symbolic similarity degree s given by mark-and-cross algorithm, we combine matching-ratio r to score overall similarity and rank document expressions by the tuple (s, r), that is, first compare s and then compare r if s is equal, to rank document expressions.

2.4.4 Illustrated by an example

We illustrate the method introduced in this chapter by a simple example here. Given a query expression $ax(a + b)$ and document expression $ax + (b + a)by$, here we show how we search the relevant document using this query and how the relevance score between them is calculated by mark-and-cross algorithm.

The query expression and document expression are represented by operation trees T_q and T_d in figure 2.6, instead of only denoting the operation symbols at the internal nodes and the variable (also can be a constant) symbols at the leaf nodes, we use the notation ${}^{i}_{l}S$ to denote a node with symbol S labeled by l (i.e. $\ell(S) = l$) with vertex number i. The three different possible labels here are v, t and a, standing for unified name "variable", "times" and "add" respectively. To be concise and descriptive, we will interchangeably use either ${}^{i}_{l}S$ notation or q_i (p_i) to represent the leaf-root path in a query (document) operation tree where the leaf vertex i resides.



query expression $ax(a+b)$ document expression $ax+(b+a)by$

Figure 2.6: Example query/document expression representation

Firstly, the generated path sets for T_q and T_d are:

$$g(T_q) = \{1 \cdot 6,\ 2 \cdot 6,\ 3 \cdot 5 \cdot 6,\ 4 \cdot 5 \cdot 6\}$$

$$g(T_d) = \{5 \cdot 8 \cdot 10,\ 6 \cdot 8 \cdot 10,\ 3 \cdot 7 \cdot 8 \cdot 10,\ 4 \cdot 7 \cdot 8 \cdot 10,\ 1 \cdot 9 \cdot 10,\ 2 \cdot 9 \cdot 10\}$$

And the labeled path sets for each of the two are:

$$\ell\left(g(T_q)\right) = \{v \cdot t,\ v \cdot a \cdot t\}$$

$$\ell\left(g(T_d)\right) = \{v \cdot t \cdot a,\ v \cdot a \cdot t \cdot a\}$$

Because $\ell\left(g(T_q)\right) \cdot a = \ell\left(g(T_d)\right)$, or equivalently

$$\ell(g(T_q)) \cdot \hat{a} \subseteq \ell(g(T_d))$$

where $\hat{a} = a$, we know T_d is searchable by T_q, so we will find a path $\hat{a}$ to append after the labeled path set $\ell\left(g(T_d)\right)$ in index Π and obtain T_d to be considered as structurally relevant to T_q. We can also infer the matching-depth d here is $|\hat{a}| = 1$.

Secondly, by the implications from observation #1 of section 2.2.2, we get the candidate set for each of the path in T_q:

$$C_{q_1} = \{p_5,\ p_6\}$$
$$C_{q_2} = \{p_5,\ p_6\}$$
$$C_{q_3} = \{p_3,\ p_4\}$$
$$C_{q_4} = \{p_3,\ p_4\}$$

In addition, get the list L containing all the query paths in T_q sorted by symbol and its occurrence in all path symbols,

$$L =^1_v a,\ ^3_v a,\ ^4_v b,\ ^2_v x.$$

Lastly, we can calculate the symbolic similarity degree between the two expressions by going through each query path in L and apply mark-and-cross algorithm. Let factor function $f(d) = \dfrac{1}{1+d}$ (so that matching-depth factor is 0.5), and $\alpha = 0.9$.

Our system Cowpie [1]

MathML vs LaTeX

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

Contact author: clock126@126.com or http://www.eecis.udel.edu/~zhongwei

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. Accessed: 2015-03-31.

[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] Richard J, Fateman, and Eylon Caspi. Parsing tex into mathematics. *SIGSAM Bulletin (ACM Special Interest Group on Symbolic and Algebraic Manipulation)*, 1999.

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

[5] Jozef Misutka. Mathematical search engine. Master's thesis, Charles University in Prague, May 2013.

[6] 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.

[7] 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.

[8] Qun Zhang and Abdou Youssef. An approach to math-similarity search. *Intelligent Computer Mathematics. International Conference, CICM*, 2014.

[9] Miller B. and Youssef A. Technical aspects of the digital library of mathematical functions. *Annals of Mathematics and Artificial Intelligence 38(1-3), 121136*, 2003.

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

[11] Jozef Miutka and Leo Galambo. Extending full text search engine for mathematical content. *Towards Digital Mathematics Library.*, 2008.

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

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

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

[15] P. Pavan Kumar, Arun Agarwal, and Chakravarthy Bhagvati. A structure based approach for mathematical expression retrieval. *Multi-disciplinary Trends in Artificial Intelligence*, 7694:23–34, 2012.

[16] Robert Miner and Rajesh Munavalli. *An Approach to Mathematical Search Through Query Formulation and Data Normalization*. Springer Berlin Heidelberg, 2007.

[17] Christopher D. Manning, Prabhakar Paghavan, and Hinrich Schutze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[18] Muhammad Adeel, Hui Siu Cheung, and Ar Hayat Khiyal. Math go! prototype of a content based mathematical formula search engine, 2008.

[19] Andrea Asperti, Ferruccio Guidi, Claudio Sacerdoti Coen, Enrico Tassi, and Stefano Zacchiroli. A content based mathematical search engine: whelp. In *In: Post-proceedings of the Types 2004 International Conference, Vol. 3839 of LNCS*, pages 17–32. Springer-Verlag, 2004.

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

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

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

[23] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (3rd Edition)*. Prentice Hall, December 2009.

[24] Peter Graf. *Term Indexing*. Springer Verlag, 1996.

[25] 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.

[26] 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.

[27] Hiroshi Ichikawa, Taiichi Hashimoto, Takenobu Tokunaga, and Hozumi Tanaka. New methods of retrieve sentences based on syntactic similarity. *IPSJ SIG Technical Reports, DBS-136, FI-79*, pages 39–46, 2005.

[28] 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.

[29] 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.

[30] 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.

[31] Shahab Kamali and Frank Wm. Tompa. A new mathematics retrieval system. *CIKM*, 2010.

[32] Kai Ma, Siu Cheung Hui, and Kuiyu Chang. Feature extraction and clustering-based retrieval for mathematical formulas. In *Software Engineering and Data Mining (SEDM), 2010 2nd International Conference on*, pages 372–377, June 2010.

[33] 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.

[34] 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.

[35] Richard Zanibbi and Bo Yuan. Keyword and image-based retrieval for mathematical expressions. *Multi-disciplinary Trends in Artificial Intelligence. 6th International Workshop, MIWAI 2012.*, pages 23–34, 2011.

[36] 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.

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

[38] Akiko Aizawa, Michael Kohlhase, Iadh Ounis, and Moritz Schubotz. Ntcir-11 math-2 task overview. *Proc. of the 11th NTCIR Conference, Tokyo, Japan*, 2014.

[39] Kamali Shahab and Tompa Frank Wm. Improving mathematics retrieval. *Towards a Digital Mathematics Library. Grand Bend, Ontario, Canada*, pages 37–48, 2009.

[40] 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.

[41] J. Roger Hindley. *Introduction to Combinators and [Lambda]-Calculus*. Cambridge University Press, 1986.

Appendix A

TITLE OF APPENDIX A

This is the information for the first appendix, Appendix A. Copy the base file, appA.tex, for each additional appendix needed such as appB.tex, appC.tex, etc. Modify the main base file to include each additional appendix file.

If there is only one appendix, then modify the main file to only use app.tex instead of appA.tex.

Appendix B

TITLE OF APPENDIX B