Expert Systems with Applications 39 (2012) 5820–5828

Contents lists available at SciVerse ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

A lattice-based approach for mathematical search using Formal Concept Analysis

Tam T. Nguyen *, Siu Cheung Hui, Kuiyu Chang

School of Computer Engineering, Nanyang Technological University, 50 Nanyang Avenue, Singapore 639798, Singapore

ARTICLE INFO

Keywords:
Formal Concept Analysis
Formal context
Math search

ABSTRACT

Mathematical (or math) search is a challenging problem as math expressions are highly symbolic and structured. The vast majority of math search systems that adopt conventional text retrieval techniques are ineffective in searching math expressions. In this paper, we propose a lattice-based approach for math search. The proposed approach is based on Formal Concept Analysis (FCA), which is a powerful data analysis technique. In the proposed approach, math expressions are first converted into the corresponding MathML representation, from which math features are extracted. Next, the extracted features are used to construct a mathematical concept lattice. At the query time, the query expression is processed and inserted into the mathematical concept lattice, and the relevant expressions are retrieved and ranked. Finally, search results can be visualized and nevigated via a dynamic graph, thanks to the lattice structure. The proposed lattice-based math search approach is benchmarked against a conventional best match retrieval technique and results show it to be almost 10% better in terms of F1 for the top 30 retrieved results.

1. Introduction

The World Wide Web has evolved very rapidly in recent years. There is a large number of web sites on digital libraries, blogs and forums (Miller & Youssef, 2003), which provide very rich contents for different users. To help users find appropriate information or documents from the Web, search engines such as Google, Yahoo! Search, and Microsoft Bing have been developed. These search engines can search for documents based on various kinds of media data including text, images, audio, and video. However, domain specific documents may contain certain special forms of data such as math expressions (or formulas), drawings, charts, tables, diagrams, etc., that cannot be effectively indexed and queried by classical search engines. For example, math documents contain math expressions, which serve as their primary signature. As such, the current text-based search engines cannot provide adequate support for searching math documents (Youssef, 2006).

Math expressions are highly symbolic and structured. Searching math expressions is a challenging problem. For example, consider the following two math expressions: $(x+1)$ and $e^{(x+1)}$. In each expression, the role of the sub-expression $(x+1)$ is totally different. Therefore, if a query is to find expressions with the exponent $(x+1)$, then all relevant expressions with $(x+1)$ as its exponent should be ranked higher in the search result. In addition, math expressions also have semantic meanings. For example, the two expressions x^2 and y^2 should have the same semantic meaning with a power of 2, though the notations used are different. In another example, consider the expressions $\int \sin(\log(x))dx$ and $\int \log(\sin(x))dx$. The order of the elements in the math expressions also indicates a certain semantic meaning. Any change in the order will also change the semantic meaning of the expression. When a query is issued for math search, it is important to consider the exact order of the elements in the query expression in order to rank the retrieved math expressions correctly.

To support math document retrieval, math-aware search systems such as MathFind (Munavalli & Miner, 2006) and ActiveMath (Libbrecht & Melis, 2006) have been developed. These search systems support users to search for documents based not only on textual data but also on math expressions. Most of the current math search systems have adopted the conventional text retrieval technique in which math expressions are converted into text features and indexed accordingly for subsequent retrieval. However, such text retrieval technique is ineffective for supporting math search. Therefore, in this research, we propose a lattice-based approach for math search based on Formal Concept Analysis (FCA). FCA is a powerful data analysis technique which has been used for information retrieval (Carpineto & Romano, 2000; Cheung & Vogel, 2005; Bedel, Ferr, Ridoux, & Quesseveur, 2008; Messai, Devignes, Napoli, & Smail-Tabbone, 2006; Rajapakse & Denham, 2006; Tho, Hui, & Fong, 2007) in recent years. The lattice-based retrieval approach considers inter-document similarity and supports context-dependent ranking. The advantages of the lattice-based retrieval technique over the conventional text retrieval technique have been discussed in Carpineto and Romano (2000).

In this paper, we present the proposed lattice-based approach for math search and its performance evaluation. The rest of the

* Corresponding author.
E-mail address: nguy0080@e.ntu.edu.sg (T.T. Nguyen).

0957-4174/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2011.11.085

paper is organized as follows. Section 2 reviews the related work. Section 3 discusses the proposed lattice-based approach for math search. Section 4 presents the performance evaluation of the proposed approach and its comparison with the best match retrieval technique. Finally, Section 5 concludes the paper and discusses the directions for future work.

2. Related work

Currently, there are several math-aware retrieval systems such as MathFind (Munavalli & Miner, 2006), ActiveMath (Libbrecht & Melis, 2006), Wolfram Formula Search (Wolfram Formula Search, n.d.), Wikipedia Formula Search (Wikipedia Formula Search, n.d.), Whelp (Andrea, Ferruccio, Sacerdoti, Enrico, & Stefano, 2004), Mathdex (Miner & Munavalli, 2007) and MathWebSearch (Kohlhase & Kohlhase, 2007; Kohlhase & Sucan, 2006) which have been proposed for math search. Most of these systems are developed based mainly on the conventional text retrieval technique, in which math expressions are treated as textual data. To support math search, there are mainly three steps (Youssef, 2005). In the first step, math symbols of a math expression are converted into textual data. For example, the math symbol '+' is mapped to the word "plus", '−' is mapped to the word "minus", etc. The next step is to serialize the structure of the math expression into a linear sequence of textual data. For example, the math expression $\int_1^2 xdx$ can be serialized into a linear sequence of "integral", "lower limit", "upper limit", etc. The third step is to perform normalization in which the math expression is normalized to a canonical form by term sorting. For example, "a plus c plus b" is re-ordered as "a plus b plus c". The output of the algorithm is a sequence of textual data which is then indexed and stored by the conventional text retrieval technique. The advantage of this approach is that it reuses the existing text retrieval technique so that it does not need to develop a new retrieval algorithm. However, with such simple text retrieval technique, although the system is able to achieve very high recall, the precision is not good (Youssef, 2005).

As most text retrieval based math search systems are quite similar, we only review some of the current math search systems in this section. In the MathFind system (Munavalli & Miner, 2006), Munavalli and Miner encoded math expressions into a sequence of textual data called math fragment. In this approach, math expressions are preprocessed and converted into the corresponding MathML format (Ausbrooks et al., 2000). The MathML data are then normalized to remove any ambiguity in its presentation. To index math expressions and support sub-expression search, Miner and Munavalli (2007) defined mathematical *n*-grams in which each mathematical token is considered as one 1-gram. For instance, "$a + b$" has three 1-grams, i.e., '*a*', '*b*' and '+'. To search for math expressions, a query is decomposed into *n*-grams. The indexes are then searched for each *n*-gram in the query. Similarly, Libbrecht and Melis (2006) proposed the ActiveMath Search Tool, which was built on top of the Apache Lucene Search (Jakarta, 2004), a full-featured text search engine library. In ActiveMath Search Tool, the math data are represented in the OMDoc (Kohlhase & Sucan, 2006) format. For retrieval purpose, the math data are converted into tokens and indexed using the Lucene text search engine.

Wolfram Formula Search (n.d.) is another math search system which was developed by Wolfram Research. In Wolfram Formula Search, users are allowed to search for specified formulas from its formula compendium of mathematical functions through the Web. Its database contains more than 300,000 formulas classified into 14 categories. The formula search engine provides a means for users to explore such vast collection of formulas. However, although the Wolfram Formula Search engine provides semantic search, its supports are quite limited since it only supports search queries based on predefined constants, operations and function names. For example, it supports users search for constants (e.g. e, π, etc.), or operations (e.g. sum, product, etc.) appearing on the left, right or anywhere in the formulas.

3. Mathematical representation

Mathematical markup languages play an important role in the development of math search systems. It affects the storage and techniques for the processing of math expressions. There are several common mathematical markup languages: LATEX (Waud, 2003), ASCIIMath (Jipsen, 2007), OMDoc (Kohlhase & Sucan, 2006), OpenMath (Buswell et al., 2004) and MathML (Ausbrooks et al., 2000). The LATEX markup language has been commonly used by many researchers, especially mathematicians, for document processing. It is content-oriented. However, it still has many ambiguities (Altamimi & Youssef, SPS Year; Miller & Youssef, 2003). Therefore, the LATEX markup language is generally not preferred to be used for processing math expressions. ASCIIMath LATEX markup is a simpler form of the LATEX markup language (Youssef, 2006), often used as a format for specifying math expression queries for math-aware search systems. Math expressions represented using ASCIIMath markup are easy to store as it mainly uses ASCII characters to represent the math expressions. However, the disadvantage of ASCIIMath is that its data representation is not well-structured, thereby taking time to parse and extract math expressions for processing. OMDoc and OpenMath are two other standards for semantic-oriented representation of mathematical documents or expressions.

MathML markup developed by W3C is currently used by many systems for the presentation and storage of math expressions in math search engines (Libbrecht & Melis, 2006; Miller & Youssef, 2008; Munavalli & Miner, 2006). There are two kinds of MathML markup languages: MathML content markup and MathML presentation markup. Between the two markup representations, MathML content markup is more commonly used for the processing of math expressions as it contains richer semantic content than the MathML presentation markup.

In this research, MathML content markup is used for encoding math expressions for the proposed lattice-based approach. MathML content markup uses XML for encoding math expressions. The elements in MathML content markup can be divided into groups such as constants & symbols, containers, operators & functions, qualifiers, relations, conditions and semantic mappings. Listing 1 shows an example math expression $(a + b)^2$ encoded in MathML content markup. In Listing 1, the *ci*, *cn*, *plus* and *power* tags stand for the variables *a* and *b*, the constant 2, and the plus and power operators respectively.

4. Lattice-based math search

In document retrieval, the relationship between documents and terms can be represented by term-document matrix. Each entry of the matrix specifies whether a term has occurred in a document or not. To transform a term-document matrix to a formal context of a lattice, Cheung and Vogel (2005) mapped each document to an object and each term to an attribute. Messai et al. (2006) proposed a lattice-based information retrieval system called BR-Explore which constructs a lattice from a term-document matrix. For query retrieval, a query is decomposed into a set of terms or attributes and then the query is inserted into the lattice. If a document shares the same attributes with the query, then it is relevant to the query and retrieved. Rajapakse and Denham (2006) developed another lattice-based document retrieval system which constructs a document

lattice from unit-concepts extracted from documents. A unit-concept is an object-attribute pair which represents the occurrence of a term in a document. The nodes of a query lattice are then matched with those in the document lattice for query retrieval.

In this section, we propose a lattice-based approach for math search using Formal Concept Analysis. As shown in Fig. 1, the proposed approach consists of the following four major steps:

- *Feature Extraction.* This step preprocesses and parses math expressions, and extracts the corresponding math features.
- *Mathematical Lattice Construction.* This step constructs a math lattice from the extracted math features.
- *Query Retrieval.* This step accepts a user query, inserts the query concept into the math concept lattice, matches the query concept with math expression concepts in the concept lattice and ranks the relevant math expressions.
- *Result Visualization.* This step supports users to visualize and navigate the search results in a dynamic graph.

4.1. Feature extraction

Conventional text retrieval systems use terms extracted from documents to index a document collection. However, this method cannot be applied directly to math expressions. For example, the math expression $(x+1)^2$ can be tokenized as *x*, *power*, 2, + and 1. But these tokens cannot be used to index the expression because the structural and semantic meanings of the expression are not fully preserved. Therefore, we propose another method to extract features that capture both semantic and structural meanings of math expressions.

In this research, math expressions are stored and represented in the ASCIIMath format. To extract features, we preprocess and convert the math expressions from the ASCIIMath representation into MathML content markup. Here, we define the following features which will be extracted from math expressions.

Definition 1 (*Mathematical Applying Set*). MathML content markup provides a number of *constructors* for combining other elements into compound objects. *Mathematical Applying Set* M_a is defined as a set of all constructors in MathML content markup such that $M_a = \{interval, list, matrixrow, set, matrix, vector, apply, lambda, piecewise, piece, mfence, otherwise, \ldots\}$.

For example, the math expression $(a+b)^2$ shown in Listing 1 has a subset of the mathematical applying set $\{mfence, apply\} \subset M_a$. The mathematical applying set contains the constructor tags which can be used as indicators when generating features during the feature extraction process.

Definition 2 (*Mathematical Symbol Set*). Let E be a math expression and T be its corresponding MathML content markup, *Mathematical Symbol Set* $M_s(E)$ is defined as a set of labels for elements

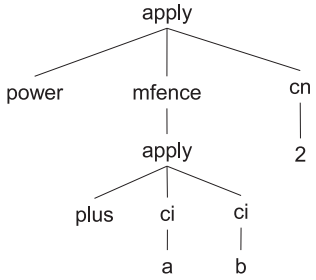

Fig. 2. MathML tree of the math expression $(a+b)^2$.

$n \in T$ where n is a MathML content element such as constant, symbol, container, operator, function, qualifier, relation, condition and semantic mapping.

For example, the math expression $(a+b)^2$ shown in Listing 1 has the mathematical symbol set $M_s((a+b)^2) = \{power, plus, ci, cn\}$. This kind of feature set plays a role in measuring the similarity between two expressions. For instance, the two expressions x^2 and a^2 are similar because they share the same features on *ci*, *cn* and *power*.

Definition 3 (*Mathematical Sub-expression Set*). A constructor applies a function or operator to its arguments. *Mathematical Sub-expression Set* $M_e(E)$ of a math expression E is defined as a set of encoded text which is generated by sorting the arguments under constructors, concatenating the arguments together, and appending them to an operator or function in the expression E. There are three ways to generate the encoded text:

- From constructing mathematical objects. For example, a scalar variable ⟨ci⟩x⟨/ci⟩ is converted into *ci_x*. A constant ⟨cn⟩123⟨/cn⟩ is converted into *cn*_123.
- From applying a function or operator to its arguments. The arguments can be constants, variables, compound objects, or results of another operator or function. For example, by applying the *plus* operator to its arguments ⟨ci⟩x⟨/ci⟩ and ⟨ci⟩y⟨/ci⟩, the encoded text is *plus_ci_x_ci_y*.
- From combining elements into a compound object under a constructor. The elements such as functions, operators and arguments are direct children of the constructor node.

For example, the math expression $(a+b)^2$ shown in Listing 1 has the mathematical sub-expression set $M_e((a+b)^2) = \{ci\_a, ci\_b, cn\_2, plus\_ci\_a\_ci\_b, power\_plus\_cn\_2\}$, where *ci_a* is composed from *ci* and *a*, and so on. The mathematical sub-expression set is the most important set and plays a crucial role in the retrieval and ranking process. For example, if we have two math expressions x^2 and a^2, and a query x^2, then the mathematical sub-expression set can be used to determine the rank of the math expression x^2 which should be higher than that of the math expression a^2.

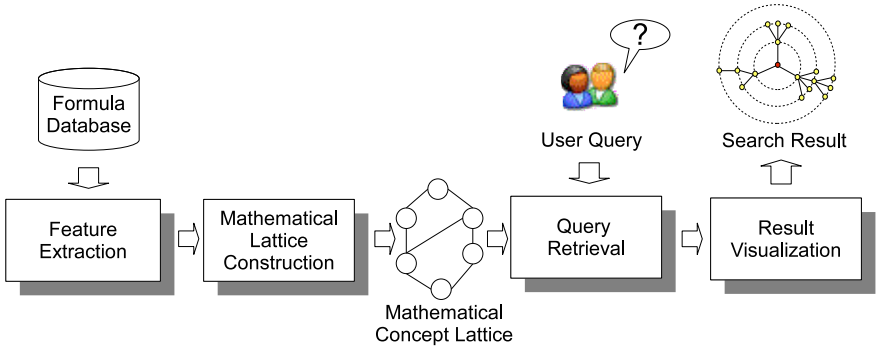

Fig. 1. Lattice-based math search.

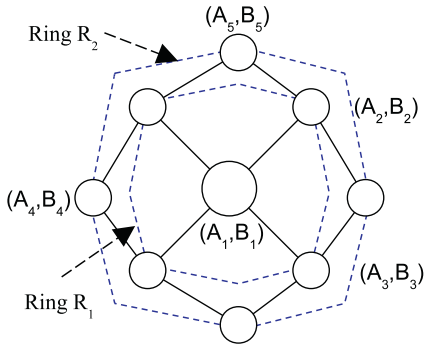

Fig. 3. An example of concept ring.

Algorithm 1 shows the feature extraction approach to generate features from the MathML content markup. Note that the *text node* in Line 3 is text data, not MathML tags. For example, in the MathML tree of the math expression $(a+b)^2$ shown in Fig. 2, the three nodes *a*, *b* and 2 are text nodes. In this algorithm, we use some new functions. Firstly, the `container_name (node)` function will return the name of the node. For example, if the current node is ⟨ci⟩, the returned value will be *ci*. Secondly, the `get_operator` and `get_operands` functions will return the operator and operands of the current node. The returned value of the `get_operands` function may have one or more operands depending on the operator. For example, if the operator is *sqrt* (square root), the returned value will have one operand. If the operator is *power*, the returned value will have two operands. And if the operator is *plus*, the returned value will have two or more operands. Finally, the `generate_feature` function will return the encoded text of the current node based on its child nodes. This function will concatenate the operator and operands to generate the encoded text. For example, if the operator node is *plus* and operands are *ci_a* and *ci_b*, the returned value will be *plus_ci_a_ci_b*.

Algorithm 1. `Feature_Extraction`

Input:
MathML – A collection of math expressions in *MathML* content markup format
Output:
M_s – A mathematical symbol set
M_e – A mathematical sub-expression set
Process:
1: Initialize $M_s \leftarrow \emptyset$ and $M_e \leftarrow \emptyset$
2: **for all** *node* ∈ *MathML* **do**
3: **if** *node* is not a text node **then**
4: **if** $node \notin M_a$ **then**
5: Append *container_name*(*node*) into M_s
6: **else**
7: $operator \leftarrow get\_operator(node)$
8: $operands \leftarrow get\_operands(node)$
9: $f \leftarrow generate\_feature(operator, operands)$
10: $M_e \leftarrow M_e \cup \{f\}$
11: **end if**
12: **else**
13: $p \leftarrow node.parent$
14: $f \leftarrow generate\_feature(p, node)$
15: $M_e \leftarrow M_e \cup \{f\}$
16: **end if**
17: **end for**
18: **return** M_s **and** M_e

Based on the math expression $(a+b)^2$ with its MathML tree in Fig. 2, the Feature_Extraction algorithm will generate the mathematical symbol set $M_s((a+b)^2) = \{ci, ci, plus, power\}$ and the mathematical sub-expression set $M_e((a+b)^2) = \{ci\_a, ci\_b, cn\_2, plus\_ci\_a\_ci\_b, power\_plus\_cn\_2\}$.

4.2. Mathematical lattice construction

This section presents the proposed *Mathematical Formal Concept Analysis* for mathematical lattice construction.

Definition 4 (*Mathematical Formal Context*). A *Mathematical Formal Context* is $K = (G, M_c, M_i, I)$, where G is a set of mathematical formulas, M_c is a set of mathematical container attributes, M_i is a set of mathematical instance attributes, and $I \subseteq G \times (M_c \cup M_i)$ is a binary relation between G and $(M_c \cup M_i)$ to represent the relation between formulas $g \in G$ and attributes $m \in M_c \cup M_i$.

Definition 5 (*Mathematical Formal Concept*). Given a mathematical formal context $K = (G, M_c, M_i, I)$, we define $A' = \{m \in M_c \cup M_i | \forall g \in A : (g,m) \in I\}$ for $A \subseteq G$ and $B' = \{g \in G | \forall m \in B : (g,m) \in I\}$ for $B \subseteq M_c \cup M_i$. A *Mathematical Formal Concept* of a mathematical formal context (G, M_c, M_i, I) is a pair (A,B) where $A \subseteq G$, $B \subseteq M_c \cup M_i$, $A' = B$, and $B' = A$. A and B are the extent and intent of the mathematical formal concept (A,B) respectively. A set of all concepts of formal context (G, M_c, M_i, I) is $C(G, M_c, M_i, I)$.

Definition 6 (*Superconcept and Subconcept*). For a given set of all concepts $C(G, M_c, M_i, I)$ of the context $K = (G, M_c, M_i, I)$. Suppose (A_1, B_1) and (A_2, B_2) are concepts in $C(G, M_c, M_i, I)$. We say that (A_1, B_1) is the *superconcept* of (A_2, B_2) if $A_1 \supseteq A_2$, which is equivalent to $B_1 \subseteq B_2$. Likewise, (A_2, B_2) is called the *subconcept* of (A_1, B_1), denoted as $(A_2, B_2) \leqslant (A_1, B_1)$.

Definition 7 (*Parent and Child Concept*). Given a mathematical formal concept (A_1, B_1) and all of its superconcepts S, a mathematical formal concept $(A_2, B_2) \in S$ is a *parent concept* of the mathematical formal concept (A_1, B_1) if (A_2, B_2) is not a superconcept of one of the mathematical formal concepts $c \in S \backslash \{(A_2, B_2)\}$ or (A_2, B_2) is the smallest element in the superconcept set of (A_1, B_1). Equivalently, a *child concept*(A_3, B_3) of the mathematical formal concept (A_1, B_1) is the greatest element in its subconcept set. A set of parent and child concepts is also called the *nearest neighbors* of (A_1, B_1).

Definition 8 (*Concept Ring*). Let $C = \{(A_1, B_1), (A_2, B_2), \ldots, (A_n, B_n)\}$ be a set of formal concepts. Consider the ordered set $(C; \succ\prec)$ where $\succ\prec$ is the nearest neighbor relation. The shortest path between two concepts (A_i, B_i) and (A_j, B_j) is the smallest integer number $m \in N$ for which the following condition holds: $\exists (A_0, B_0), (A_1, B_1), \ldots, (A_m, B_m) \in C$ such that $(A_i, B_i) = (A_0, B_0) \succ\prec (A_1, B_1) \cdots \succ\prec (A_m, B_m) = (A_j, B_j)$. The *concept ring* $R_k(A,B)$ of a formal concept (A,B) comprises all the formal concepts whose shortest path to (A,B) equals to k.

For example, Fig. 3 shows the first and second concept rings of the formal concept (A_1, B_1). The first concept ring $R_1(A_1, B_1) = \{(A_2, B_2), (A_3, B_3), \ldots\}$ is a set of nearest neighbors of the formal concept (A_1, B_1), where (A_2, B_2) and (A_3, B_3) are its upper and lower neighbors respectively. And $R_2(A_1, B_1) = \{(A_4, B_4), (A_5, B_5), \ldots\}$ is the second concept ring of the formal concept (A_1, B_1).

Definition 9 (*Mathematical Concept Lattice*). A *Mathematical Concept Lattice* of a formal context $K = (G, M_c, M_i, I)$ is a set $C(G, M_c, M_i, I)$ of all mathematical formal concepts of K with the partial order $(\leqslant)$.

Table 1
Sample data for constructing lattice.

Math expression	Features
$E_1 : x$	*ci* (1), *ci_x* (2)
$E_2 : y$	*ci*, *ci_y* (3)
$E_3 : z$	*ci*, *ci_z* (4)
$E_4 : t$	*ci*, *ci_t* (5)
$E_5 : x + y$	*ci*, *plus* (6), *ci_x*, *ci_y*, *plus_ci_x_ci_y* (7)
$E_6 : y + t$	*ci*, *plus*, *ci_y*, *ci_t*, *plus_ci_t_ci_y* (8)
$E_7 : x + z$	*ci*, *plus*, *ci_x*, *ci_z*, *plus_ci_x_ci_z* (9)
$E_8 : x + y + z$	*ci*, *plus*, *ci_x*, *ci_y*, *ci_z*, *plus_ci_x_ci_y*, *plus_ci_x_ci_z*, *plus_ci_y_ci_z* (10)
$E_9 : x + y + t$	*ci*, *plus*, *ci_x*, *ci_y*, *ci_t*, *plus_ci_x_ci_y*, *plus_ci_t_ci_x* (11), *plus_ci_y_ci_t*

To illustrate how to construct a mathematical concept lattice, consider the math expressions given in Table 1. The numbers enclosed in brackets in the feature list represent the labels of the features. For example, the feature *ci_x* is labeled as (1). After feature extraction, we generate a formal context and construct a math concept lattice using the Ganter's `Next_Closure` algorithm (Ganter & Wille, 1997) which is known to be a fast algorithm. We know that there are $2^{|G|}$ subsets of G (i.e. the set of math expressions). Instead of finding the set of concepts from the whole set of subsets of G, the `Next_Closure` algorithm applies a special strategy to find the set of concepts by using closure based on the order of the subsets of G (Carpineto & Romano, 2003). In this way, the next subset can be generated from the current subset. So only some subsets of G are considered in the concepts finding process. Fig. 4 shows the mathematical concept lattice constructed based on the mathematical formal context given in Table 2 using the Ganter's `Next_Closure` algorithm.

4.3. Query retrieval

When a query (say Q) is submitted, it is then converted from the ASCIIMath format into the corresponding MathML content markup representation and its math features are extracted. The query concept is then generated and inserted into the concept lattice. The math expressions that share the same concept with Q are the most relevant to it. Other less relevant math expressions could be found from the *extent set* of the nearest neighbors of the query concept. In order to do that, the nearest neighbors are needed. However, the formal concepts in the concept lattice using the `Next_Closure` algorithm do not give the nearest neighbors. To identify the nearest neighbors of a concept, we need to find its upper and lower neighbors. In fact, we only need to find the upper neighbors of all concepts, as the lower neighbors of the concepts can be identified from the corresponding parent–child relationship. For example, if the concept C_1 is the upper neighbor of the concept C_2, then the concept C_2 must be the lower neighbor of the concept C_1. As such, we only need to find the upper neighbors of all concepts in the concept lattice. To achieve this, we have developed the `Get_Upper_Neighbors` algorithm as shown in Algorithm 2, which has the following three steps: (i) assign each concept in the lattice an identifier number, called concept ID; (ii) iterate over the concepts to get all their superconcepts; and (iii) for each concept in the superconcept set, remove the concept which is the superconcept of another concept in this set.

Algorithm 2. `Get_Upper_Neighbors`

Input:
Lattice – A mathematical lattice created by the `Next_Closure` algorithm
Output:
Upper – A set of upper concepts of each concept
Process:
1: *Concepts* ← all concepts of *Lattice*
2: Assign a unique concept ID for each concept $c \in$ *Concepts*
3: **for all** $c \in$ *Concepts* **do**
4: $P \leftarrow$ superconcept of c
5: **for all** $p \in P$ **do**
6: **if** p is a superconcept of any concept in $P \backslash \{p\}$ **then**
7: $P \leftarrow P \backslash \{p\}$
8: **end if**
9: **end for**
10: *Upper*(c) ← P
11: **end for**
12: **return** *Upper*

To speed up the query retrieval process, instead of creating a new concept lattice, we use an incremental algorithm by inserting the query concept into the math concept lattice. The `Update_Local_Lattice_Structure` algorithm (Carpineto & Romano, 2003) is adopted for this purpose. The implementation of the algorithm can be found in Carpineto and Romano (2003).

For example, given the concept lattice in Fig. 4 and the query $Q : x + y$. Let $M_s(Q) = \{ci, plus\}$ and $M_e(Q) = \{ci\_x, ci\_y, plus\_ci\_x\_ci\_y\}$ be the mathematical symbol set and mathematical sub-expression

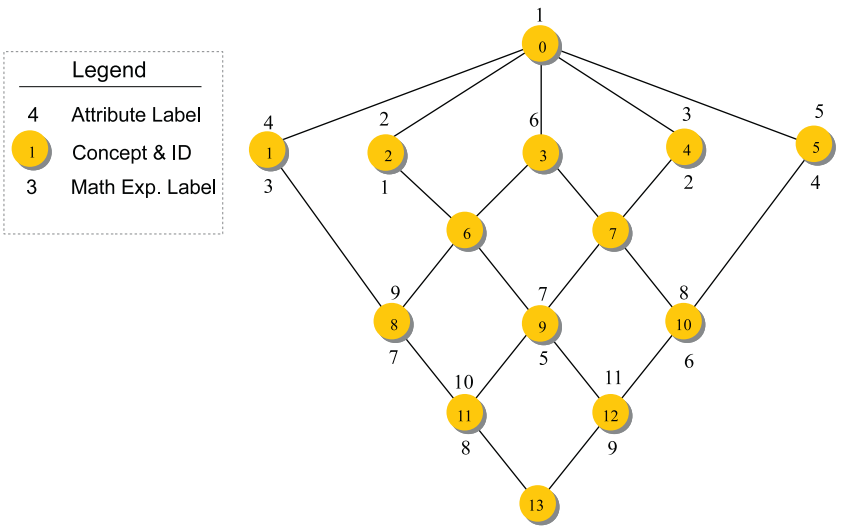

Fig. 4. Concept lattice of the math expressions in Table 1.

Table 2
Mathematical formal context.

Math expression	1	2	3	4	5	6	7	8	9	10	11
1	×	×									
2	×		×								
3	×			×							
4	×				×						
5	×	×	×			×	×				
6	×		×		×	×		×			
7	×	×		×		×			×		
8	×	×	×	×		×	×		×	×	
9	×	×	×		×	×	×	×			×

set of the query respectively. From Table 1, we have the feature set of Q as $\{1,2,3,6,7\}$ which exactly matches the intent set of concept 9 in Fig. 4. Therefore, there is no new concept created and the query will be inserted into concept 9. Fig. 5 shows the concept lattice with the query. In the figure, the number on the right of each concept indicates the distance from that concept to the query. The extent set of concept 9 contains the query. The distances from the query to other concepts are shown next to the concepts. In this case, the distances from concept 9 to concepts 6, 7, 8, 10, 11 and 12 are 1, 1, 2, 2, 1 and 1 respectively. Since the query is inserted into concept 9 which contains expression E_5, the expression E_5 has the highest rank. The expressions E_8 and E_9 are ranked second because they are in the nearest neighbor concepts of concept 9. And similarly, it follows by the expressions E_6, E_7, E_1 and E_2.

For math expressions belonging to one concept, the lattice-based ranking cannot discriminate these expressions because they have the same distance to the query. In order to rank them, the best match retrieval technique should be used. To apply the best match retrieval technique, we use the Jaccard coefficient to calculate the similarity between two expressions based on their attributes.

Definition 10 (*Concept Distance*). Given a *concept ring* $R_d(C)$ of a mathematical formal concept C, the *concept distance* from the concept $C_i \in R_d(C)$ to the concept C is defined as $\mathrm{Dist}(C,C_i) = d$, where d is the shortest distance from C to C_i.

It is trivial to see that the concept distance, i.e., the shortest path between the two concepts, satisfies the following conditions:

(i) (Positivity) $\mathrm{Dist}(C_i,C_j) \geqslant 0$, $\forall i,j \in [1,n]$, where n is the number of concepts in the concept lattice. And $\mathrm{Dist}(C_i,C_j) = 0$ only if $C_i = C_j$.
(ii) (Symmetry) $\mathrm{Dist}(C_i,C_j) = \mathrm{Dist}(C_j,C_i)$, $\forall i,j \in [1,n]$.
(iii) (Triangle Inequality) $\mathrm{Dist}(C_i,C_j) \leqslant \mathrm{Dist}(C_i,C_k) + \mathrm{Dist}(C_k,C_j)$, $\forall i,j,k \in [1,n]$.

The first condition (i) is always true because the shortest path between the two concepts is a non-negative integer. It equals to zero if the two concepts are the same. The second condition (ii) is also true because the shortest path from concept C_i to C_j is the same as that from C_j to C_i. Finally, the third condition (iii) is true because $\mathrm{Dist}(C_i,C_j)$ is the shortest path between C_i and C_j, so there is no other paths shorter than this one. Therefore, the concept distance is a metric.

Definition 11 (*Similarity*). Given two expressions E_1 and E_2, let $\mathrm{M}(E_1)$ and $\mathrm{M}(E_2)$ be their attribute sets. We define the similarity measure between E_1 and E_2 as $\mathrm{Sim}(E_1,E_2) = \frac{\|\mathrm{M}(E_1)\cap\mathrm{M}(E_2)\|}{\|\mathrm{M}(E_1)\cup\mathrm{M}(E_2)\|}$.

From the above definition, $\mathrm{Sim}(E_1,E_2)$ is always less than or equal to 1. Its value is equal to 1 if the two expressions are the same. For example, given the two expressions $E_5 : x + y$ and $E_6 : y + t$, we have the attribute sets:

$$\mathrm{M}(E_5) = \{ci, plus, ci\_x, ci\_y, plus\_ci\_x\_ci\_y\}$$
$$\mathrm{M}(E_6) = \{ci, plus, ci\_y, ci\_t, plus\_ci\_t\_ci\_y\}$$

The similarity can be computed as follows:

$$\mathrm{Sim}(E_5,E_6) = \frac{\|\{ci,plus,ci\_y\}\|}{\|\{ci,plus,ci\_x,ci\_y,ci\_t,plus\_ci\_x\_ci\_y,plus\_ci\_t\_ci\_y\}\|} = 0.43$$

To rank the search result, we define a dis-similarity measure on the concept lattice. This measurement is a linear combination of the concept distance and the similarity measurement. The search result will be sorted in ascending order by dis-similarity values between the query and the expressions.

Definition 12 (*Concept Dis-similarity*). Given a query Q in the extent set of concept C_q and an expression E_i in the extent set of concept C_j. We define the *concept dis-similarity* measure between the query Q and the expression E_i as $\mathrm{Dissim}(Q,E_i) = \mathrm{Dist}(C_q,C_j) + 1 - \mathrm{Sim}(Q,E_i)$.

For example, in Fig. 5, the dis-similarity measures between the query Q and the two expressions $E_5 : x + y$ and $E_6 : y + t$ are $\mathrm{Dissim}(Q,E_5) = 0$ and $\mathrm{Dissim}(Q,E_6) = 2 + 1 - 0.43 = 2.57$ respectively.

After inserting the query into the lattice, we can then retrieve all the relevant expressions by breadth-first search and rank them accordingly. In FCA, the `Focus_&_Context_View` algorithm (Carpineto & Romano, 2003) is most suitable for this purpose. The `Focus_&_Context_View` algorithm searches for all the neighbors of the current concept from the input concept lattice. In this research, we have modified the `Focus_&_Context_View` algorithm to search for neighbors of more than one concept. In addition, as we do not need to have all relevant expressions to be returned, the parameter called *maxNumObjects* is used to control the number of returned expressions according to the *distance* threshold. The result set will then be sorted according to *distance*. The new algorithm, namely `Get_Ranked_Expression_List`, is presented in Algorithm 3.

Algorithm 3. `Get_Ranked_Expression_List`

Input:
Lattice – A mathematical lattice with inserted query
Query – A query concept
maxNumObjects – A maximum number of objects threshold
maxDistance – A maximum distance threshold
Output:
ResultSet – A set of sorted pairs (concept, distance)
Process:
1: Initialize *currentRing* ← {*Query*}
2: *distance* ← 0
3: *numObjects* ← 0
4: *nextRing* ← ∅
5: **while** *numObjects* < *maxNumObjects* **do**
6: *distance* ← *distance* + 1
7: **for all** $c \in$ *currentRing* **do**
8: $N \leftarrow GetParents(c) \cup GetChilds(c)$
9: **for all** $n \in N$ **do**
10: **if** *numObjects* < *maxNumObjects* **then**
11: *ResultSet* ← *ResultSet* ∪ {(*n*, *distance*)}
12: *numObjects* ← *numObjects* + 1
13: **end if**
14: *nextRing* ← *nextRing* ∪ {*n*}
15: **end for**
16: **end for**
17: *currentRing* ← *nextRing*
18: **end while**
19: Sort *ResultSet* by *distance*
20: **return** *ResultSet*

Table 3 shows the ranked result for the query $x + y$. In this example, the query is inserted into the same concept as the expression $E_5 : x + y$. As their concept distance is zero, the expression E_5 is ranked 1. The

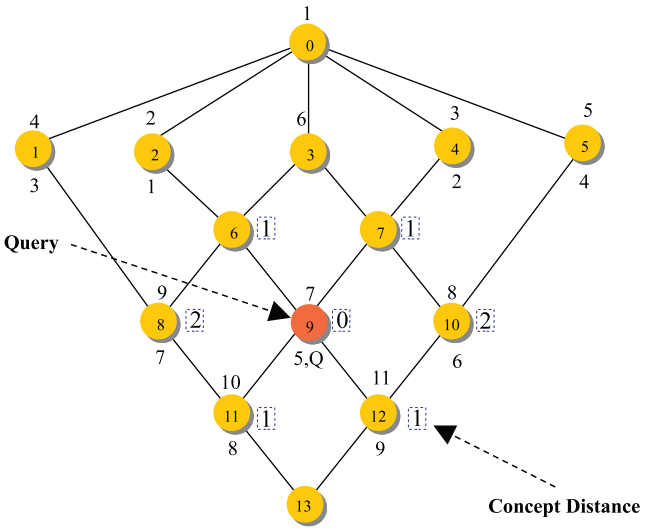

Fig. 5. Concept lattice of the math expressions in Table 1 with an inserted query.

Table 3
Ranked results for the query $x + y$.

Math expression	Concept distance	Dis-similarity	Rank
$E_5 : x + y$	0	0.0	1
$E_8 : x + y + z$	1	1.3	2
$E_9 : x + y + t$	1	1.3	2
$E_6 : y + t$	2	2.57	4
$E_7 : x + z$	2	2.57	4
$E_1 : x$	2	2.6	6
$E_2 : y$	2	2.6	6
$E_3 : z$	3	3.9	8
$E_4 : t$	3	3.9	8

expressions $E_8 : x + y + z$ and $E_9 : x + y + t$ are ranked 2 because they belong to the nearest neighbors of the query concept (their concept distances equal to 1). Similarly, we derive the ranks for other expressions.

4.4. Result visualization

Cole, Eklund, and Stumme (2003) applied FCA as a visualization tool for email discovery. The lattice-based visual file structure for emails was developed to support interactive visual navigation of emails. We extend this idea to math search. In our proposed approach, we support interactive exploration of concept lattice using dynamic graph with radial layout (Yee, Fisher, Dhamija, & Hearst, 2001). To display a concept lattice, we map its formal concepts to the graph nodes. Each node represents a formal concept. The relation between two formal concepts is presented by an edge connecting two corresponding nodes in the graph. The focused node lies at the center of the view. The other nodes that are directly connected to the focused node will be arranged around this node. In other words, the formal concepts belonging to the concept ring of the focused concept will lie on the same circle around the center node.

To display the layout of the graph, we treat the graph as a radial tree with the focused or selected node to be the root node of the tree. Nodes that are directly related to the root node, including child nodes, are drawn on the same circle. These nodes may have other neighbors as child nodes, which will be drawn on another circle and so on. After drawing the tree in this manner, the remaining relations will then be drawn on the graph. The final graph will comprise a center node and concentric rings. Fig. 6 shows the graph for the formal concepts given in Fig. 5 in which the focused or selected concept is the query. This graph shows the initial search result to the user. The user can then navigate the search result by selecting another

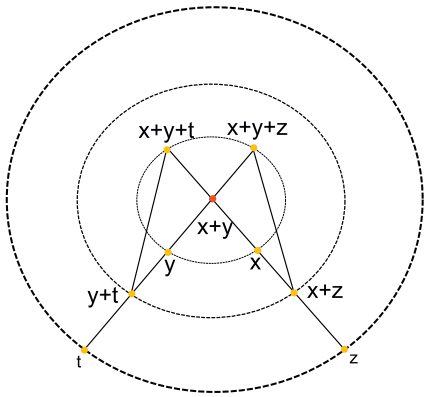

Fig. 6. Graph visualization for the concept lattice in Fig. 5 in which the selected concept is the query $x + y$.

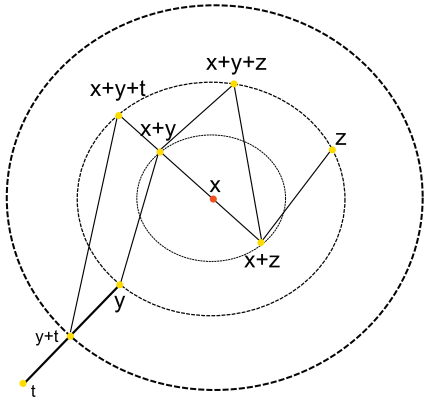

Fig. 7. Graph visualization for the concept lattice in Fig. 5 when node x is selected.

node in the graph. For example, if the user selects node x, the graph will animate to display a new layout as shown in Fig. 7. To explore further, the user can select any other node in the graph, which will then become the focused node and the new display layout will be determined by calculating the distance from the new focused node to other nodes using breadth-first search (Yee et al., 2001).

5. Performance evaluation

In this section, we present the performance of the proposed approach for math search. In addition, we also compare the performance of the proposed approach with the best match retrieval technique. The best match retrieval technique is chosen for comparison as it outperforms the clustering based retrieval technique according to the experimental results given in Carpineto and Romano (2000).

As there is no benchmark data available for performance evaluation in the research community (Youssef, 2005), we conduct the evaluation by collecting a dataset of 489 integral and limit expressions categorized based on different functions such as polynomial, rational, irrational, trigonometrical, exponential and logarithm (Thomas & Weir, 2004). Table 4 shows some sample data of the dataset. To conduct the test, 20 test query expressions are created. The test queries are also collected according to the different math expression function categories. Four test queries from each of the function categories are used to form the 20 test queries. Table 5 shows some sample data of the test queries.

The test queries aim to test different math search properties such as sub-expression search, similar expression search and near-similar expression search which are briefly discussed as follows:

Table 4
Some sample integral expressions.

Math expression	Function category
$\int 4x^3 dx$	Integral with polynomial function
$\int_1^2 (1+x^2)^3 dx$	Integral with polynomial function
$\int_0^1 \frac{2x^2+3}{(x^2+1)^2} dx$	Integral with rational function
$\int \frac{5x^2+20x+6}{x^3+2x^2+x} dx$	Integral with rational function
$\int x^3\sqrt{9-x^2} dx$	Integral with irrational function
$\int \frac{x^2}{\sqrt{x+1}+2} dx$	Integral with irrational function
$\int_0^{\frac{\pi}{2}} \sin x \cos^3 x dx$	Integral with trigonometrical function
$\int_0^{\frac{\pi}{3}} \sin^3 x dx$	Integral with trigonometrical function

- *Sub-expression search.* Sub-expression is a part of an expression. For example, math expression queries such as $x+1$ and $x+2$ can be used to evaluate sub-expression search for expressions such as $(x+1)(x+2)$ from the dataset.
- *Similar expression search.* Similar expression is an expression which has similar meaning to the existing expressions in the dataset. For example, math expression queries such as $(x+2)(x+1)$, $(1+x)(x+2)$ and $(1+x)(2+x)$ can be used to evaluate similar expression search for expressions such as $(x+1)(x+2)$ in the dataset.
- *Near-similar expression search.* Near-similar expression is an expression which has similar meaning to the expressions in the dataset. For example, math expression queries such as $x+1$ and $x+2$ can be used to evaluate near-similar search to expressions such as $x+a$ in the dataset.

Table 6 shows the sample test queries with their relevant expressions in the dataset. The relevant expressions are ordered by their similarities to the queries. Take the query $\sqrt{x+1}$ as an example, the most relevant expression should be $\int(x^2+2x+1)\sqrt{x+1}dx$ as it contains the query, while the third expression $\int_2^5 \frac{x}{x+1} dx$ is least relevant. However, it shares the same sub-expression $x+1$ with the query.

To evaluate the performance of the proposed lattice-based approach, the features of the expressions from the dataset and the test queries are extracted using the `Feature_Extraction` algorithm discussed in Section 4.1. Then, the performance of the proposed lattice-based approach and the best match retrieval technique is measured based on the extracted features. For the conventional best match retrieval technique, it uses the similarity measure given in Definition 11 of Section 4.3 for retrieval and ranking. The performance is then evaluated based on *recall*, *precision* and *F-measure* (F1) which are defined as follows:

$$Recall = R_A/R$$
$$Precision = R_A/A$$
$$F1 = 2 \times Recall \times Precision/(Recall + Precision)$$

Table 5
Some sample test queries.

Query	Function category	Query type
$(x^2+1)^2$	Polynomial function	Sub-expression search
$\sqrt{x+1}$	Irrational function	Sub-expression search
$\frac{x^2}{x+1}$	Rational function	Sub-expression search
$\cos x \sin x$	Trigonometrical function	Similar expression search
x^2+1	Polynomial function	Similar expression search
$3x^2+2x$	Polynomial function	Near similar expression search

Table 6
The relevant expressions for the test queries in Table 5.

Query	Relevant expressions
$(x^2+1)^2$	$\int \frac{x^2+1}{(x+1)^2} dx,\ \int \frac{x^2+4}{(x^2+1)^2(x^2+2)} dx,\ \int_0^1 \frac{1-x^2}{(1+x^2)^2} dx, \ldots$
$\sqrt{x+1}$	$\int(x^2+2x+1)\sqrt{x+1}dx,\ \int \frac{1}{\sqrt{x+1}+\sqrt{x}} dx,\ \int_2^5 \frac{x}{x+1} dx, \ldots$
$\frac{x^2}{x+1}$	$\int \frac{x^2}{x+1} dx,\ \int \frac{x^2}{(x+1)(x-1)^2} dx,\ \int \frac{x^2}{(x+1)(x^2+1)} dx, \ldots$
$\cos x \sin x$	$\int e^{2x} \cos x \sin x dx,\ \int_0^t \cos x \sin x \ln(\cos x) dx,\ \int_0^{\frac{\pi}{2}} \sin x \cos^3 x dx, \ldots$
x^2+1	$\int(1+x^2)dx,\ \int(1+x^2)^3 dx,\ \int(3x^2+2x+1)dx, \ldots$
$3x^2+2x$	$\int(3x^2+2x+1)dx,\ \int(3x^2+2x+4)dx,\ \int_0^2 \lvert x^2+2x-3\rvert dx, \ldots$

Table 7
Performance results.

		Best match (%)	Lattice-based (%)
@Top-10	Precision	86.66	90.00
	Recall	36.38	37.39
	F-measure	51.25	52.83
@Top-20	Precision	79.36	84.04
	Recall	68.17	72.20
	F-measure	73.35	77.67
@Top-30	Precision	64.29	75.00
	Recall	79.72	87.33
	F-measure	71.18	80.69

where A is the number of retrieved expressions, R_A is the number of retrieved relevant expressions, and R is the number of relevant expressions. The *F-measure* takes into account both recall and precision. It is the Harmonic mean of recall and precision.

Table 7 shows the performance results of the proposed lattice-based approach and the best match retrieval technique based on the 20 test queries. With each test query, we retrieve the top ranked expressions. The performance results on *recall*, *precision* and *F-measure* are calculated based on the top 10, 20 and 30 retrieved expressions over all queries. The results show that the proposed lattice-based approach outperformes the best match retrieval technique (52.83% vs. 51.25% F-measure). Notable, our approach gives more than 3% better precision and 1% better recall. For the top-20 and top-30, our approach is 4% and 9% better in terms of

```
<apply>                 % apply operator
   <power/>             % power operator
   <mfence>
      <apply>           % apply operator
         <plus/>        % plus operator
         <ci>a</ci>     % variable (ci) a
         <ci>b</ci>     % variable (ci) b
      </apply>
   </mfence>
   <cn>2</cn>           % constant (cn) 2
</apply>
```

Listing 1. MathML content markup of $(a+b)^2$.

F1, respectively. Precision at top-30 is even more impressive, surpassing the best match approach by more than 10%. The results demonstrate the robustness of our proposed approach over the best match retrieval technique. In particular, using a lattice approach solved the low-precision problem prevalent in many text-based approaches. When we consider the concept ring in the lattice-based approach, we can see that the expressions in the concept ring form a cluster. This cluster contains expressions which have similar semantic meaning to the query. The expressions in different rings will have different ranks. When the "radius" of the concept ring increases, the expressions in that ring are less relevant to the query. This helps remove many irrelevant expressions before applying the best match retrieval technique for ranking the expressions of the same concept ring.

6. Conclusion

In this paper, we have proposed a lattice-based approach for mathematical search based on Formal Concept Analysis. The proposed approach consists of four major steps on feature extraction, mathematical lattice construction, query retrieval and search result visualization. The performance of the proposed approach has been evaluated and encouraging results have been obtained. The performance results have shown that the proposed approach has consistently performed better than the conventional best match retrieval technique. Another important advantage of the proposed lattice-based approach lies in its support for the visualization and navigation of search results via a dynamic graph. Currently, we are extending our research work by integrating math search with text retrieval for mathematical document retrieval and visualization. As such, scientific mathematical documents can be indexed, retrieved and visualized using both textual data and math expressions in a meaningful and effective manner.

Acknowledgement

This research was supported in part by Singapore Ministry of Education's Academic Research Fund Tier 1 Grant RG 30/09.

References

Altamimi, M. E., & Youssef, A. (2007). A more canonical form of content mathml to facilitate math search. In *Extreme markup languages 2007*.

Andrea, A., Ferruccio, G., Sacerdoti, C. C., Enrico, T., & Stefano, Z. (2004). A content based mathematical search engine: Whelp. In *TYPES* (pp. 17–32).

Ausbrooks, R., Buswell, S., Dalmas, S., Devitt, S., Diaz, A., Hunter, R., Smith, B., Soiffer, N., Sutor, R., & Watt, S. (2000). Mathematical markup language (mathml) version 2.0.

Bedel, O., Ferr, S., Ridoux, O., & Quesseveur, E. (2008). GEOLIS: A logical information system for geographical data. *Revue Internationale de Gomatique, 17*(3-4), 371–390.

Buswell, S., Caprotti, O., Carlisle, D. P., Dewar, M. C., Gaetano, M., & Kohlhase, M. (2004). The open math standard version 2.0.

Carpineto, C., & Romano, G. (2000). Order-theoretical ranking. *Journal of the American Society for Information Sciences*, 587–601.

Carpineto, C., & Romano, G. (2003). *Concept Data Analysis: Theory and Applications*.

Cheung, K. S. K., & Vogel, D. (2005). Complexity reduction in lattice-based information retrieval. *Information Retrieval, 8*, 285–299.

Cole, R. J., Eklund, P., & Stumme, G. (2003). Document retrieval for email search and discovery using formal concept analysis. In *Applied artificial intelligence* (pp. 257–280).

Ganter, B., & Wille, R. (1997). *Formal Concept Analysis: Mathematical Foundations*. Springer-Verlag.

Jakarta, A. (2004). A high-performance, full-featured text search engine library. Available at: <http://lucene.apache.org/>.

Jipsen, P. (2007). Translating ascii math notation to mathml and graphics. Available at: <http://www1.chapman.edu/~jipsen/mathml/asciimath.html>.

Kohlhase, A., & Kohlhase, M. (2007). Reexamining the mkm value proposition: From math web search to math web research. In *Calculemus '07/ MKM '07: Proceedings of the 14th symposium on towards mechanized mathematical assistants* (pp. 313–326). Berlin, Heidelberg: Springer-Verlag.

Kohlhase, M., & Sucan, I. (2006). A search engine for mathematical formulae. In J. Calmet, T. Ida, & D. Wang (Eds.), *AISC '06: Proceedings of 8th international conference on artificial intelligence and symbolic computation* (pp. 241–253). Springer-Verlag.

Kohlhase, M., & Sucan, I. A. (2006). A search engine for mathematical formulae. In *Proceedings of artificial intelligence and symbolic computation. LNAI* (Vol. 4120, pp. 241–253). Springer.

Libbrecht, P., & Melis, E. (2006). Methods to access and retrieve mathematical content and activemath. In *ICMS '06: In proceedings of the 2nd international congress on mathematical software*.

Messai, N., Devignes, M.-D., Napoli, A., & Smail-Tabbone, M. (2006). Br-explorer: A sound and complete fca-based retrieval algorithm. In *ICFCA '06: The 4th international conference on formal concept analysis*.

Miller, B.R., & Youssef, A. (2003). Technical aspects of the digital library of mathematical functions. In *Annals of mathematics and artificial intelligence* (Vol. 38, pp. 121–136).

Miller, B. R., & Youssef, A. (2008). Augmenting presentation mathml for search. In *MKM '08: Proceedings of the 7th international conference on mathematical knowledge management* (pp. 536–542).

Miner, R., & Munavalli, R. (2007). An approach to mathematical search through query formulation and data normalization. In *Calculemus '07/ MKM '07: Proceedings of the 14th symposium on towards mechanized mathematical assistants* (pp. 342–355). Berlin, Heidelberg: Springer-Verlag.

Munavalli, R., & Miner, R. (2006). Mathfind: A math-aware search engine. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on research and development in information retrieval*. New York, NY, USA: ACM. 735.

Rajapakse, R. K., & Denham, M. (2006). Text retrieval with more realistic concept matching and reinforcement learning. *Information Processing and Management, 42*(5), 1260–1275.

Tho, Q. T., Hui, S. C., & Fong, A. C. M. (2007). A citation-based document retrieval system for finding research expertise. *Information Processing and Management, 43*(1), 248–264.

Thomas, G. B., & Weir, M. D. (2004). *Thomas' calculus*. Addison Wesley.

Waud, D. (2003). What is tex? Available at: <http://www.tex.ac.uk/cgi-bin/texfaq2html>.

Wikipedia Formula Search. Available at: <http://shinh.org/wfs/>.

Wolfram Formula Search. Available at: <http://functions.wolfram.com/formulasearch>.

Yee, K.-P., Fisher, D., Dhamija, R., & Hearst, M. (2001). Animated exploration of dynamic graphs with radial layout. In *INFOVIS '01: Information visualization* (pp. 43–50).

Youssef, A. (2005). Search of mathematical contents: Issues and methods. In *Proceedings of the ISCA 14th international conference on intelligent and adaptive systems and software engineering* (pp. 100–105).

Youssef, A. S. (2006). Roles of math search in mathematics. In J. M. Borwein & W. M. Farmer (Eds.), *MKM '06: Proceedings of the 5th international conference on mathematical knowledge management* (pp. 2–16). Berlin Heidelberg: Springer-Verlag.