PAPER

XSemantic: An Extension of LCA Based XML Semantic Search*

Umaporn SUPASITTHIMETHEE†a), **Toshiyuki SHIMIZU**††b), ***Nonmembers***,
Masatoshi YOSHIKAWA††c), ***Member***, ***and*** **Kriengkrai PORKAEW**†d), ***Nonmember***

SUMMARY One of the most convenient ways to query XML data is a keyword search because it does not require any knowledge of XML structure or learning a new user interface. However, the keyword search is ambiguous. The users may use different terms to search for the same information. Furthermore, it is difficult for a system to decide which node is likely to be chosen as a return node and how much information should be included in the result. To address these challenges, we propose an XML semantic search based on keywords called XSemantic. On the one hand, we give three definitions to complete in terms of semantics. Firstly, the semantic term expansion, our system is robust from the ambiguous keywords by using the domain ontology. Secondly, to return semantic meaningful answers, we automatically infer the return information from the user queries and take advantage of the shortest path to return meaningful connections between keywords. Thirdly, we present the semantic ranking that reflects the degree of similarity as well as the semantic relationship so that the search results with the higher relevance are presented to the users first. On the other hand, in the LCA and the proximity search approaches, we investigated the problem of information included in the search results. Therefore, we introduce the notion of the Lowest Common Element Ancestor (LCEA) and define our simple rule without any requirement on the schema information such as the DTD or XML Schema. The first experiment indicated that XSemantic not only properly infers the return information but also generates compact meaningful results. Additionally, the benefits of our proposed semantics are demonstrated by the second experiment.

key words: *LCA, ontology, semantic search, XML, XSemantic*

1. Introduction

As described in [1], XML search tasks can be divided into Content-And-Structure (CAS) tasks where both structure and content are queried and Content-Only (CO) tasks where XML documents are searched by using keywords.

There have been several approaches in the area of CAS tasks such as in [2]–[5]. However, users still use the familiar keyword query to perform an XML search because there is no knowledge required about XML structure such as DTD or XML Schema or learning a new user interface. Therefore, a great deal of research [6]–[13], [16], [19], [20] focuses on how to improve the query results based on a keyword search.

A keyword search provides a convenient way for the users to input information, but it is difficult for a system to find and return the part of the XML document that the users are really interested in. Especially, in a large XML document, the search results may turn out to be big subtrees which are time-consuming for the users to examine.

There are many challenges in XML keyword search engines as shown in Fig. 1.

First, the users may use various terms to search for the same information. However, the classical search engine provides only term matching and fails to capture the information request semantics. For example, consider the XML tree in Fig. 2 (b) as the tree represents information in a university laboratory. When the users would like to find research works done by "Terada", different terms can be entered by the users such as "work terada", "publication terada", "research terada". Suppose that the user inputs the following keywords "research terada" as shown in Q1, Fig. 2 (a), the systems in [6]–[9], [12], [13], [16], [19], [20] fail to recognize the meaning of the keyword "research" and return an empty node in this case. In fact, a set of node ids {12, 13, 21} refers to the concept of "research" even if the exact term "research" is not used. To fill the gap between user queries and semantic queries, we build an ontology graph based on WordNet and our richer node types to translate a list of keywords into equivalent semantic queries.

Second, it is difficult to consider which XML node is likely to be chosen as a return node from pure user keywords. For example, consider Q1: "research terada" in Fig. 2 (a). Suppose that we replace the keyword "research" with "publication", the possible return nodes can be node id 1 or node id 12 because both of them contain "publication" and "terada" nodes. However, if a system assigns node id 1 as a return node, it would become a big tree because node id 1 is also a root node of XML document tree. To solve this problem, there have been several research works [6], [11]–[13], [19] using the concept of Lowest Common Ancestor (LCA) and choosing node id 12 as a return node. However, we investigated that LCA node is not always presented as the appropriate return node. Therefore, we refine LCA and define the concept of Lowest Common Element Ancestor (LCEA).

Third, the keyword queries do not specify exactly how much information should be included in the results. The

Manuscript received July 4, 2008.
Manuscript revised December 22, 2008.

†The authors are with the School of Information Technology, King Mongkut's University of Technology Thonburi, Thailand.
††The authors are with the Graduate School of Informatics, Kyoto University, Kyoto-shi, 606–8501 Japan.

*A preliminary version of part of this paper appears under the title "An Extension of LCA based XML Keyword Search" in the *International Workshop on Information-explosion and Next Generation Search* (INGS 2008), April 26, 2008.

a) E-mail: umaporn@sit.kmutt.ac.th
b) E-mail: shimizu@soc.i.kyoto-u.ac.jp
c) E-mail: yoshikawa@i.kyoto-u.ac.jp
d) E-mail: porkaew@sit.kmutt.ac.th

DOI: 10.1587/transinf.E92.D.1079

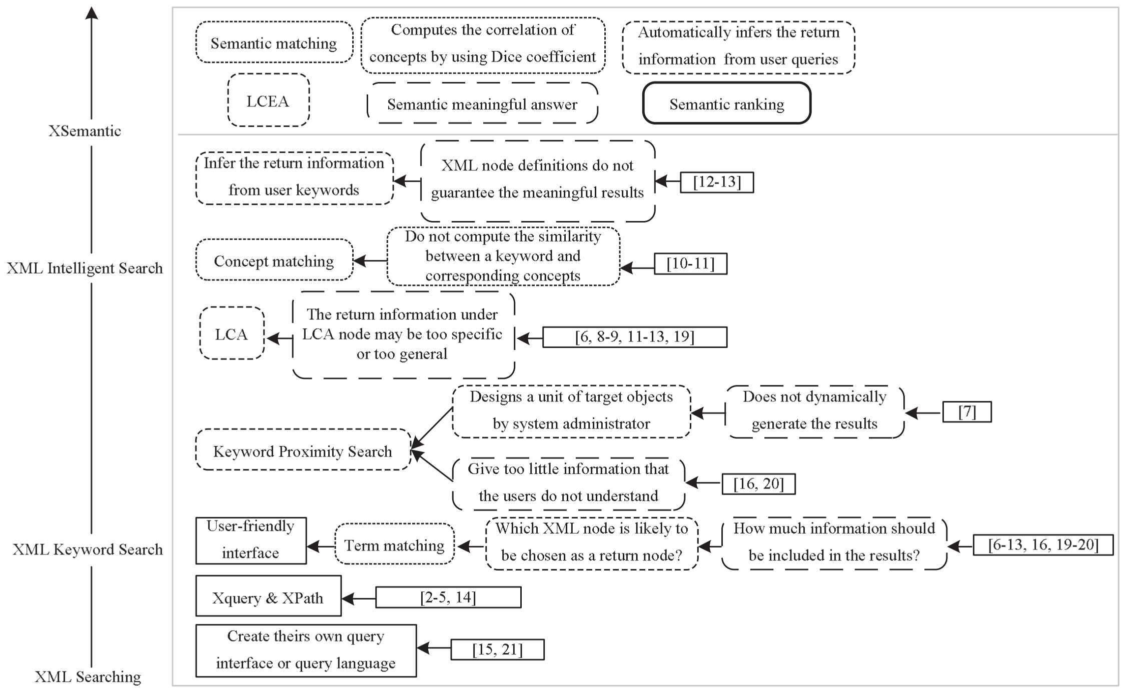
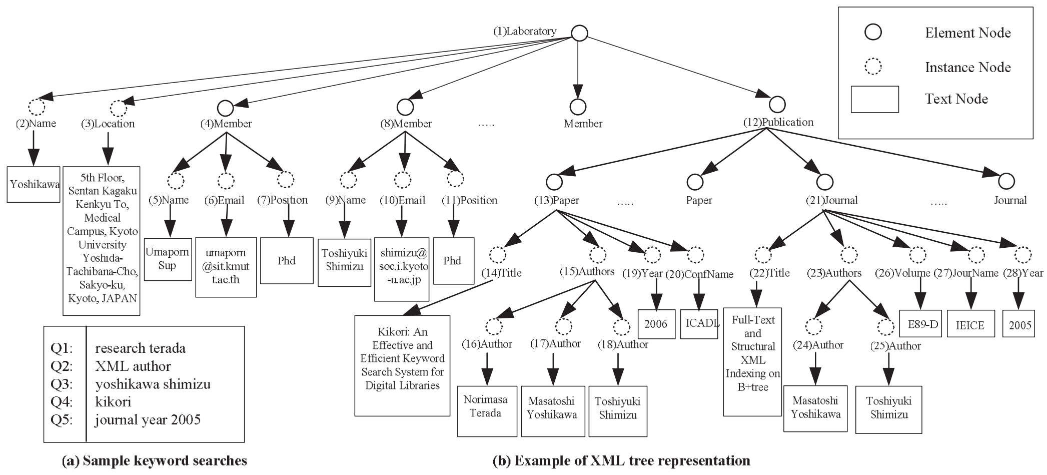

Semantic matching
Computes the correlation of concepts by using Dice coefficient
Automatically infers the return information from user queries
LCEA
Semantic meaningful answer
Semantic ranking
XSemantic
Infer the return information from user keywords
XML node definitions do not guarantee the meaningful results
[12-13]
Concept matching
Do not compute the similarity between a keyword and corresponding concepts
[10-11]
XML Intelligent Search
LCA
The return information under LCA node may be too specific or too general
[6, 8-9, 11-13, 19]
Designs a unit of target objects by system administrator
Does not dynamically generate the results
[7]
Keyword Proximity Search
Give too little information that the users do not understand
[16, 20]
User-friendly interface
Term matching
Which XML node is likely to be chosen as a return node?
How much information should be included in the results?
[6-13, 16, 19-20]
XML Keyword Search
Xquery & XPath
[2-5, 14]
Create theirs own query interface or query language
[15, 21]
XML Searching

Fig. 1 Issues, previous works and our ideas.



Q1:	research terada
Q2:	XML author
Q3:	yoshikawa shimizu
Q4:	kikori
Q5:	journal year 2005

Fig. 2 Sample keyword searches and XML tree based representation.

LCA concept has been suggested as an effective way to return the deepest node containing the keywords but it is inefficient in some cases. The information under LCA node often includes too many irrelevant nodes. Consider Q1, suppose that the keyword “research” is replaced with “publication”. LCA will return the whole subtree rooted at node id 12 which contains not just the desired paper node (node id 13) but also the unwanted journal node (node id 21) as shown in Fig. 3 (a). In contrast to XSemantic, the answer will be returned only if the paper node matches the condition plus the path to the publication node as shown in Fig. 4 (a). In addition, LCA is not guaranteed to return enough information for the users’ understanding. For example, consider Q4, LCA returns the text node under node id 14 to the users because the node itself matches a keyword “kikori” and also presents a lowest common ancestor node. To better understand the answer, in this case, XSemantic navigates up to a larger element (Paper node) which contains more information as shown in Fig. 4 (a). Another approach, the keyword proximity search, has been proposed to solve the limitation of LCA by applying the concept of shortest path to connect only a set of relevant nodes [15], [16], [20]. For example, consider Q1, it generates a path from node id 12 to connect a set of related node ids 13, 15 and 16 as shown in Fig. 3 (b). Even though it produces a result which includes only related nodes, it gives insufficient information for the users to understand. To return semantic meaningful answers, XSemantic automatically infer the return information from the

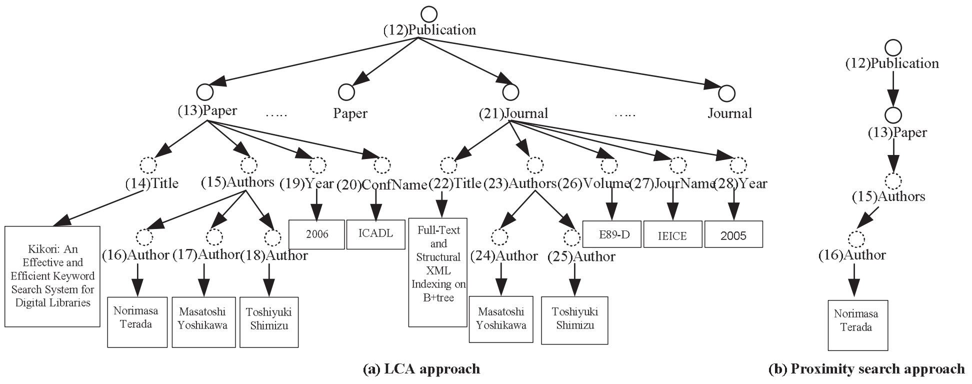


Fig. 3 Search result of Q1.

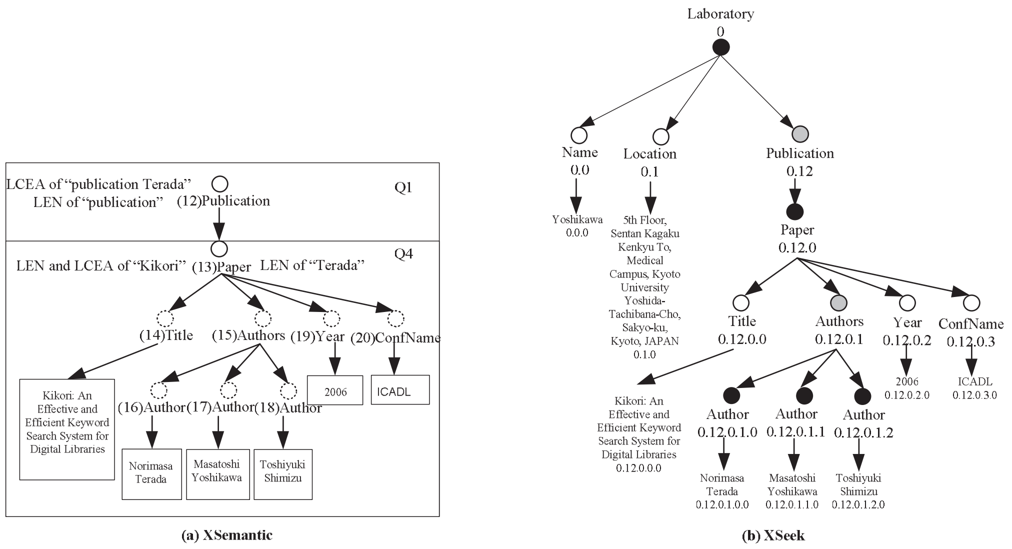


Fig. 4 Search result of Q1.

user queries. Then we use the concept of LCEA instead of LCA and our simple rule to generate compact and informative search results.

In particular, ranking the search results over our proposed semantics is one of the focuses of this paper. We present an effective ranking function that reflects the degree of the query and result similarity as well as the semantic relationship so that the search results with higher relevance are presented to the users first.

The paper is structured as follows. We begin with a discussion of related work in Sect. 2. Then the syntax of keyword search is introduced in Sect. 3. In Sect. 4, we detail our proposed semantics, namely, (1) semantic term expansion, (2) semantic meaningful answer and (3) semantic ranking. In Sect. 5, we describe XSemantic architecture and query processing. In Sect. 6, the experimental studies are presented. Finally, in Sect. 7, we offer the conclusion.

2. Related Works

There has been a lot of research done in searching XML documents. The main goals are firstly to provide a simple method to construct search queries. Secondly, to return the search results that satisfy what the users are looking for. The key point is that the results should be as compact as possible but at the same time large enough to carry meaningful information to the users.

There is research evidence that supports the former goal, for example, [15] creates its own graphical user interface which allows users to create a search query without any knowledge of the query language. [14] helps the users who have only a limited knowledge of XML structure to construct the correct XQuery. [21] proposes a flexible XML search language called XXL query language that has been designed to allow SQL-style queries on XML data. However, as opposed to introducing a new query interface or a new query language, there are still substantial research works that focus on XML keyword search [6]–[13], [16], [19], [20] because it provides the most user-friendly interface for the users to express what information they want without structure knowledge.

There has been little attempt to apply more semantic based searches which will be very important if we are to move toward the next generation search engines. [15] and

[21] provide the similarity operator (˜) which expands a keyword with similar terms but it works on the authors' user interface. [14] similarly proposes a simple function expand() to solve the problem of ambiguous terms but it is based on XQuery language. [10] proposes the conception of an XML search engine that includes the translation of user keywords into their corresponding concepts as one of the key techniques. Although [11] presents an intelligent search based on domain ontology, it does not calculate the similarity between a keyword and corresponding concepts. Furthermore, the users must always input the search words because the system cannot infer the return information from user keywords.

To achieve the latter goal, several works have been done. Most of these works [6], [11]–[13], [19] which focus on keyword queries attempt to find the smallest subtree that contains all the keywords. The concept of Lowest Common Ancestor (LCA) has been widely accepted to perform this task. Similar ideas are extended under different names, namely SLCA [8], Interconnection [9] and MLCA [14]. However, LCA is itself meaningless as explained in Sect. 1. It does not guarantee the information under LCA node. It often returns unrelated parts or lacks any additional information. Only a few research works have addressed this issue [7], [12], [13]. However, [7] does not dynamically generate the search results because it requires the system administrator to design a unit of target objects that will be presented to the users in advance.

The approach closest to ours is XSeek [12], [13]. We also aim at finding return information by inferring from user keywords and generating meaningful search results. However, XSemantic can retrieve results which XSeek can never find by using semantic term expansion, and also XSemantic can provide a ranking which XSeek does not provide. We define XML node categories differently and produce different results. XSeek classifies XML nodes into three types: *entity, attribute,* and *connection* nodes. If a node has siblings of the same name then it is considered to represent an entity. If a node does not have siblings of the same name, and it has one child, which is a value, then it is considered to represent an attribute. A node is a connection node if it represents neither an entity nor an attribute. We use black and gray circles representing entity and connection nodes respectively as shown in Fig. 4 (b). However, XSeek may return answers containing unrelated parts because only entities can be presented as a return node. Consider Q1 again, suppose that XSeek supports the semantic search and the publication node matches the keyword "research". In fact, XSeek cannot determine the meaning of a query which is one of the crucial steps in a semantic search engine. Referring to XSeek definitions, publication is a connection node because it has no siblings of the same name and also has no child value. Therefore, the publication node becomes a link node when the result is generated. XSeek then seeks the nearest ancestor (entity) node and assigns it as a master entity (return node). If such an entity cannot be found, the root node of the XML tree is considered as a return node. In this case, XSeek returns the laboratory node as the master entity that includes name and location attributes to the users as shown in Fig. 4 (b). It is clear that XSeek definitions are not concise enough to choose an appropriate return node and generate meaningful results.

3. Query Syntax

Our query interface extends the standard keyword search engine by allowing the users to explicitly specify the return information. We design the syntax of the keyword query into two parts: return information and condition information.

Query ::= [KRI:]S+
KRI ::= keyword of return information
S ::= search clause
search clause ::= search keyword|search condition

In the first part, the users can explicitly tell the system about the return information of search results. However, this part is optional. In case the users do not explicitly specify the return information, the system will automatically analyze the return information by inferring from the search clauses. The second part allows the users to express search keywords or search conditions. The users can use any functions that return a Boolean value (TRUE/FALSE) such as <, >, =, contain(). If the functions return TRUE, it means that the search conditions are met, otherwise, the nodes fail in the search conditions.

For example, "author: paper kikori", "author" is a keyword of return information and other terms "paper" and "kikori" are search keywords. This example shows that the user is interested in the information about an author who has written a paper containing the word "kikori". After our system finds nodes that satisfy the search keywords ("paper" and "kikori"), we generate the path that connects the author information and the nodes matched search keywords to the user. Another example, "publication year > 2005", it is likely that the user is interested in the information about publications which have been published since 2006. In this case, the user does not identify the return information; hence, the system automatically determines the return information by inferring from the search clause.

In this paper, we focus on a conjunctive semantic search. The search results that contain all of the search keywords and match the search conditions are returned to the users.

Given the query syntax, it is easy to transform our query syntax to other XML query languages such as XQuery or SQL statements. The return information (user defined or system inferred) can be mapped to the return node of the SELECT clause in an SQL statement or the return clause in an XQuery statement. The second part can be mapped to WHERE clause.

4. XML Semantic Search

In this section we detail the integration of our semantics to bridge the critical gap between semantic search and the

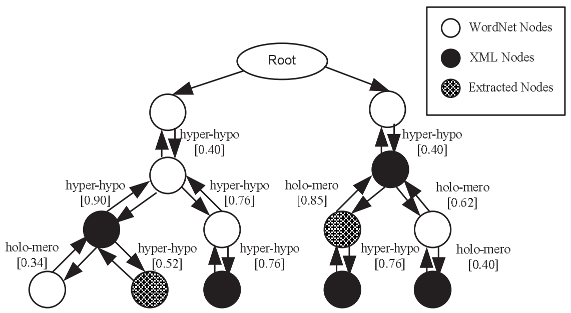


Fig. 5 The domain ontology graph.

users as follows:

4.1 Semantic Term Expansion

Each keyword specified in the user queries may be different from the actual name of the element or attribute even the term in the XML element/attribute content. To better satisfy users, we find the element name, attribute name and the term in the XML element/attribute content that conceptually relates to a given keyword by using domain ontology. The use of domain ontology allows us to perform a semantic search between keywords in the query and the terms in the XML documents.

Definition 1. *A domain ontology* is a directed graph $G_1 = (V_1, E_1)$ where V_1 is a finite set of nodes and E_1 is a finite set of edges. There are two node types: XML nodes and non-XML nodes. There is an edge between two nodes n_1 and n_2 if n_1 and n_2 are semantically related terms. Each edge e is labeled with a type of relationship and a weight which expresses the similarity of two connected nodes. The range of edge weight is (0-1]. Figure 5 shows our domain ontology graph.

Definition 2. *A non-XML node* is a WordNet node or an extracted node.

The WordNet lexical knowledge base [22] provides the different word senses and relationships between them. The type of relationship can be hypernyms and hyponyms (Gen/Spec), holonyms and meronyms (Whole/Part) and synonyms. The edges of hyponym and meronym types are reverse directions of hypernym and holonym respectively. In an implementation, we can shrink a pair of edges into a single edge yielding an undirected graph.

An extracted node is derived or generalized from the existing XML nodes by the system administrator. For example, the XML nodes that contain terms "exchange student", "ph.d.", "master", and "undergraduates", can be generalized as an extracted node "student".

For each keyword in a user query Q, we find one or more candidate concepts $\{n_1, \ldots, n_m\} \in V_1$ that match XML nodes in G_1. For $k,\ n \in V_1$ that are connected by a path $P = \{v_1{=}k,\ v_2,\ v_3,\ v_4,\ \ldots,\ v_l{=}n\}$, the similarity of the node k and the node n is the product of the edge weights along the path P. Then we use Dijkstra's algorithm [23] to compute the similarity score of all possible paths and choose the maximum scores for the similarity between the node k and the node n. For very large ontology graphs, we can stop the algorithm when the similarity falls below the threshold.

$$sim(k, n) = \max_{j=1}^{t}\left(P_j\left(\prod_{i=1}^{l-1} weight\ \langle v_i, v_{i+1}\rangle\right)\right) \tag{1}$$

where $v_i = k,\ v_l = n,\ P = possible\ paths\ P_1, \ldots, P_t$. The following algorithm illustrates how we build our domain ontology.

Step 1. The XML element names and keywords of element content are collected as XML nodes of ontology. We use stopword elimination to choose important keywords from element content. Term frequencies are also collected for ranking.

Step 2. The concepts that relate to the term given at Step 1 are searched based on WordNet lexical database. The extracted nodes can be inserted in this step.

Step 3. A variety of correlation coefficients can be used to express the degree of similarity between two concepts such as Jaccard coefficient, Dice coefficient, and Cosine coefficient. In this paper, we choose the Dice coefficient to compute the edge weights for two given concepts. Since a word may have more than one sense, the word alone is not enough to represent one of its senses. Therefore, to capture the precise meaning of the word, we use both word and its context to compute the correlation of concepts. For synonym relationship, we give the similarity equal to 1.

$$Dice \quad coefficient = \frac{2 * f(x \cap y)}{f(x) + f(y)} \tag{2}$$

where $f(c)$ is the approximate frequency of concept c and its definition which appears in web pages of large web search engines like Google or Yahoo. $f(c_1 \cap c_2)$ is the approximate frequency of concept c_1, c_2 and their definitions that appear in web pages together.

Step 4. The edge between XML nodes and related nodes is created according to the type of relationship.

Step 5. If there is no edge connecting between nodes in Step 4 then the fictitious root node is created to connect each subgraph.

Based on this domain ontology, if the specified keyword itself does not have an exact matching in the target XML documents, XSemantic can handle it by using semantic term expansion.

4.2 Semantic Meaningful Answer

Since retrieved elements which match a user query can be found at any level of granularity, an element can be a large element such as Publication or a small element such as Title. The challenge is which document portions are more meaningful to return as the query answer.

XSemantic does not always return the whole subtree rooted like the LCA approach but generally returns only a part of the subtree. We call this part a *return body*.

To identify a meaningful retrieval unit, firstly, we automatically infer the return information from the user queries.

Secondly, because user keywords may appear in different parts of the XML tree, the schematic of XML document is exploited to limit the search context as opposed to whole document and also to define our simple rule. This simple rule is used to determine which document fragments are more meaningful to return as a query answer. In a sense, the simple rule maps user queries to implicitly structured queries. Based on this rule, XSemantic may generate a return body that is specific to a small element or navigate up to larger elements which express what the user is looking for.

We model an XML document as a tree (as shown in Fig. 2 (b)) and do not consider ID/IDREF and XLink. However, our system can be extended to model XML information as a graph. We do not consider mixed contents for simplicity.

In contrast to earlier work, we do not require the schema information to classify node categories. Instead, we use the same definitions in the XPath specification [18] to define text nodes and element nodes. To avoid returning too little information to the users, we introduce our *instance node* type which is an element node close to a text node. An element node is an instance node if it has a text node as a child node or its meaning is equivalent to the meaning of another instance node. Equivalence of the meaning of nodes is determined by the system administrator. We treat XML attribute nodes as instance nodes as well. To make the notion of instance node clear, consider a set of node ids {2, 3, 5, 6, 7, 9, 10, 11, 14, 16, 17, 18, 19, 20, 22, 24, 25, 26, 27, 28} as shown in dot circles in Fig. 2 (b) representing the instance nodes. Also the set of node ids {15, 23} are the instance nodes because node id 15 and node id 23 have the same meaning as node ids {16, 17, 18} and {24, 25} respectively.

Hereafter, we use the term element node to represent the element node which is not the instance node.

We observed that the instance node and the text node alone should not be chosen as the return node because they do not carry sufficient information for the users. Therefore, in general, the top most node returned as a search result is the element node.

Definition 3. Given a search clause s, *Lowest Element Node* (*LEN*) of a matching node n of s is an XML element node which is

1. the lowest ancestor node of n (if n is an instance node or a text node); or
2. the self element node n (if n is an element node.)

We consider that two or more search clauses appear in the same context if they share the same LEN node.

According to the problem of the search results in the LCA as described in Sect. 1, we refine LCA and define the concept of Lowest Common Element Ancestor (LCEA) as shown in Definition 4.

Definition 4. Let q be a query with clauses c_1, c_2, c_k. *Lowest Common Element Ancestor* (*LCEA*) for q is an XML element node

1. of which self-or-descendant nodes contain at least one LEN for each c_i ($i = 1, 2, k$); and
2. none of its descendant element node satisfies the condition (1).

Definition 5. LCEA for q is a return node.

Note that LCEA node must be an element node because LENs are element nodes.

General Rule: A return body contains a collection of LENs for each search clause c_i and LCEA of those LENs, plus paths between each LEN and LCEA.

To clarify this rule, we break down the rule into cases as follows:

Case 1: If all matching nodes of every c_k are text nodes, then XSemantic infers that all text nodes represent the condition information. To better understand the search results, for one LEN node, we return the subtree rooted at LCEA node. For more than one LEN node, we generate the shortest path from LCEA node to each subtree rooted at each LEN node. In case the users specify the return information, the return body includes the path that connects only the user's return information and the condition information.

Example 1.1: Consider Q4: "Kikori", it would not make sense to return only the node (under node id 14) itself to the users even if it matches the keyword "Kikori". Instead, we return the subtree rooted at LCEA node (node id 13) because it carries other more useful information that is understandable by the users.

Case 2: If there is at least one keyword matching an element or instance node that has no text node descendant that matches a keyword, then XSemantic infers that this element or instance node is the return information and all other nodes that match keywords represent the condition information. In this case, the return body includes the return information, condition information and the connection between them. For the return information, we return the subtree rooted at LEN node if it matches the element node. Otherwise, we return the matched instance node and path to its LEN node. For the condition information, we return its descendant matched condition and path to its LEN node. In the case where users explicitly specify the return information, the path of the matched user's return information is connected to the return body as well. To clarify this case, we give some examples below and their results as shown in Fig. 6.

Example 2.1: Consider a query "journal", it is likely that the user is interested in the general information about the journal. In this case we return the subtree rooted at LCEA node (node id 21) to the user.

Example 2.2: Consider a query "email", it is likely that the user would like to know only the email information. In this case we generate the shortest path from LCEA node, Member node, to only the instance node Email.

Example 2.3: Consider Q2: "XML author", it indicates that the user intends to find the author who has published publications about "XML". In this example, "author" matches an instance node and "XML" matches the text node that is not a descendant of the author node. Therefore, we

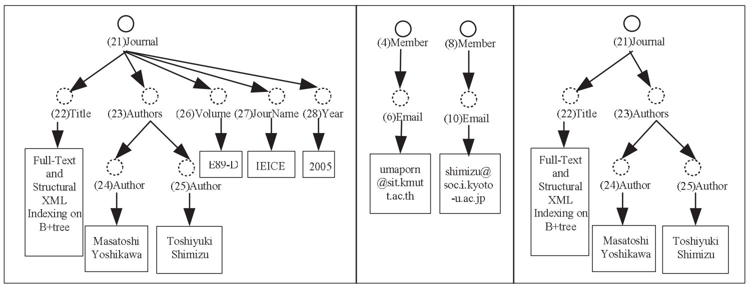


Fig. 6 The return bodies.

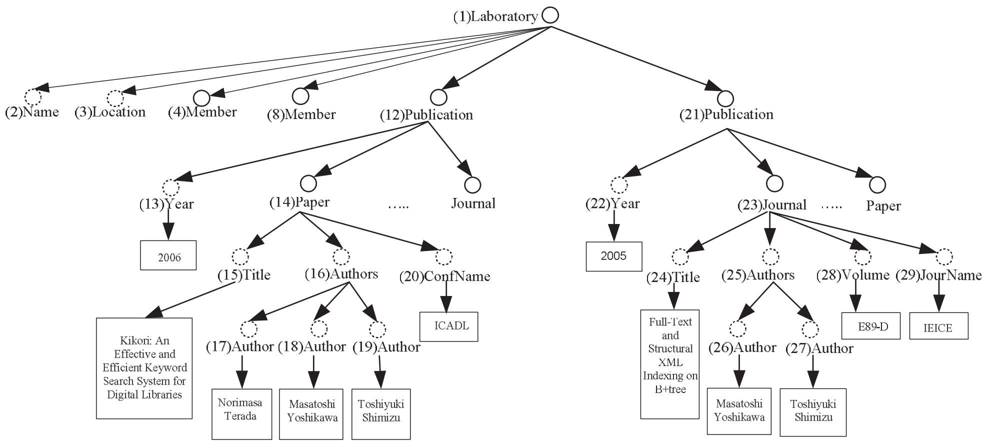


Fig. 7 A different schema of laboratory document.

generate the shortest path from LCEA node (node id 21) to the instance node id 22 containing "XML" to the instance node id 24 and 25 that match the Author node.

Case 3: As opposed to Case 2, if all element or instance nodes that match keywords have text node descendants that match some keywords as well, XSemantic infers that keywords that match the element or instance nodes represent the condition information, not the return information. If there is only one LEN node, we return the subtree rooted at LCEA node to the users. For more than one LEN node, we generate the shortest path from LCEA node to each subtree rooted at each LEN node. In the case where users explicitly specify the return information, the return body includes the path that connects only the user's return information and the condition information.

Example 3.1: Consider Q1: "research terada". Suppose that based on ontology, a term "research" matches an element node Publication (node id 12) which has a structural relationship with "terada" (node id 16). We generate the shortest path from LCEA node, Publication node, to the subtree rooted at Paper node (node id 13) that is now presented as LEN node of "terada" as shown in Fig. 4 (a).

In addition, we demonstrate that XSemantic can return meaningful results with unknown schema or multiple schemas because knowledge of XML structure is not required for the query. Consider Q5: "journal year 2005", this asks journal information published in 2005 on the same XML laboratory data in Fig. 2 (b) and Fig. 7 arranged in two different schemas. Figure 2 (b) organizes publications according to their types (paper or journal), whereas Fig. 7 organizes publications based on the year of publication. The different meaningful results of Q5 are generated as shown in Fig. 8. Based on the document structure of Fig. 2 (b), the query matches Case 3. All user keywords present the condition information. Therefore, LCEA node, Journal node (node id 21), plays a return node. Hence, XSemantic returns the subtree rooted at LCEA node which its children contain both keywords "year" and "2005". While the query is on the document structure of Fig. 7, it matches Case 2. The search keywords "year 2005" present the condition information which matches node id 22 and another keyword "journal" presents the return information which matches the LEN node id 23. Then XSemantic generates the path from Publication node which presents as LCEA node connecting the information.

4.3 Semantic Ranking

When searching for large XML documents, it is possible that there may be more than one search result that matches. It is important to rank them effectively so that the most relevant results appear first. As opposed to a keyword search over a flat HTML document, a nested structure of an XML document must be taken into account. The ranking has to be

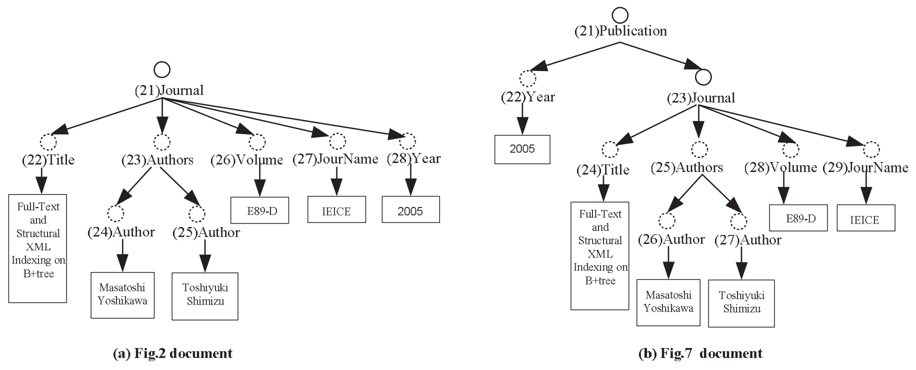


Fig. 8 Search result of Q5 with multiple schemas.

done at the granularity of XML nodes rather than the entire XML documents. Given the user queries, the search results should be ranked in descending order of relevant semantics. Our ranking function takes the following three factors into consideration.

1) Semantic similarity based on the domain ontology is denoted as $sim(k, n)$.

2) Term frequency is denoted by $freq(n)$ to express weight of the XML node n. $Freq(n)$ is equal to the number of times that n appears in the element, instance or text node divided by the maximum number of times that a word appears in that element, instance or text node.

The following formula is defined for the node score of node n.

$$ns(n) = freq(n) * sim(k, n) \tag{3}$$

The accumulated node score alone is not enough to determine the degree of semantic relationship between concepts because they may appear in different parts of the XML tree. Therefore, to properly rank search results, we should prefer potential answers that concepts can be reached through only a few nodes. We exploit our node types to compute the weight of nodes that are in the spanning tree. If nodes along the spanning tree belong to zero or one element node type, it means that they share a common parent/ancestor and are more semantically related. For example, consider the result groups in Q3, a set of node ids {15, 17, 18} and {23, 24, 25} are the spanning tree of the first and the second groups respectively. There are no element node types appearing along the path. Therefore, we can conclude that both author nodes appear in the same context of Paper and Journal node and they have a strong relationship with each other. In contrast to the third result group, the path between keyword "yoshikawa" and "shimizu" contains two element nodes, Laboratory and Member node. It indicates that node ids 2, 9, and 10 are not semantic related because node id 2 appears in the context of Laboratory as opposed to node id 9 and 10 appear in the context of Member. Therefore, the third result is ranked behind the first and the second results.

To assess the compactness of search results as described above, we shrink each directed edge of the XML tree into an undirected edge yielding an undirected acyclic graph G_2=(V_2, E_2) where V_2 is a set of XML nodes and E_2 is a set of edges connecting between XML nodes. Let $wtype$ be the weight of node type and let $\{t_1, t_2, t_3, \ldots, t_n\} \in V_2$ be nodes that are in the spanning tree T. We assign the weight of each node type as $wtype(element) > wtype(instance) > wtype(text)$.

3) Compactness $C(N)$ of a potential search result N is the reciprocal of the sum of weights of all nodes in the minimal spanning tree as shown below:

$$C(N) = \frac{1}{\sum\limits_{t \in T} wtype(t_i)} \tag{4}$$

Given a user query Q, the score of a potential answer N denoted by $S(N)$ is defined as the production between the sum of node score and the compactness value.

$$S(N) = C(N) * \sum_{n \in Q} ns(n) \tag{5}$$

5. System Architecture

The system architecture of XSemantic is presented in Fig. 9. XML documents are parsed and categorized according to our node types namely, element, instance and text nodes. We treat attribute nodes as the instance nodes. To speed up the query processing, three types of indexes are built into our system: the translation index, the semantic node index and the inverted semantic keyword index. We partition a text value in an XML element/attribute node in the same way of partitioning an inverted file in information retrieval [17].

5.1 Query Processing

To demonstrate our query processing, we use Q1: "research terada" and Q3: "yoshikawa shimizu" in Fig. 2 (a) as running examples. The search process of XSemantic is comprised of five major steps as follows:

Step 1. We translate the user queries into corresponding semantic queries by using the translation index. Based on our domain ontology, we can map each keyword into related concepts. For example, a set of different keywords

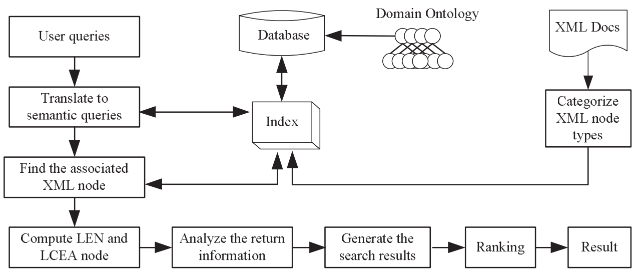

User queries
Database
Domain Ontology
XML Docs
Translate to semantic queries
Index
Categorize XML node types
Find the associated XML node
Compute LEN and LCEA node
Analyze the return information
Generate the search results
Ranking
Result

Fig. 9 The system architecture of XSemantic.

{"coauthor", "author", "writer"} can be mapped to the same concept of author with their different similarity values. After this step, a keyword is no longer a lexical string but instead a concept expresses the exact search meaning that the users are looking for.

Example: Q1, a keyword "research" is mapped to a set of concepts {"publication", "paper", "journal"}. Therefore, we can construct a set of corresponding semantic queries as {Q1.1: "publication terada", Q1.2: "paper terada", Q1.3: "journal terada"}.

Q3 is itself matching in the target XML nodes.

Step 2. For a set of semantic queries in step 1, we retrieve the associated XML nodes matching each concept by searching nodes from the semantic node index and the inverted semantic keyword index. If the users input the search conditions in this step, the conditions must be all satisfied at the same time.

Example: we find a set of matches to each semantic query as follows:

Q1.1: publication: (12), terada: (16)

Q1.2: paper (13), terada (16)

Q1.3: journal (21), terada (16)

Q3: yoshikawa: (2, 17, 24), shimizu: (9, 10, 18, 25)

Step 3. We find the Lowest Element Node (LEN) node of each concept. If there is more than one LEN node, we then compute LCEA node by using node connections stored in our indexes.

In Q1, for concept "terada", all semantic queries (Q1.1–Q1.3) have the same LEN node (node id 13).

Q1.1, LEN node of publication is itself (node id 12). Since there is more than one LEN node, we then compute LCEA node that contains both LEN nodes. According to the definition of LCEA, node id 12 presents LCEA node.

Q1.2, LEN node of paper is the same as terada.

Q1.3, LEN node of journal is node id 21 and their LCEA node is node id 12.

For Q3, LEN nodes of yoshikawa are node id {1, 13, 21}, while LEN nodes of shimizu are node id {8, 8, 13, 21} respectively. After that, we group each LEN node based on their LCEA nodes. LCEA node in the first group is paper node (node id 13) which contains LEN node {13}. LCEA node in the second group is journal node (node id 21) which contains LEN node {21}. LCEA node in the third group is laboratory node (node id 1) which contains LEN nodes {1, 8}.

Note that if each concept has the common LEN node, then we automatically assign this LEN node as LCEA node because it indicates that both concepts share a common parent/ancestor.

Step 4. After concepts are grouped according to their LCEA nodes, we analyze the return information by inferring from the user queries. We then generate the shortest path to produce a piece of information that is large enough to be meaningful by using our rule. Continue our running examples below.

Q1.1 matches Case 3. Terada is a descendant of publication node. We then generate the shortest path from LCEA node, Publication node, to the subtree rooted at LEN node of "terada".

Similar to Q1.1, Q1.2 matches Case 3 because terada is also a descendant of paper node. They share the same LEN node (node id 13). Therefore, we return the subtree rooted at LCEA node, Paper node, which contains the concept "paper" and "terada".

Q1.3 matches Case 2 because terada is not a descendant of journal node. We then generate the shortest path from LCEA node, Publication node (node id 12), to descendant terada and to the subtree rooted at node id 21 that now presents LEN node of Journal node.

Another Example, Q3 matches Case 1. We can generate three answers as follows:

The first group contains one LEN node, so we return the subtree rooted at the Paper node (node id 13).

The second group, similar to the first group, there is only one LEN node. Therefore, we return the subtree rooted at the Journal node (node id 21).

The third group, LCEA node is the laboratory (node id 1). Therefore, we generate the shortest path from the Laboratory node to all instance nodes including the concept "yoshikawa" to the subtree rooted at the Member node (only node id 8) that contains concept "shimizu".

Step 5. Multiple candidate answers may be produced like Q1 and Q3. Therefore, it is important to rank the answers according to the degree of relevance.

Consider Q1, the result of semantic query Q1.1 is first presented because based on our ontology the term "publication" receives the highest semantic similarity score with the term "research" and also has a strong relationship with another keyword "terada". For Q1.3, it has a lower total score than Q1.2. The reason is that Journal node is not a direct parent of Terada node.

Another example Q3, the first and second groups are the most relevant results because both yoshikawa and shimizu concepts appear in the same context as the Paper and Journal node respectively. Therefore, our ranking will return these results as the first answer. We also return the third group but after the first two answers because "yoshikawa" which matches the context of laboratory, while "shimizu" matches the different context member.

Table 1 Test queries for Reed college course description.

No.	Query	Keyword(s)	XSeek node types	XSemantic node types
QR_1	Find all course	course	entity	element
QR_2	Find all instructor	instructor	attribute	instance
QR_3	Find the course which contains the keyword *anth* in the subj	course, subj, anth	entity, attribute, value	element, instance, text
QR_4	Find all instructor of the course	instructor, course	attribute, entity	instance, element
QR_5	Find the course which contains the keyword *anth* in the subj and the crse is *211*	course, subj, anth, crse, 211	entity, attribute, value, attribute, value	element instance, text, instance, text
QR_6	Find the course which contains the keyword *anthropology*	course, anthropology	entity, value	element, text
QR_7	Find the title of the course which contains the keyword *431*	course, 431, title	entity, value, attribute	element, text, instance
QR_8	Find the subj and the instructor of the course	course, subj, instructor	entity, attribute, attribute	element, instance, instance

Table 2 Test queries for Yahoo auction.

No.	Query	Keyword(s)	XSeek node types	XSemantic node types
QY_1	Find all bid_history	bid_history	connection	element
QY_2	Find the opened	opened	attribute	instance
QY_3	Find the seller_info which contains the keyword *webaxion*	seller_info, webaxion	connection, value	element, text
QY_4	Find the opened of the auction_info	opened, auction_info	attribute, connection	instance, element
QY_5	Find the listing which the seller_info contains the keyword *jenzen12* and the item_info contains the keyword *RDRAM*	listing, seller_info, jenzen12, item_info, RDRAM	entity, connection, value, connection, value	element, element, text, element, text
QY_6	Find the auction_info which contains the keyword *mike* in the bidder_name	auction_info, bidder_name, mike	connection, attribute, value	element, instance, text
QY_7	Find the highest_bid_amount of the keyword *scanner*	highest_bid_amount, scanner	attribute, value	instance text
QY_8	Find the high_bidder and the memory of the listing	listing, high_bidder, memory	entity, connection, attribute	element, instance, instance

6. Experiments

We set up a couple of experiments in order to assess XSemantic search quality and to point out improvements on a classical search engine by applying our proposed semantic techniques. XSemantic was implemented in Java. We stored XML data and built our index on a relational database system. To reduce the query processing cost, we precomputed the similarity score of all possible paths in the domain ontology graph and recorded the maximum scores for the similarity of concepts in the same relational database.

6.1 Search Quality

In the first experiment, we compared XSemantic with XSeek as described in Sect. 2. We ignored our semantic search and ranking function in this experiment because XSeek does not support these features. The experiment was performed by using both Reed college course descriptions and Yahoo auction datasets from the XSeek website (http://xseek.asu.edu/).

Since we define XML node categories differently from XSeek, it produces different results. XSeek classifies nodes into three types: entity, attribute, and connection nodes. To demonstrate that XSemantic produces different results from XSeek, we have tested eight queries in various node types for each dataset as shown in Table 1 and Table 2.

The quality of the search results is still an open issue. There is no standard accepted measurement. However, in order to carry out the evaluation of XSemantic, we applied the precision and recall as done in XSearch [9], as shown below:

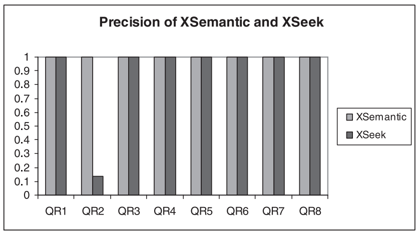
$$Precision = \frac{\text{number of correct nodes returned}}{\text{number of nodes returned}} \tag{6}$$

$$Recall = \frac{\text{number of correct nodes returned}}{\text{number of correct nodes}} \tag{7}$$

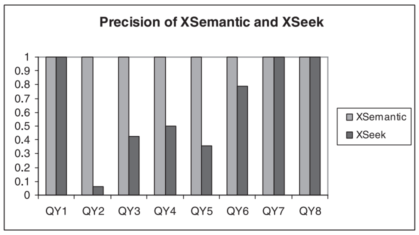
To measure precision and recall, we count the number of correct nodes that appear in the set of nodes returned by each system. We obtained correct nodes for a query by generating the XQuery which expressed the user requirements shown in the Query column. It should be noted that an equivalent XQuery was constructed based on human judgement. For example, in QY_3, the user intends to find the seller information that contains the keyword "webaxion". The corresponding XQuery was constructed as shown below:

```
for $a in //seller_info
where contains(string($a),"webaxion")
return $a
```

Then we chose the underlined keywords which ex-



(a) Reed College course description



(b) Yahoo auction

Fig. 10 Precision measurement.

pressed each of query requirements to search on both XSeek and XSemantic. All queries were run on the XSeek web-based interface (http://xseek.asu.edu/) on March 6th, 2008.

Both XSemantic and XSeek yield perfect recall. However, the precision differs in each case. The precision of XSemantic and XSeek on each dataset is presented in Fig. 10.

XSemantic has perfect precision on both Reed course descriptions and Yahoo auction datasets. On the other hand, XSeek usually has a high precision in the Reed dataset because there are only a few connection nodes. The only exception is QR_2. XSeek inferred that the users were interested in the general information about the course, so when generating the results, XSeek returned the course node as the master entity including all attribute nodes. In fact, the users would like to know only the instructor of each course but XSeek returned the whole subtree rooted at course node as the results. Interestingly for QR_4, there is the same user requirement as QR_2 but we added another keyword "course". XSemantic generated the same result as QR_2 but this is not the case for XSeek. In this query, XSeek did not infer the course as the return node like QR_2 but instead assigned the instructor node as the return node. For the remaining queries, XSemantic generated the same results as that of XSeek.

As described in Sect. 2, when the keywords match the connection node types and there are no other keywords matching the entity nodes, XSeek returns search results with irrelevant entity nodes as shown in QY_3, QY_4, and QY_6 in the Yahoo auction dataset. The listing entity node and its attributes were always returned as the nearest ancestor (entity) node when generating the results. The reason is that XSeek defines only the entity node type as a return node. In contrast to XSemantic, we define the auction_info and the seller_info as the element nodes. When generating the results, QY_3 and QY_6 matched Case 3 so we returned the subtree rooted at LCEA nodes, seller_info and auction_info respectively. For QY_4, it matched Case 2 so we generated the shortest path from LCEA node, auction_info, to only the opened instance node. For QY_1, even though XSeek has perfect precision, if we replaced the keyword "bid_history" with "history", XSeek would return the whole subtree rooted at listing node instead of returning only the bid_history node and incur low precision. The reason is that under the bid_history XSeek found another attribute node, highest_bid_amount, matching the keyword "bid". Then XSeek incorrectly inferred that the users were interested in the "highest_bid_amount" and considered the "highest_bid_amount" as the return node. When generating the result, XSeek generated the path from VLCA node, listing, to the highest_bid_amount under the bid_history connection node. For QY_2, XSeek has the lowest precision. In fact, it is the same case as QR_2. The keyword "opened" matches the attribute node type like the keyword "instructor" in QR_2 but "opened" node is the attribute node under the connection node, "auction_info", not the entity node like "course" in QR_2. Therefore, XSeek must find the nearest ancestor (entity) node. In this case the listing node presented the nearest entity node and XSeek returned the whole subtree rooted at listing node instead of only the "auction_info" node. For QY_5, we added the entity node, listing, in the search query, XSeek still returned unrelated nodes such as "auction_info" in the search results. In XSemantic, QY_5 matched Case 3 so we generated the shortest path from the listing node that presented LCEA node to each subtree root at LEN node of "seller_info" and "auction_info". For the remaining queries, both XSemantic and XSeek have perfect precision.

6.2 Semantic Search and Ranking

The main objectives of the second experiment are to test our proposed semantics. We did not compare XSemantic with INEX systems though INEX benchmark provides Content Only (CO) task. This is because search results of XSemantic are return body whereas search results of systems for INEX are always whole subtrees. We aim at finding the return information that is as compact as possible by inferring the user queries. We also did not use the two datasets from the first experiment because these datasets were designed for exact match and none ranked queries. Therefore, we manually collected XML data from Yoshikawa laboratory website (http://www.db.soc.i.kyoto-u.ac.jp). In this website, it is possible to obtain information about the labs, professors, students, staffs, and publications. The size of the XML document is 477 KB with a maximal depth of 6. Then we built our domain ontology dealing with above information based on two sources; WordNet together with extracting the new types from XML data. We chose the weight of element, instance, and text node types as 1.0, 0.5, and 0.1 respectively, which gave a good ranking result. We have tested eight queries as shown in Table 3.

To analyze the query results of our proposed seman-

Table 3 Test queries for Yoshikawa laboratory.

No.	Test Queries
QL_1	title: IEICE journal
QL_2	member mail
QL_3	yr=2008 paper
QL_4	student name email
QL_5	yoshikawa shimizu
QL_6	work terada
QL_7	research terada
QL_8	coauthor: yoshikawa XML

tics, we searched the literatures on semantic search for a reasonable amount and did not find one that could be used in our case. We decided to do a qualitative analysis by conducting user surveys on the test queries. We randomly selected 10 students who are active in the XML research area in Yoshikawa laboratory to participate in the survey. Then we used the following three metrics to assess XSemantic.

1) To evaluate the search quality of our semantic meaningful answer, each participant was asked to specify his/her desired return body for each answer. We then computed the ratio of the number of desired nodes according to user studies with the number of nodes returned by XSemantic as follows:

$$Precision = \frac{\text{number of desired nodes returned}}{\text{number of nodes returned}} \tag{8}$$

$$Recall = \frac{\text{number of desired nodes returned}}{\text{number of desired nodes}} \tag{9}$$

2) We applied the standard precision and recall to assess the quality of XSemantic especially for our semantic term expansion. Ordering of results was not considered in this step. We called Prec$_{stan}$ and Recall$_{stan}$ to prevent the reader's misunderstanding. Note that the correct answers were specified by participants.

$$Prec_{stan} = \frac{\text{number of correct answers returned}}{\text{number of answers returned}} \tag{10}$$

$$Recall_{stan} = \frac{\text{number of correct answers returned}}{\text{number of correct answers}} \tag{11}$$

3) In order to evaluate how our semantic ranking model effects the relevancy degree of retrieved answers. We computed the precision of ranking by using the following formula as a metric to compare two ranked lists: one of retrieved answers (the result of a query) and one of expected answers (specified by participants).

$$Prec_{rank} = \sum_{i \in \text{retrieved}} \frac{\text{ranking score}_i}{|\text{ rank(ret,i)} - \text{rank(exp,i)} \mid +1} * \frac{1}{\text{total ranking score}} \tag{12}$$

where *rank* (*List, i*) is the rank order of answer *i* in List. The denominator is the rank difference between the two rank lists, which are retrieved list and expected list, being compared. The difference between the two ranked lists indicates how good the ranking performs. Then the ranking score of each answer is divided by the rank difference. Finally, in order to keep the precision of ranking score to 1, we normalize the sum of evaluated ranking score by the total ranking score, thus producing a value in the range 0 (worst) to 1 (best).

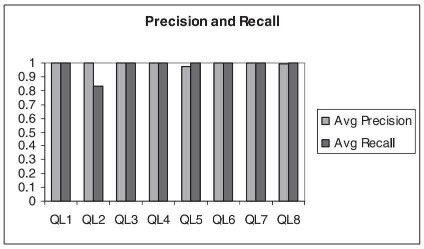


Fig. 11 Search quality.

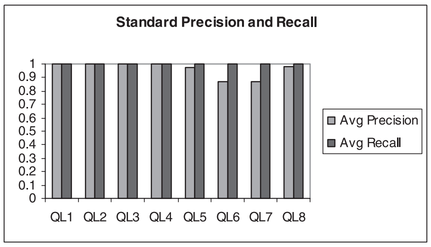


Fig. 12 Standard Precision-Recall.

Figure 11 shows the average precision and recall of the first metric. As shown in the graph, XSemantic achieves perfect precision and recall. In QL_2, a few participants would like to see other information of members rather than only email. The results of the user study confirm that not only does XSemantic choose the appropriate return nodes but also generate meaningful answers that meet the user's information needs. This is because XSemantic does not return the whole subtree rooted at LCA node. As described earlier, the information under LCA node is not guaranteed to be meaningful where in some cases, it may be too specific or too general for the users. Instead, XSemantic dynamically generates the answers by inferring from the user queries. XSemantic would return a specific answer when the users explicitly specify the return information or our system finds the implicit return information. Otherwise, we attach other more information that the users may be looking for.

The average standard precision and recall are shown in Fig. 12. The Prec$_{stan}$ of XSemantic is 96% on average with an average precision of 86.6% for the worst query (2 out of 8 queries). XSemantic achieves perfect Recall$_{stan}$ for all queries. We obtain perfect (high) precision and perfect recall in QL_2, QL_3, QL_4, QL_6, QL_7 and QL_8. The result of these queries demonstrates that XSemantic is robust from keyword ambiguity. The absence of semantic term expansion would yield the worst recall because none of the results will be generated. For example, since we generalized the new type "student" from the existing XML node types: "phd", "master", "undergraduate", we take this advantage to expand the keyword "student" in QL_4. For QL_2, QL_3 and QL_8, the keyword "mail", "yr" and "coauthor" were trans-

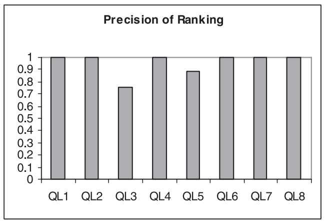


Fig. 13 Precision of ranking.

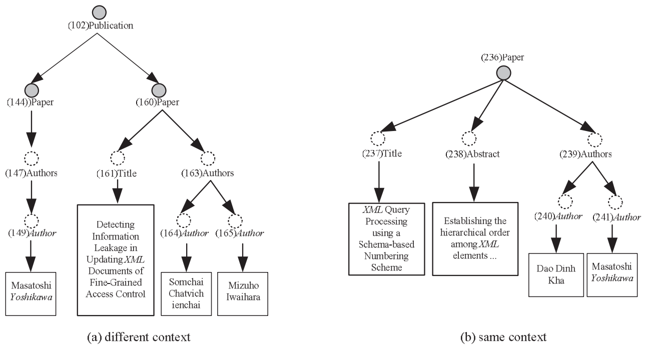


Fig. 14 Example of ranking answers.

lated into "email", "year" and "author" respectively. In QL_6 and QL_7, the keyword "work" and "research" were mapped to the same set of ontology terms, "publication", "paper", and "journal", but the different semantic similarity scores.

The average of precision of ranking is shown in Fig. 13. XSemantic has the lowest precision in QL_3. This is because when the node score of potential answers is insignificant, our system may select answers where nodes appear at a shorter distance to each other to present first. In QL_5, a few participants would like to see the results of paper and journal which contain both keywords ranked according to published year and the number of authors. However, it is out of the scope of this paper. For the remaining queries, the participants felt that our ranking function produced appropriate ranked results. The key point is that XSemantic always returns first the answer which contains all keywords and appears in the same LEN node or the same direct parent. In other words, the more LEN nodes the more different contexts and consequently leads to less compactness.

To demonstrate the effectiveness of our ranking model, consider QL_8, this query may be retrieved in many answers because "yoshikawa" and "XML" keywords often appear on the Yoshikawa laboratory website. Two possible answers of QL_8 are shown in Fig. 14. The gray circles represent the context. We can see that both of the answers satisfy the user requirement. However, by looking carefully at the details in Fig. 14 (a), the return node (publication) brings together the unrelated "yoshikawa" and "XML" nodes and leads to a meaningless answer. In contrast to the answer in Fig. 14 (b), "yoshikawa" and "XML" keywords are related with a strong semantic relationship to each other by belonging to the same paper context and are regarded as a meaningful answer. To ensure that all XML nodes match the user queries are presented to the users even though they appear in a different context, thus both answers are returned but with a different ranking score. As a result, the ranks of "yoshikawa" and "XML" in Fig. 14 (b) were presented before the answer of Fig. 14 (a) with a higher ranking. This illustrates the effectiveness of our compactness factor.

7. Conclusions

In this paper, we present an extension of LCA based XML semantic search called XSemantic. To the best of our knowledge, XSemantic is the first XML semantic search that automatically infers return information from user queries. We propose three definitions of semantic, namely, (1) semantic term expansion, (2) semantic meaningful answer and (3) semantic ranking. In addition, to solve the problem of the information in the LCA and proximity search approaches, we introduce the notion of Lowest Common Element Ancestor (LCEA) and define our simple rule to generate the meaningful results. Our experiments show that XSemantic has an improved search quality and returns a good ranking result.

References

[1] INitiative for the Evaluation of XML Retrieval (INEX), http://inex.is.informatik.uni-duisburg.de

[2] T. Shimizu and M. Yoshikawa, "Full-text and structural XML indexing on B^+-tree," Proc. 16th International Conference on Database and Expert Systems Applications, LNCS 3588, pp.451–460, Copenhagen, Denmark, 2005.

[3] S. Liu, Q. Zou, and W.W. Chu, "Configurable indexing and ranking for XML information retrieval," Proc. 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp.88–95, Sheffield, United Kingdom, 2004.

[4] R. Kaushik, R. Krishnamurthy, J.F. Naughton, and R. Ramakrishnan, "On the integration of structure indexes and inverted lists," Proc. 2004 ACM SIGMOD International Conference on Management of Data, pp.779–790, Paris, France, 2004.

[5] M. Theobald, R. Schenkel, and G. Weikum, "An efficient and versatile query engine for TopX search," Proc. 31st International Conference on Very Large Data Bases Table of Contents, pp.625–636, Trondheim, Norway, 2005.

[6] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram, "XRANK: Ranked keyword search over XML documents," Proc. 2003 ACM SIGMOD International Conference on Management of Data, pp.16–27, San Diego, California, 2003.

[7] V. Hristidis, Y. Papakonstantinou, and A. Balmin, "Keyword proximity search on XML graphs," Proc. 19th International Conference on Data Engineering, pp.367–378, 2003.

[8] Y. Xu and Y. Papakonstantinou, "Efficient keyword search for smallest LCAs in XML databases," Proc. 2005 ACM SIGMOD International Conference on Management of Data, pp.527–538, Baltimore, Maryland, 2005.

[9] S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv, "XSEarch: A semantic search engine for XML," Proc. 29th International Conference on Very Large Data Bases, pp.45–56, Berlin, Germany, 2003.

[10] F. Yuan, Y.N. Hao, and G. Yu, "The study of key techniques in intelligent XML search engine," Proc. 3rd International Conference on Machine Learning and Cybernetics, pp.1194–1197, 2004.

[11] X.Y. Li, J.S. Yuan, and X.M. Yang, "Intelligent search engine for XML based on index and domain ontology," Proc. 5th International Conference on Machine Learning and Cybernetics, pp.4501–4506,

2006.
[12] Z. Liu and Y. Chen, "Identifying meaningful return information for XML keyword search," Proc. 2007 ACM SIGMOD International Conference on Management of Data, pp.329–340, Beijing, China, 2007.
[13] Z. Liu, J. Walker, and Y. Chen, "XSeek: A semantic XML search engine using keywords," Proc. 33rd International Conference on Very Large Data Bases, pp.1330–1333, Vienna, Austria, 2007.
[14] Y. Li, C. Yu, and H.V. Jagadish, "Schema-free XQuery," Proc. 30th International Conference on Very Large Data Bases, pp.72–83, Toronto, Canada, 2004.
[15] J. Graupmann, R. Schenkel, and G. Weikum, "The spheresearch engine for unified ranked retrieval of heterogeneous XML and web documents," Proc. 31st International Conference on Very Large Data Bases, pp.529–540, Trondheim, Norway, 2005.
[16] V. Hristidis, N. Koudas, Y. Papakonstantinou, and D. Srivastava, "Keyword proximity search in XML trees," IEEE Trans. Knowl. Data Eng., vol.18, no.4, pp.525–539, April 2006.
[17] W.B. Frakes and R. Baeza-Yates, Information Retrieval: Data Structures and Algorithms, Prentice-Hall, 1992.
[18] W3C, XML Path Language (XPath) Version 1.0, http://www.w3.org/TR/xpath, 1999.
[19] S. Pradhan, "An algebraic query model for effective and efficient retrieval of XML fragments," Proc. 32nd International Conference on Very Large Data Bases, pp.295–306, Seoul, Korea, 2006.
[20] M. Barg and R.K. Wong, "Structural proximity searching for large collections of semi-structured data," Proc. 10th International Conference on Information and Knowledge Management, pp.175–182, Atlanta, Georgia, USA, 2001.
[21] A. Theobald and G. Weikum, "The index-based XXL search engine for querying XML data with relevance ranking," Proc. 8th International Conference on Extending Database Technology: Advances in Database Technology, pp.477–495, 2002.
[22] C. Fellbaum, "Wordnet:An electronic lexical database," http://wordnet.princeton.edu/, MIT Press, 1998.
[23] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, Introduction to Algorithms, 2nd ed., MIT Press, 2001.

Umaporn Supasitthimethee received the B.Sc. (First Class Honours) and M.Sc. degrees from School of Information Technology, King Mongkut's University of Technology Thonburi, Thailand in 2002 and 2004 respectively. She is currently a doctoral candidate in Computer Science at School of Information Technology, King Mongkut's University of Technology Thonburi, Thailand. From September 2007 to August 2008, she has been visiting as a research student at Department of Social Informatics, Graduate School of Informatics, Kyoto University. Her current research interests include Database support for XML data retrieval.

Toshiyuki Shimizu received a B.E. degree in Engineering in 2003, M.E. degree in Information Science from Nagoya University in 2005, and a Ph.D. degree in Informatics from Kyoto University in 2008. He is currently a JSPS research fellow of the Graduate School of Informatics at the Kyoto University, and a visiting scholar of Computer Science at the University of Arizona. His current research interests include XML databases, indexing techniques for XML documents, full-text search in XML documents, and XML information retrieval. He is also a member of ACM.

Masatoshi Yoshikawa received the B.E., M.E. and Ph.D. degrees from Department of Information Science, Kyoto University in 1980, 1982 and 1985, respectively. From 1985 to 1993, he was with Kyoto Sangyo University. In 1993, he joined Nara Institute of Science and Technology as an Associate Professor of Graduate School of Information Science. From April 1996 to January 1997, he has stayed at Department of Computer Science, University of Waterloo as a visiting associate professor. From June 2002 to March 2006, he served as a professor at Nagoya University. From April 2006, he has been a professor at Kyoto University. His current research interests include XML databases, databases on the Web, and multimedia databases. He is an Editor of Information Systems (Elsevier/Pergamon) and The VLDB Journal (Springer-Verlag). He is a member of ACM, the IEEE Computer Society and IPSJ.

Kriengkrai Porkaew received the MS and PhD degrees in computer science from the University of Illinois at Urbana-Champaign, in 1996 and 2000, respectively. Since 2000, He joined the School of Information Technology (SIT) at King Mongkut's University of Technology Thonburi (KMUTT), Thailand. He is currently an assistant professor at SIT, KMUTT. His research interests include database management, multimedia information retrieval, and XML retrieval. He is a member of ACM.