Returning Clustered Results for Keyword Search on XML Documents

Xiping Liu, Changxuan Wan, and Lei Chen, *Member*, *IEEE*

Abstract—Keyword search is an effective paradigm for information discovery and has been introduced recently to query XML documents. In this paper, we address the problem of returning clustered results for keyword search on XML documents. We first propose a novel semantics for answers to an XML keyword query. The core of the semantics is the *conceptually related* relationship between keyword matches, which is based on the conceptual relationship between nodes in XML trees. Then, we propose a new clustering methodology for XML search results, which clusters results according to the way they match the given query. Two approaches to implement the methodology are discussed. The first approach is a conventional one which does clustering after search results are retrieved; the second one clusters search results actively, which has characteristics of clustering on the fly. The generated clusters are then organized into a cluster hierarchy with different granularities to enable users locate the results of interest easily and precisely. Experimental results demonstrate the meaningfulness of the proposed semantics as well as the efficiency of the proposed methods.

Index Terms—XML keyword search, search results clustering, cluster hierarchy.

1 Introduction

Keyword search is an effective paradigm for information discovery from flat documents (text, HTML, etc.). As XML documents are prevalent in various areas, it is natural to extend keyword search techniques to support XML data.

XML keyword search has attracted considerable attentions from research community recently [7], [16], [17], [18], [20], [23], [24], [30]. However, results returned by many XML search engines are still far from satisfactory from user's perspective. Some problems are listed as follows:

Overwhelming results. Results of XML keyword search are usually overwhelming for users to explore. Since XML search engines return XML fragments to users instead of documents, there are potentially much more answers for XML search. The large number of answers hinders users from identifying relevant query results easily. On one hand, users may waste time in examining irrelevant results; on the other hand, some relevant results may be overlooked.

Mixed results for ambiguous queries. Keyword queries are usually ambiguous and XML search engines are typically unable to disambiguate semantically different answers.

Example 1. An example of XML document representing a portion of a bibliography database is shown in Fig. 1. The subscripts in the labels are used to differentiate nodes with the same label. This example will be used throughout this paper. Given a keyword query "cloud computing," multiple query intentions can be interpreted from the query, e.g., a user may intend to find conferences about "cloud computing," or papers titled "cloud computing," or books about "cloud computing," etc. Results of this kind of query represent different semantics but are usually mixed, making it inconvenient for users to locate the results of interest. On the other hand, it is usually very difficult, if not impossible, to exactly figure out the desired intentions of the user in the absence of a context.

Nonintuitive interface. Query results are usually presented in a plain interface where a ranked list of results is output to users. The interface is not intuitive enough because users cannot easily understand what are in the search results. What is more, the poorly designed interface will make users refine and submit queries repeatedly, thus decreasing users' experience and causing burden on the system.

Example 2. John wants to investigate the topic of "cloud computing," and he issues a query "cloud computing" on the bib database. From the search results that are presented in a ranked list, he finds that the top results are all papers. Suppose there is a book in the results providing an overview on that topic, but that answer is ranked very low for some reasons, then this answer may be overlooked by John. When John feels frustrated in front of the long-winded search results, he decides to refine the query by adding a term "book." Actually, this refinement can be saved since there is already an answer in previous results.

These problems can be approached from different directions, e.g., top-k query processing techniques [9], ranking methods [7], and snippet generation [18]. In this paper, we try to address these problems from a different

- X. Liu and C. Wan are with the School of Information Technology, Jiangxi University of Finance and Economics, Nanchang 330013, China. E-mail: lewislxp@gmail.com, wanchangxuan@263.net.
- L. Chen is with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Clearwater Bay, Hong Kong, China. E-mail: leichen@cse.ust.hk.

Manuscript received 30 Apr. 2010; revised 7 July 2011; accepted 22 July 2011; published online 5 Aug. 2011.
Recommended for acceptance by S. Chaudhuri, Y. Chen, and J.X. Yu.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDESI-2010-04-0254.
Digital Object Identifier no. 10.1109/TKDE.2011.183.

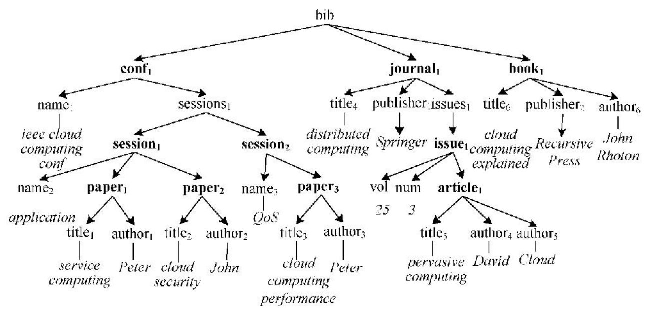


Fig. 1. A sample XML document bib.

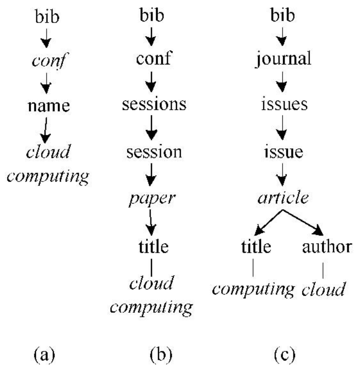


Fig. 2. KMP of (a) conf_1, (b) paper_3, and (c) $\mathrm{article}_1$.

perspective, i.e., returning clustered search results to users instead of simple list of results.

Search results clustering has been proven effective in many retrieval tasks in web search field [8]. However, the techniques developed for web search are not suitable for clustering XML search results, because they do not consider structure information, which is inherent in XML data model, and they do not utilize query semantics. Studies on clustering XML documents have also been conducted widely [14], [22], but they are also blind to query semantics and the clusters generated may be meaningless to users.

Observation on XML search results reveals that different results match the query in different ways, which indicate different matching semantics. For example, in Fig. 1, for query "cloud computing," nodes conf_1, paper_3, and $\mathrm{article}_1$ are answers, but they match the query in different ways. These answers are of different implications and interests to users. For example, users may be only interested in conf about "cloud computing" (conf_1) or paper with "cloud computing" in title (paper_3). It is desirable to cluster results such that results with different matching semantics are separated into different clusters.

With this motivation, we propose a new clustering methodology for XML search results. The basic idea is to cluster results according to the ways they match the given query. We define the notion of *keywords matching pattern* (KMP) to capture the matching semantics of the results with respect to a given query. For example, the KMPs of conf_1, paper_3, and $\mathrm{article}_1$ are depicted in Fig. 2. (The formal definition of KMP will be given later.) Then, based on the KMPs of various results, we differentiate and cluster results in such a way that results with the same or similar KMPs are grouped into one cluster. In addition, we label each cluster with the common KMP of the results so that each cluster can be understood from the label.

The clustering methodology offers several salient features. First, it helps to disambiguate keyword query. The method in its essence classifies results according to interpretations of the query. In the aforementioned example, conf_1, paper_3, and $\mathrm{article}_1$ will be in different clusters, thus are distinguished from each other. Second, by skipping irrelevant clusters, users can focus only on the desired clusters directly, thus are relieved from the overload of results. For example, if the user intends to find conference papers titled "cloud computing," he can go directly to the clusters labeled by the KMP in Fig. 2b. Third, the cluster-based interface provides a high-level view of the query results, which helps users understand and explore the search results. For example, users are informed from the labels of clusters that how many clusters there are and what they are about. Note that though the semantics we consider in this paper is targeting at data-centric XML documents, the clustering method is not confined to certain kind of documents.

In this paper, we investigate the clustering of XML search results using the new methodology. We first introduce a novel answer semantics, which lays a basis for the clustering algorithms. The semantics considers the conceptual relationship between nodes in XML trees; it leads to more meaningful answers and alleviates the problem of false answers and missing answers. Then, we investigate the implementation of the proposed clustering methodology. We propose a novel algorithm, which clusters search results actively as opposed to the traditional method of clustering in a postretrieval phase. The active algorithm exploits some special properties of the proposed answer semantics. It not only saves the time caused by the postretrieval phase, but also offers some additional benefits. Finally, we discuss the construction of a cluster hierarchy. The hierarchy provides overviews of the results with different degrees of abstraction, and can facilitate the browse and explore of search results.

The contributions of this paper are listed as follows:

1. We define a novel answer semantics for XML keyword queries. The semantics is built on a conceptual view of XML documents; it captures the semantic relationships between nodes and leads to more meaningful answers.
2. We propose a new methodology that clusters XML search results based on the concept of *keywords matching pattern*. Then, we discuss algorithms to implement the methodology. In addition to the conventional approach, we propose a novel one, which performs clustering actively, i.e., it first generates cluster labels, and then produces clustered results using these labels.
3. We improve the cluster-based interface by aggregating clusters into a hierarchy. We employ the relaxations of KMPs to aggregate the clusters. The constructed cluster hierarchy is interpretable and provides a general-to-specific view of the results.
4. We demonstrate through extensive experiments the meaningfulness of the proposed semantics and effectiveness of our clustering methodology as well as efficiency of the implementing approaches.

The remainder of the paper is structured as follows: we propose an answer semantics for XML keyword search in Section 2, and introduce a new methodology as well as algorithms to cluster XML search results in Section 3. Section 4 discusses the construction of a cluster hierarchy. Comparative analyses are conducted in Section 5 and experimental results are given in Section 6. Section 7 reviews related work and we conclude the paper in Section 8.

2 XML Keyword Query Answer Semantics

We model an XML document as a rooted, unordered, labeled tree. Each node of the tree corresponds to an XML element. We use $u \prec v$ $(u \succ v)$ to denote that u is an ancestor (descendant) of v, and $u \preceq v$ $(u \succeq v)$ to denote that $u \prec v$ or $u = v$ $(u \succ v$ *or* $u = v)$. If $u \prec v$, we also say u *contains* v, or v is *under* u in the document. We differentiate node types from nodes in XML documents. The *type* of a node is the label path from root to the node. For example, the type of node paper_1 is /bib/conf/sessions/session/paper. When there is no confusion, we sometimes use a simplified XPath-like representation of types, e.g., the previous type may be simplified as /bib//paper or even //paper. For any node v, we use $type(v)$ to denote the type of v.

An *XML keyword query* is a sequence of terms where each term corresponds to a keyword in text or an XML tag. Two example queries are frequently used in this paper, they are "cloud computing" and "John cloud computing." For brevity, we use Q_{cc} and Q_{jcc} to denote them in the following sections.

We differentiate the *match* and *answer* of a keyword query. A *match* of a term is a node whose tag is the term or a leaf whose text content contains the term. A *match* of a keyword query is a node such that for every given term, it has a descendant that matches the term. An *answer* to a keyword query is a match that is *meaningful* to users. There are many variances on the definition of meaningfulness [12], [13], [20]. In this paper, we take a new perspective, that is, we understand an XML document at a concept level. Specifically, we classify nodes according to their conceptual roles, and analyze connections between nodes based on their conceptual relationships. Then, we define the answer semantics for an XML keyword query.

2.1 Conceptually Related Entity Nodes

An XML document can be perceived as a repository of real-world entities. The reasonableness and practicability of this idea have been justified [23]. In an XML document $\mathcal{D}$, if a node represents a real-world entity, we say it is an *entity node*. Note that the term *entity* used in this paper refers to the notion defined in ER-model, rather than that in XML specification. The entity nodes in the bib database are depicted in bold. If a node represents an entity, its type represents an *entity type*. The set of entity types in bib database includes //conf, //conf//session, //conf//paper, //journal, //journal//issue, //journal//article and //book.

As entities are generally informative and interesting, it is sensible to return entity nodes as answers. Therefore, the *first guideline* of our query semantics is that, answers should be entity nodes that match the query.

Entity nodes may relate to the query in various ways, and some of them may have no practical meanings. Our *second guideline* is that, an answer should match the query in a meaningful way; in other words, the keyword matches under the answer should be meaningfully related. In this paper, we analyze the relationships between keyword matches under the conceptual view of XML documents. We first introduce our idea through some examples.

Example 3. In Fig. 1, for query Q_{jcc}, the keyword matches author_2, title_2, and title_1 are not meaningfully related, because the former two belong to entity node paper_2 whereas the last one belongs to paper_1, while paper_2 and paper_1 do not relate to each other meaningfully (we will make this clear later). On the other hand, author_6 and title_6 are meaningfully related, because they belong to the same entity node book_1. Consider another case, author_2 and name_1 may be considered as meaningfully related, because they belong to entity nodes paper_2 and conf_1, respectively, which have ancestor-descendant relationship.

We now formalize our idea. Given a node v, if u is the lowest ancestor-or-self entity of v, we say v belongs to u. Given a set of keyword matches, whether they are meaningfully related depends on the relationship between the entity nodes they belong to, so we first investigate the problem of when two entity nodes are meaningfully related.

Given two entity nodes e_1 and e_2 in $\mathcal{D}$, the connection of e_1 and e_2 in $\mathcal{D}$ can be classified into one of three categories: 1) $e_1 \preceq e_2$ or $e_1 \succeq e_2$; 2) e_1 and e_2 are connected through a common ancestor entity node e'; and 3) otherwise. Obviously, for the first case, e_1 and e_2 are meaningfully related. For the second case, the relationship between e_1 and e_2 is uncertain. For example, $\mathrm{session}_1$ and paper_3 are connected through an entity node conf_1, but they should not be deemed meaningfully related. Assume $\mathrm{publisher}_1$ is an entity node, then $\mathrm{publisher}_1$ and $\mathrm{article}_1$ are connected through $\mathrm{journal}_1$. In this case, $\mathrm{article}_1$ may be considered as meaningfully related to $\mathrm{publisher}_1$ (because $\mathrm{article}_1$ is published by the publisher). For the last case, there is no meaningful relationship between e_1 and e_2 (e.g., $\mathrm{journal}_1$ and conf_1), because they are just wrapped under some dummy nodes.

To clarify the second case, we first infer whether the node types are meaningfully related. Given an XML document, the relationships between entity types can be discovered from the schema of the document. We assume a DataGuide-like schema S is available for a document (the schema will be made clear later) [15]. Let T_1 and T_2 be two entity types, T_1 and T_2 are meaningfully related if one of the following conditions holds on S: 1) $T_1 \preceq T_2$ or $T_1 \succeq T_2$; and 2) T_1 and T_2 have a common ancestor T' in S which is also an entity type. While the first condition is self-explaining, the second one implies that both T_1 and T_2 have relationships with T', which means that T_1 and T_2 are also loosely related. For example, it is obvious that //conf is meaningfully related to //paper, because //conf is an ancestor type of //paper; //journal is not meaningfully related with //paper, because //journal and //paper has only one common ancestor type /bib, which does not represent an entity type.

Now, we are ready to define *conceptually related* entity nodes. Intuitively, two entity nodes e_1 and e_2 are conceptually related if their types are meaningfully related, and they are connected in XML tree as their types do in the schema.

Definition 1. *Let e_1, e_2 be two entity nodes with types T_1, T_2, respectively. e_1 and e_2 are conceptually related if one of the following conditions holds: 1) $e_1 \preceq e_2$ or $e_1 \succeq e_2$; and 2) e_1 and e_2 are connected through a common ancestor entity node e', T_1 and T_2 are connected through a common ancestor entity type T' (in schema), and $T' = type(e')$.*

For example, entity nodes $\mathrm{session}_1$ and paper_3 are connected through conf_1, but the types //session and //paper are connected through //session, so $\mathrm{session}_1$ and paper_3 are not conceptually related.

Based on the relationships between entity nodes, we can determine the relationships between keyword matches. Let n_1 and n_2 be two keyword matches, and n_i belongs to entity node $e_i (i = 1, 2)$. If e_1 and e_2 are conceptually related, we say n_1 and n_2 are conceptually related, denoted as $n_1 \bowtie n_2$. Given k keyword matches $n_1, \ldots, n_k$, if for any pair of matches n_i, n_j, we have $n_i \bowtie n_j$, then $n_1, \ldots, n_k$ are *conceptually related*, denoted as $n_1 \bowtie \cdots \bowtie n_k$.

For example, author_2 and name_1 are conceptually related, but author_2, name_1, title_1 are not, because author_2 and title_1 are not conceptually related.

2.2 Answer Semantics

Let v be a match of a query $Q = \langle t_1, \ldots, t_k \rangle$. If for each term t_i, v contains a match n_i of t_i, then we say n_i is a *keyword witness* of v about t_i, and $w = \langle n_1, \ldots, n_k \rangle$ is a *query witness* of v about Q; if $n_1 \bowtie \cdots \bowtie n_k$, we say w is a *meaningful witness* of v.

Given a query Q, if an entity node v matches the query, and v has a meaningful witness, then v is an *answer candidate*. Given a set of answer candidates, some of them may be redundant. For example, for query Q_{cc}, both $\mathrm{session}_2$ and paper_3 are answer candidates, but not both could be valid answers, because both have the same witness $<\mathrm{title}_3, \mathrm{title}_3>$. In this case, we prefer the more specific one, i.e., paper_3. For this issue, we have the *third guideline*, i.e., answers should be nonredundant and as specific as possible.

Based on the guidelines above, we introduce a novel answer semantics for an XML keyword query.

Definition 2. *Given an XML keyword query $Q = \langle t_1, \ldots, t_k \rangle$, an XML node v is an answer if 1) v is an answer candidate and, 2) there is not another answer candidate v' such that $v \prec v'$ and for each meaningful witness $w = \langle n_1, \ldots, n_k \rangle$ of v, w is also a witness of v'.*

Given an answer v, w is a meaningful witness of v, if v does not have a descendant answer v' such that w is also a witness of v', we say w is a *ground witness* of v.

Take Q_{cc} as an example, $\mathrm{session}_2$ is not an answer, because its only meaningful witness $<\mathrm{title}_3, \mathrm{title}_3>$ is also a witness of paper_3, a descendant of $\mathrm{session}_2$. In fact, $<\mathrm{title}_3, \mathrm{title}_3>$ is a ground witness of paper_3. conf_1 is an answer, and one of its ground witnesses is $<\mathrm{name}_1, \mathrm{name}_1>$.

Note that the proposed semantics does not remove all overlaps between answers as some methods such as SLCA and XSeek do, because we hold that some overlaps also make sense. For example, the overlap between $\mathrm{journal}_1$ and $\mathrm{article}_1$ should be allowed, since they provide different matching semantics. Our policy toward overlap is also different from ELCA and LCA. For example, $\mathrm{journal}_1$ is not an ELCA; $\mathrm{session}_1$ is an LCA but not an answer and book_1 is an answer but not LCA (title_6 is). The rationale behind the policy is that we should collect all possible answers, because it is usually impossible to exactly figure out the desired intentions, especially for ambiguous queries.

More comprehensive comparisons between the proposed semantics and other semantics are conducted in Section 5.

3 Clustering XML Search Results

In this section, we investigate the clustering of XML keyword search results. We first propose a novel methodology to cluster XML search results, then present algorithms that implement the methodology.

3.1 Clustering Methodology

The basic idea of the clustering methodology is to cluster search results based on the ways they match keyword queries. We define *keywords matching pattern* for this purpose.

Definition 3. *Given a query $Q = \langle t_1, \ldots, t_k \rangle$, let r be an answer with $w = \langle n_1, \ldots, n_k \rangle$ as a ground witness. A keywords matching pattern of r about Q derived from w is a keywords-associated twig pattern P such that: 1) the structural part is composed of the path from the root of the document to r and the paths from r to $n_i (1 \leq i \leq n)$; and 2) for each n_i that matches t_i, t_i is attached to the corresponding node of n_i in P. There is a special node in P which is mapped to r, called answer node of P.*

Example 4. Fig. 2 depicts KMPs of conf_1, paper_3, and $\mathrm{article}_1$ about query Q_{cc}, with the answer node in italic. To denote a KMP, we use an XPath-like notation [11], where the answer node corresponds to the return node (target node) of the XPath-like expression. For example, the KMPs in Fig. 2 can be denoted as: (a) //conf[name: cloud computing], (b) //paper[title: cloud computing], and (c) //article[title: computing][author: cloud].

The KMP P of an answer r about a query Q is designed to meet the following properties: 1) all terms in the query are embedded in P; 2) it represents one of the specific interpretations of Q; and 3) r matches P exactly (at the answer node). Therefore, P is a representative of the matching between r and Q, and we say that r matches Q through P.

Given an answer r of a keyword query Q, there may be multiple KMPs of r about Q, because there may be multiple ground witnesses of r, indicating that r matches Q in multiple ways. Given a set of answers, it is reasonable to cluster answers according to their KMPs, so that answers with same or similar KMPs are in the same cluster. This novel way of clustering has several desired features: 1) The process of clustering using this methodology is actually a process of disambiguation, because answers in the same cluster are semantically homogeneous and answers in different clusters are heterogeneous. 2) Each KMP is actually a specific interpretation of the query; thus, users are able to focus on the clusters with desired semantics directly. 3) The cluster-based interface presents a high-level overview on the search results, so that users can acquire a comprehensive knowledge about the search results, and potential relevant answers can be easily identified.

The clustering of search results requires that each cluster be assigned a meaningful label so that users get to know the results in the cluster from the label. Naturally, the common KMP of all results in a cluster can serve as the label of the cluster.

Now, we formalize the problem of *KMP-based XML search results clustering* as follows:

Problem definition. Given a keyword query Q and an XML database $\mathcal{D}$, construct labeled clusters such that the following conditions are met: 1) each answer belongs to at least one cluster; 2) answers in each cluster C have a unique common KMP, which is used as the label of the cluster; and 3) given an answer r, if r matches Q through P, then r is in the cluster labeled by P.

Below, we discuss how to realize this methodology in combination with the answer semantics proposed in the previous section.

3.2 The Passive Approach

A straightforward approach to implement the KMP-based clustering proceeds in three steps: 1) obtain all answers of a given query; 2) derive KMPs from answers; and 3) cluster search results based on the KMPs. Below, we briefly introduce an algorithm that follows this approach.

We first define a notion named LCEA. The LCEA of a set of nodes $n_1, \dots, n_k$ (keywords $t_1, \dots, t_k$) is the lowest common entity ancestor of the nodes (matches of keywords), denoted as $lcea(n_1, \dots, n_k)$ ($lcea(t_1, \dots, t_k)$).

Given a query $Q = \langle t_1, \dots, t_k \rangle$ and k nodes $n_1, \dots, n_k$, n_i matches $t_i (1 \leq i \leq k)$, if $r = lcea(n_1, \dots, n_k)$, and $n_1 \bowtie \cdots \bowtie n_k$, then r is an answer. Therefore, to compute answers, we can first compute all LCEAs of query terms, and then filter out the ones that do not have a meaningful witness.

Straightforward computing of LCEA of keywords on XML document according to its definition is costly. We use a different strategy here. The idea is to first find potential LCEAs, and then check each of them to see if it is an LCEA. A node is a potential LCEA if it is an entity node and contains at least one keyword match. To test if a potential LCEA v is an LCEA, we just compute the LCEA u of its keyword witnesses, if $u = v$, then v is an LCEA. Considering that the witnesses of a potential LCEA are also witnesses of its ancestors, the set of potential LCEAs are iterated in a bottom-up manner, so that the witnesses of a potential LCEA can be passed to its ancestors.

Example 5. Consider a query Q_{cc} on the bib database. Suppose the current potential LCEA is paper_1. We check paper_1 and find that it is not an LCEA because it does not contain "cloud." The next potential LCEA is paper_2, which is not an LCEA either. Then, we reach session_1. Its witnesses are obtained from paper_1 and paper_2. We compute the LCEA of the witnesses and find that session_1 is an LCEA. The other potential LCEAs are processed similarly.

After LCEAs have been found, we check whether they are answers. This step can be combined with the next step, i.e., enumerating all KMPs of the answers. Let e be an LCEA, for each witness $\langle n_1, \dots, n_k \rangle$ of e, we verify whether $n_1 \bowtie \cdots \bowtie n_k$ as defined in Section 2. If that is the case, e is an answer, and from e and $n_1, \dots, n_k$, we derive a KMP.

Example 6. To check session_1, we need test if title_1 and title_2 are conceptually related. It turns out that they are not, so session_1 is not an answer. Similarly, conf_1 is an answer, since we can find multiple meaningful witnesses, so we can derive multiple KMPs, such as //conf[name: cloud computing], //conf[name: cloud][.//paper/title: computing], etc.

After answers are obtained and KMPs are generated, it is trivial to cluster answers according to our clustering methodology. In this approach, clustering is performed in a passive way, i.e., only after all search results are obtained can we accomplish generation of clusters. Therefore, we call this approach the *passive* approach.

The pseudocode of the algorithm is put in Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2011.183.

Claim 1. *The time complexity of the passive algorithm is*

$$O\left(d\sum_{i=1}^{k} |S_i| + \sum_{j=1}^{m} (d|S^j|^2 + k^2 \prod_{i=1}^{k} |S_i^j|) + |\mathcal{P}||\mathcal{A}| \right),$$

where $S_1, \dots, S_k$ are the sets of keyword matches in the document; d is the depth of the tree; m is the number of top entities in the document, and S_i^j is the set of matches of term t_i under the jth top entity; $S^j = \cup_{i=1}^{k} S_i^j$, $|\mathcal{P}|$ is the maximal number of KMPs extracted from an answer, $|\mathcal{A}|$ is the number of answers.

The detailed analysis is given in Appendix B.1, which can be found on the Computer Society Digital Library.

3.3 The Active Approach

The passive approach is a blocking approach, because clusters can be fulfilled only after results have been generated. In this section, we propose a novel approach, which infers the labels of clusters first and then generates clusters using the labels. Below, we call this approach the *active* approach.

Two problems need to be addressed in the active approach: how to infer KMPs without actually searching the documents, and how to enumerate answers given KMPs.

3.3.1 Inferring KMPs

It is possible to generate KMPs first, because a keyword typically appears in certain document contexts, which can be identified at a low cost. We construct a summary index so that document contexts of keywords can be retrieved from that index instead of documents. The summary is like DataGuide [15], where each unique document path appears exactly once. It augments DataGuide in that each document path is associated with a list of keywords that can be accessed by traversing the path. The summary can also be modeled as a tree, and we call the tree *Data Summary Tree* or

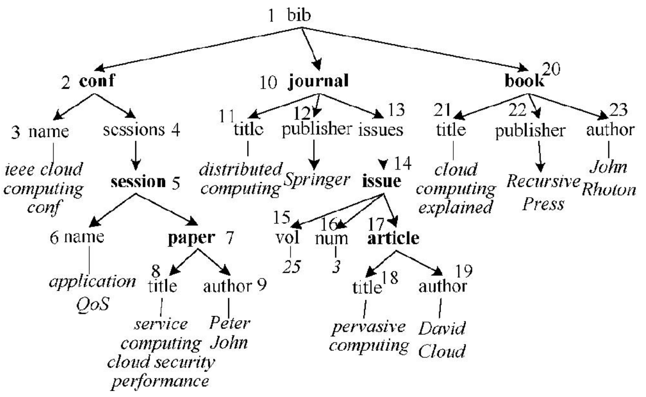


Fig. 3. The DST of bib database.

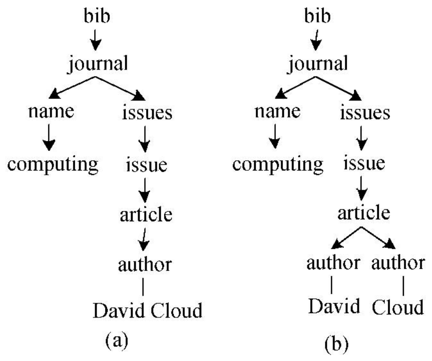


Fig. 4. Standard representation of KMP. (a) Original KMP. (b) Standard KMP.

DST in short. An example DST is shown in Fig. 3, which is summarized from the bib database in Fig. 1. The numbers in the figure are path identifiers (called *pid*). Note that each keyword appears only once under a leaf in the tree and each node or path in a DST actually represents a node type. In Fig. 3, the nodes in bold represent entity types.

Let $\mathcal{D}$ be an XML database, and $S_{\mathcal{D}}$ be the DST built on $\mathcal{D}$. Since $S_{\mathcal{D}}$ is like an ordinary XML document, we can also infer KMPs from $S_{\mathcal{D}}$.

The KMPs derived from DST may be slightly different from that of XML documents: paths are unique in the former, but not in the latter. For example, consider a query "article David Cloud," from the bib database, we may get a KMP P_1: //article[author: david][author: cloud], but from DST, we get another KMP P_2: //article[author: david cloud]. The two KMPs in fact represent the same matching semantics. To make KMPs derived from XML documents and DST comparable, we need to transform a KMP into a *standard* representation. Let n be a leaf in a KMP P with keywords $t_1, \ldots, t_i$, and suppose e is the lowest entity type containing n. We duplicate the path from e to n multiple times to get leafs $n_1, \ldots, n_i$ replacing n and assign $t_1, \ldots, t_i$ to $n_1, \ldots, n_i$, respectively. The resulting KMP is a standard KMP. For example, the KMP in Fig. 4b is a standard representation of that in Fig. 4a. From now on, when we say KMP, we mean the standard KMP.

Now, we put forward an important theorem about the relationship between KMPs derived from DST and that from documents. The theorem states that all KMPs can be safely derived from DST.

Theorem 1. *Let $Q = \langle t_1, \ldots, t_k \rangle$ be a keyword query, $S_{\mathcal{D}}$ be the DST of document $\mathcal{D}$, and P be the set of KMPs of Q derived from $S_{\mathcal{D}}$. Then, we have*

1. *for any answer r of Q on $\mathcal{D}$, if P is a KMP of r, then $P \in P$,*
2. *for each KMP $P \in \mathcal{P}$, if $r \in \mathcal{D}$, r is a match of P, i.e., r matches P exactly at the answer node, then r is an answer to Q on $\mathcal{D}$.*

Proof. The proof is listed in Appendix B.2, which can be found on the Computer Society Digital Library. □

According to this theorem, we can safely infer all KMPs from $S_{\mathcal{D}}$ without searching the document $\mathcal{D}$.

3.3.2 Generating Clustered Results

After the KMPs are inferred, generating search results and clusters can be done in a straightforward way: we iterate over all KMPs, for each KMP P, fetch all results (if any) matching P, and group them into a cluster labeled by P. Note that if P has no match, no cluster will be generated for P. Since a KMP, by its definition, is in fact a pattern query, its matches can be collected using any pattern query processing techniques.

The correctness of the active approach is guaranteed by the following theorem:

Theorem 2. *The active approach is sound and complete.*

Proof. See Appendix B.3, which can be found on the Computer Society Digital Library. □

3.3.3 Algorithm

The algorithm of active clustering is depicted in Fig. 5. It proceeds in two phases. It infers KMPs from DST in the first phase, and then evaluates the set of KMPs on the document in the second phase.

To infer KMPs from DST, we first compute the answers of Q on DST. There is a property on DST that, for any two nodes, if they have common ancestor entity types, they are conceptually related. Therefore, given a query $Q = \langle t_1, \ldots, t_k \rangle$, the LCEAs of $t_1, \ldots, t_k$ on the DST are answers of Q. In Algorithm 1, we first compute $lcea(t_1, \ldots, t_k)$ on DST in the same way as that in Section 3.2. Then, for each answer r, we infer the KMPs of r on DST in function *kmp*. The function first gets all keyword matches under r, then check each query witness to see if it is a ground witness (line 4), if true, a KMP is constructed from the witness (line 5). A KMP P from DST can be uniquely represented by a set of pairs $\{\langle p_1, t_1 \rangle, \ldots, \langle p_k, t_k \rangle\}$, where p_i is a pid indicating the context of t_i. After all KMPs have been obtained, the algorithm then evaluates each KMP on the document to get the answers. In our experiments, the KMPs are evaluated using the iTwigJoin [10] algorithm. In *evaluate* function, we first prepare the input streams for the twig join algorithm. The streams $S_1^P, \ldots, S_k^P$ for the leaf nodes of a KMP are subsets of the sets $S_1, \ldots, S_k$ of keyword matches, other streams are computed based on $S_1^P, \ldots, S_k^P$. After that, the iTwigJoin algorithm is called to evaluate the KMPs.

```
   //Let $\mathcal{S_D}$ be the DST on $\mathcal{D}$
1. let $L_1, \ldots, L_k$ be the sets of matches of $t_1, \ldots, t_k$ on
   $\mathcal{S_D}$, and $S_1, \ldots, S_k$ be the set of matches of $t_1, \ldots, t_k$
2. on $\mathcal{D}$
3. compute $R = lcea(t_1, \ldots, t_k; \mathcal{S_D})$
4. compute $\mathcal{P} = \bigcup_{r \in R} kmp(r, Q, L_1, \ldots, L_k)$
5. for each KMP $P \in \mathcal{P}$
6.     if $A$=evaluate($P$, $S_1, \ldots, S_k$) is not empty
           construct a new cluster $C'$, label it with $P$, and fill
7.         it with answers in $A$, then push it into $\mathcal{C}$
   return $\mathcal{C}$
```

```
Function kmp($r$, $Q$, $L_1, \ldots, L_k$)
1. compute the sets $L_1^r, \ldots, L_k^r$ of term matches under $r$
2. $\mathcal{P} = \emptyset$
3. for each witness $\langle p_1, \ldots, p_k \rangle$, where $p_i \in L_i^r$
4.     if $r = lcea(p_1, \ldots, p_k)$
5.         push $P = \{\langle p_1, t_1 \rangle, \cdots, \langle p_k, t_k \rangle\}$ into $\mathcal{P}$
6. return $\mathcal{P}$
```

```
Function evaluate($P$, $S_1, \ldots, S_k$)
1. let $P = \{\langle p_1, t_1 \rangle, \cdots, \langle p_k, t_k \rangle\}$
2. for each pair $\langle p_i, t_i \rangle$
3.     let $S_i^P$ be set of nodes in $S_i$ that match $\langle p_i, t_i \rangle$
4. for each node $v$ in $P$
5.     compute the stream of $v$ based on $S_1^P, \cdots, S_k^P$
6. compute $A$=twigjoin($P$) using the streams
7. return $A$
```

Fig. 5. Active clustering algorithm.

Example 7. Given a query Q_{cc}, we get the following answers on DST shown in Fig. 3: conf, paper, journal, article, and book. We take conf as an example. There are four query witnesses (denoted by pids): $<3,3>$, $<3,8>$, $<8,3>$, $<8,8>$. The last one is not a ground witness of conf, so we get three KMPs of conf. Below, we only discuss the first KMP: //conf[name: cloud][name: computing]. We evaluate the KMP on the bib document. The streams for the leafs of the KMP are $\{\text{name}_1\}$ and $\{\text{name}_1\}$, respectively, and the stream for the node //conf is $\{\text{conf}_1\}$, they are used to compute the answers of the KMP. The KMP has only one answer in the document: conf_1. So, we get a cluster labeled by the KMP which has only one answer. Other KMPs of conf are processed similarly. The results of various clusters are shown in Table 1. We finally get seven clusters, with $P_1, \ldots, P_7$ as labels, respectively.

Claim 2. *The time complexity of the active algorithm is*

$$O\left(d\sum_{i=1}^{k}|L_i| + d\sum_{j=1}^{m}\prod_{i=1}^{k}|L_i^j| + d^2|P|^2|\mathcal{P}|\sum_{i=1}^{k}|S_i| + d|P|^2|\mathcal{A}|\right),$$

where $L_1, \ldots, L_k$ are the sets of keyword matches in DST, and d is the depth of the tree; L_i^j is the set of matches in DST of term t_i under a top entity type e_j; m is the number of top entities in the document; $|P|$ is the size of P with respect to the number of nodes; $|\mathcal{P}|$ is the number of KMPs generated during the method, $S_1, \ldots, S_k$ are the sets of keyword matches on document, and $|\mathcal{A}|$ is the total number of answers.

TABLE 1
KMPs and Clusters

KMP	Label	Answers
P_1	//conf[name: cloud][name: computing]	conf_1
P_2	//paper[title: cloud][title: computing]	paper_3
P_3	//conf[name: cloud] [sessions/session/paper/title: computing]	conf_1
P_4	//conf[name: computing] [sessions/session/paper/title: cloud]	conf_1
P_5	//article[title: computing][author: cloud]	article_1
P_6	//journal[title: computing] [issues/issue/article/author: cloud]	journal_1
P_7	//book[title: cloud][title: computing]	book_1

The detailed analysis is given in Appendix B.4, which can be found on the Computer Society Digital Library.

The complexity of active algorithm is linear with respect to the total number of keyword matches in document ($\sum_{i=1}^{k}|S_i|$). This is in contrast to the passive algorithm. On the other hand, active algorithm performs heavy scan on keyword matches in DST. Generally, the frequency of keywords in the document is much higher than that in DST. For example, for DBLP 230 M document, it is typical that the frequency of a term in the document is two or three orders of magnitude greater than that in DST. So, we can roughly conclude that the active method is more efficient than the passive one.

The number of disk accesses in the passive method is $O(B)$, where B is the number of disk blocks for the keyword lists. There may be additional cost of visiting the DST in the active algorithm. The size of a DST in terms of the number of nodes and keywords is small compared with that of a document. The size of the DST is mainly determined by the number of keywords. In our experiments, the DST for DBLP 230 M document has 1,665,961 keywords, and that number on XMark 227 M document is 1,405,345. It needs no more than 30 M space to hold the DSTs, so they can be loaded into memory. Even if the DST cannot be kept in memory, we can build disk-based inverted index. At this time, we just need a few additional disk accesses, because each keyword typically has only tens or hundreds of matches in DST, and the number does not grow much with the increase of the document size.

The active approach has two options of interacting with users: 1) it first returns a set of KMPs, upon which users can specify the desired ones, and then outputs corresponding clusters; and 2) it generates all clusters and returns them to the user. Both options are desirable: in the first option, users can directly control which clusters to be generated; in the second one, the clusters can be output in order, e.g., according to some quality measures of the cluster. In our experiments, we only implement the second option, because the first one involves users' participation, making it impossible to compare with the passive method.

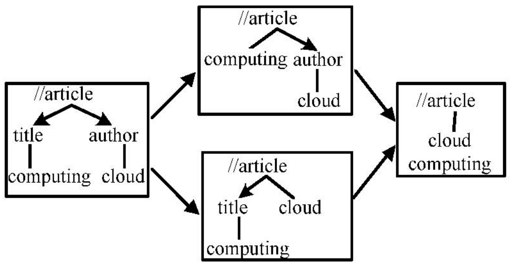


Fig. 6. A relaxation graph.

4 Constructing Cluster Hierarchy

The generated clusters so far may still not be satisfactory because there may exist many clusters so that users still have to go through the clusters one by one which is laborious. In this section, we improve the cluster-based interface by constructing a cluster hierarchy.

A cluster hierarchy on base clusters $C_1, \ldots, C_k$ is a tree where $C_1, \ldots, C_k$ constitute leafs. Since the label of a cluster is a semantical representation of the results in the cluster, we use the labeling KMP as the only feature of a base cluster. As a result, we actually discuss clustering of KMPs.

In this paper, we employ a relaxation-based approach to cluster KMPs. Query relaxation enables systems to relax a query to a less restricted form [6]. Relaxation can also be applied to a KMP to abstract from some details in the KMP, resulting in a more general one. By applying relaxations to different KMPs, we may find that some of them are identical after relaxation. These KMPs can then be grouped together, because they are more close to each other.

There are many relaxations proposed in the literature [6]. In this paper, we consider two simple relaxations: keyword predicate promotion (KPP) and leaf deletion (LD). Keyword predicate promotion moves the keyword predicate associated with a leaf in the KMP to its parent. In leaf deletion, a leaf in the KMP is deleted. For example, after conducting a KPP and an LD operation on //paper[title: cloud][title: computing], we may get //paper[title: cloud][.: computing], where "." stands for the context node. In the context of our problem, the two operations KPP and LD are done together, so we view the two operations as an atomic one, denoted as PD.

Given a KMP P, we can apply relaxing operation PD on P repeatedly, resulting in a sequence of relaxations of P, until we arrive at an ultimate relaxation of P. The ultimate relaxation of a KMP is the relaxation where the answer node becomes a leaf. The set of relaxations of a KMP can be organized into a directed acyclic graph (DAG) called *relaxation graph*, where each node represents a KMP. If a KMP P' is relaxed from P in one step, then there is an edge from the node representing P to that representing P'. For brevity, we use *node* P or simple P to denote the node representing P. Each path from the starting KMP to the ending KMP is a complete plan that transforms the original KMP to the ultimately relaxed KMP. For example, a relaxation graph is shown in Fig. 6.

The relaxation-based clustering algorithm is shown in Fig. 7. It proceeds in two phases. In the first phase, it relaxes the base KMPs and constructs an all-in-one relaxation graph on the fly. The second phase is a reducing phase, in which redundant nodes are removed so that each original KMP has exactly one relaxation route. If a node P' is relaxed from base node P in one or more steps, we say node P' *covers* P, and the set of base nodes covered by P' is called the *coverset* of P', denoted as $\Omega(P')$. Let the base KMPs be $P_1, \ldots, P_k$. In the second phase, the algorithm finds at each level, starting from level 1, a set of nodes $P'_1, \ldots, P'_n$ such that they collaboratively cover the original set of KMPs, i.e., $\cup_{i=1}^{n} \Omega(P'_i) \supseteq \{P_1, \ldots, P_n\}$, and any subset of it cannot cover the base KMPs. Then, all other nodes that are not reachable from $P'_1, \ldots, P'_n$ are deleted. This process continues until it reaches the top level.

```
    Phase 1: relaxing phase
1   create an empty graph $G$, add $n$ nodes $P_1, \ldots, P_n$ to $G$
2   for each node $P$ in $G$,
3     if $P$ can be relaxed
4        compute a one-step relaxation $P'$ of $P$
5        if node $P'$ does not exist
6          create a node $P'$ in $G$
7        add an edge from node $P$ to node $P'$ in $G$

    Phase 2: reducing phase
8   let $P^i_j$ be the $j$-th KMP at level $i$ //$P_1, \ldots, P_n$ are at level 0
9   for each level $i$ of the relaxation graph $G$
10    find a set of nodes, say $P^i_1, \ldots, P^i_{n_i}$, that are reachable
          from nodes at low levels, such that they collabora-
          tively cover $\mathcal{P}$ but any subset of them cannot
11    remove other nodes at level $i$ and their edges
```

Fig. 7. Relaxation-based clustering algorithm.

Note that the set of nodes that collaboratively cover the original set of KMPs is not unique. For this problem, we use a greedy strategy: we give priority to the nodes with greater sizes of coversets. If a node in hierarchy has very few children (e.g., fewer than four children), we try to remove the node, and make the children of the node become children of its parent. The improved hierarchy will be more reasonable in shape and more usable.

Example 8. Consider the clusters generated in Example 7, we can construct a cluster hierarchy as shown in Fig. 8 (the dummy root is not shown).

Claim 3. *The relaxation-based clustering algorithm has time complexity of $O(k|P||\mathcal{P}|)$, where k is the number keywords in the query, $|P|$ is the number of edges in P, and $|\mathcal{P}|$ is the number of KMPs to be clustered.*

The detailed analysis is given in Appendix B.5, which can be found on the Computer Society Digital Library.

In the cluster hierarchy, each cluster can be described by a KMP, which is understandable and consistent with our clustering methodology. What is more, clusters in the hierarchy provide high-level views on the KMPs as well as the associating results, which is beneficial for users.

5 Discussions

In this section, we conduct comparative analyses of our methods with existing ones. Experimental comparisons are presented in Section 6.

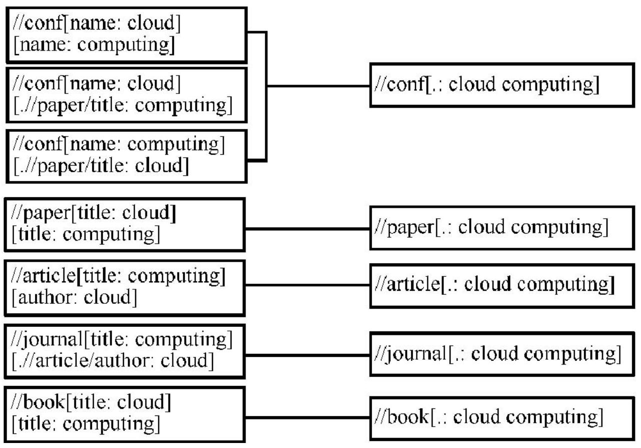

//conf[name: cloud] [name: computing]
//conf[name: cloud] [.//paper/title: computing]
//conf[name: computing] [.//paper/title: cloud]
//conf[.: cloud computing]
//paper[title: cloud] [title: computing]
//paper[.: cloud computing]
//article[title: computing] [author: cloud]
//article[.: cloud computing]
//journal[title: computing] [.//article/author: cloud]
//journal[.: cloud computing]
//book[title: cloud] [title: computing]
//book[.: cloud computing]

Fig. 8. A cluster hierarchy.

5.1 Answer Semantics

In this section, we compare the answer semantics proposed in Section 2 through the following examples with earlier methods, SLCA [30], ELCA [16], MLCA [21], VLCA and CVLCA [20], interconnection [13], $\mathcal{P}^r_{all}(S)$, $\mathcal{P}^r_{uca}(S)$, and $\mathcal{P}^r_{min}(S)$ [12], and XSeek [23].

Example 9. For query Q_{jcc}, SLCA, ELCA, and XSeek get $\mathrm{session}_1$ and book_1 as answers, but actually $\mathrm{session}_1$ is not meaningful. conf_1 could be an answer, because name_1 and author_2 may be regarded as meaningfully related under certain interpretation of the query, but conf_1 is not recognized in the methods above, nor in MLCA; name_1 and author_2 are not considered as interconnected in $\mathcal{P}^r_{uca}(S)$ *and* $\mathcal{P}^r_{min}(S)$ either. Interconnection [13], VLCA, and $\mathcal{P}^r_{all}(S)$ will falsely report that bib is an answer, because it is an LCA of name_1 and author_6. Consider another query "David Cloud," $\mathrm{article}_1$ should be an answer, but it is not in $\mathcal{P}^r_{all}(S)$, $\mathcal{P}^r_{uca}(S)$, and $\mathcal{P}^r_{min}(S)$. If one searches for "cloud computing conf publisher," bib is an MLCA because of name_1 and $\mathrm{publisher}_1$, but in fact, it is meaningless. CVLCA improves on VLCA, but it still inherits the problems from VLCA. These cases can be processed correctly in our semantics.

Compared with existing methods, the proposed semantics has the following features:

1. It improves the existing semantics by recognizing meaningful answers more accurately, and alleviates the problem of false answers and missing answers, as shown in examples above.
2. The semantics, especially the notion of "conceptually related," is based on the conceptual view of XML documents; it has a solid ground, whereas other semantics are typically motivated by some intuitive heuristics.
3. We utilize the schema information as well as the conceptual relationships between nodes to identify meaningful relation between keyword matches.
4. We take a different policy toward the overlap between (potential) answers.

By this policy, we can guarantee high recall while reasonable precision.

One potential drawback of our semantics is that we may get many weakly related results given a specific interpretation of the query. This problem can be alleviated by our clustering methodology, which can differentiate results with different semantics so that users can go directly to the desired clusters.

Another problem of our technique is that it relies on the discovery of entities. There are several optional methods to discover the entities. One is to set the entity types manually by experts. The second method is to conduct formal analysis of XML documents using the ER or certain extended ER model [27]. The third one is to derive entities based on some heuristics, such as [23]. Note that the discovery of entities is done before searching, so we could employ the most effective method. In addition, we could utilize the participation and feedback of users to improve these methods. For example, we could first find some candidate entity types, and then ask users to choose the desired ones. We could also use implicit feedback of users when they are navigating the resulting tree of search results. Due to space constraint, we do not go into the details of these issues, and just assume the set of entity nodes have been identified given an XML document. In the experiments, the entities in the documents are identified as follows: we first generate sets of entity candidates using the method in [23], and then select from the candidates.

5.2 Clustering Methodology

Recently, Liu and Chen proposed a different method of clustering XML search results in [25]. The basic intuition of the method is to cluster search results based on the categories of query terms: *search predicate* or *return node*. We show by an example the differences of two methods.

Example 10. Suppose a query Q_{cc} is issued. The clustered results by our method are shown in Table 1. We only consider paper_3, $\mathrm{article}_1$, and book_1 because others are not answers according to the semantics in [25]. Since both keywords "cloud" and "computing" match text values, both serve as predicates. Then, paper_3, $\mathrm{article}_1$, and book_1 will be grouped into one cluster by the method in [25]. Nevertheless, the three answers are very different: paper_3 and book_1 discuss "cloud computing" but are of different types, and $\mathrm{article}_1$ has "cloud" as the name of an author, which may not be desired by users. The clustering results showed in Table 1 explicitly separate these results, so that users will not feel confused.

In addition to the basic automatic clustering method, Liu and Chen [25] also allow users to explicitly specify the desired number of clusters. If that happens, the basic clusters will be split iteratively until the number of clusters is as close to the desired number as possible. For example, if the user wants three clusters, the cluster consisting of paper_3, $\mathrm{article}_1$, and book_1 will be split into three, each containing a unique result. At this time, the quality of clusters is improved, but that improvement is gained on the condition of users' involvement.

The differences between the proposed clustering method and that in [25] can be summarized as follows: 1) The clustering criteria of two methods are different. Our method is based on the matching pattern of a keyword query, i.e., KMP. Since KMP encodes much more information than the categories of keywords, it has more power in discriminating semantically different answers. 2) Our method will produce fine-grained clusters automatically, and each cluster is pure in semantics, while the clusters generated by automatic method in [25] are usually too coarse to be used. 3) The clusters generated by our method are organized in a hierarchy, and users can "drill down" or "roll up" in the hierarchy without explicit input and further computation. On the other hand, the method in [25] may involve users' participation to get desired clusters. The problem is that, users may feel reluctant to specify or do not know what number is proper. What is more, users may try multiple times to get satisfactory clusters. This process is sometimes boring and may impair users' experience. We believe that our method will be more user friendly and timesaving.

The specific clustering algorithms are also different. We propose a novel active clustering algorithm in Section 3, which has different flavor than traditional clustering methods (including the ones in [25]). The active method has an advantage over the traditional methods, that is, it allows us to have control over the process of clustering, which is especially useful in certain circumstances.

6 Experimental Evaluation

In this section, we investigate the effectiveness and efficiency of the proposed methods through experimental study. We first investigate the meaningfulness of our answer semantics, and then effectiveness of the proposed clustering methodology. We also study the efficiency as well as the scalability of the passive and active approaches.

We use three real data sets in our evaluation: DBLP (230 M) [1], Mondial (1.7 M) [2], and Sigmod Record (0.7 M) [3]. They exhibit different characteristics: DBLP has a rather flat structure, the other two are more complex in structure but Sigmod Record is richer in text. We test eight keyword queries for each data set. The queries are listed in Table 2, where queries DQ1-DQ8 are proposed for DBLP data set, MQ1-MQ8 for Mondial data set, and QS1-QS8 for Sigmod Record. The query set covers a variety of cases: keywords matching tags only (such as QD4, QM3), keywords matching values only (QD1-QD3, QM1-QM2, QS1-QS2), mixture of keywords matching tags and keywords matching values (QD5-QD7, QM5-QM8, QS3-QS8), and keywords that can match both tags and values (QD8, QM4). Keywords in the queries are expected to appear within the same text units (QM1, QM2, QS1, QS8) or different text units (QD2, QD3, QS2, etc.). The numbers of keywords in the queries vary from 1 to 8. The relationships between keyword matches in the queries are diverse, and the expected matching patterns are of different shapes. The query set also contains some queries illustrating the cases on which our method performs poorly (QM8, QS4, QS8). Some of the queries are designed by us, others are proposed by volunteers who do not participate in this work. Except for these queries, we do more tests using query sets borrowed from existing work. The results are similar, so we do not report here but put them in the Appendix, which can be found on the Computer Society Digital Library.

TABLE 2
Tested Queries

No.	Query	No.	Query
QD1	PODS	QM5	singapore country
QD2	Divesh Srivastava Dan Suciu	QM6	border f0_475 f0_358
QD3	concept model 2003	QM7	religions christian muslim
QD4	mastersthesis title	QM8	province houston dallas
QD5	author peter chen	QS1	directions database research
QD6	XML database author	QS2	Jennifer Widom Jeffrey D. Ullman
QD7	cite Codd70 Codd79	QS3	relational model author Date
QD8	journal database	QS4	volume 28 number 2 3 4
QM1	france territory	QS5	Michael Stonebraker volume 12
QM2	new york	QS6	authorPosition David J. DeWitt 00 Jim Gray 01
QM3	lake located	QS7	issuesTuple special issue rule management
QM4	river colorado	QS8	issuesTuple Jennifer Widom Dan Suciu

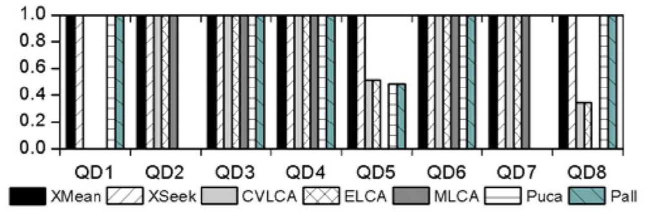
6.1 Meaningfulness of Answers

We first test the meaningfulness of the proposed answer semantics. We compare our answer semantics with several prevalent ones, they are XSeek [23], ELCA [16], MLCA [21], interconnection [12], and CVLCA [20]. We do not compare SLCA because XSeek improves on SLCA [23]. The basic interconnection [13] is not considered because CVLCA takes similar idea and experiments verify that CVLCA is superior to it [20]. For the interconnection semantics proposed in [12], we consider only $\mathcal{P}^r_{all}(S)$ and $\mathcal{P}^r_{uca}(S)$, because the third one $\mathcal{P}^r_{min}(S)$ is inferior to the former two in our experiments. We do not compare XReal [7] here because it does not adopt AND semantics between keywords so the answers are incomparable.

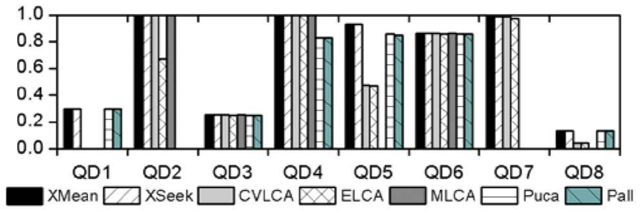
The measures we use are recall and precision. The desired results are collected in a way similar to [7], [24], [25]. The intentions of these queries are determined through user study. Specifically, we ask a group of 40 users which do not participate in this project to specify the intentions of the designed queries. For each query, the intentions that more than 80 percent users have in common are specified as the ground truth for the query. The intentions of the other queries are given by users who proposed them.

Figs. 9 and 10 illustrate the comparisons of recalls and precisions of various methods, where XMean denotes the semantics we propose in this paper, $\mathcal{P}_{uca}$ and $\mathcal{P}_{all}$ are short for $\mathcal{P}^r_{uca}(S)$ and $\mathcal{P}^r_{all}(S)$ in [12], respectively. The interconnection semantics do not specify how to return the sets of interconnected keyword matches. To make it comparable, we return answers in the same way as in XMean, that is, we return the LCEA of the interconnected nodes, so the only difference between their methods and XMean is the different ways of inferring meaningfully related nodes. The same set of entity types are used for XMean, XSeek, and interconnection.

As observed from Figs. 9 and 10, XMean and XSeek keep perfect recalls on DBLP and Sigmod Record data sets for



(a) Recall on DBLP



(b) Precision on DBLP

Fig. 9. Measures on DBLP data set.

most queries. The precisions of two methods are almost the same on DBLP. The situation on SigmodRecord is rather different, where a striking difference is observed between XMean and XSeek. The performances of $\mathcal{P}^r_{all}(S)$ and $\mathcal{P}^r_{uca}(S)$ are comparable to XMean on many queries, but unfortunately they fail to collect any answers for several queries (QD2, QD7, etc.). The differences on performances of these methods can be explained by their different ways of relating keyword matches. XSeek is based on SLCA, which does not consider the relationships between keyword matches, so it produces some meaningless answers. $\mathcal{P}^r_{all}(S)$ and $\mathcal{P}^r_{uca}(S)$ require that the keyword matches should be contained by a data tree without duplicate tags. For queries like QM1 and QS1, the semantics fits well; but for other queries like QD2, QD7, QM6, and QS2, the semantics is improper. XMean infers whether nodes are meaningfully related not according to the tags or paths of nodes, but based on the conceptual roles and relationships of nodes, so it can differentiate when duplicate tags can be allowed, and when they should not.

The other methods, ELCA, CVLCA, and MLCA, are inferior in terms of both recall and precision. They are even unable to collect any answer for some queries (nine queries for CVLCA, six for ELCA, and 12 for MLCA). The reason is twofold: their criteria of meaningfully related nodes are not proper, and answer types are improper: they usually return tiny nodes such as title, which is not satisfactory.

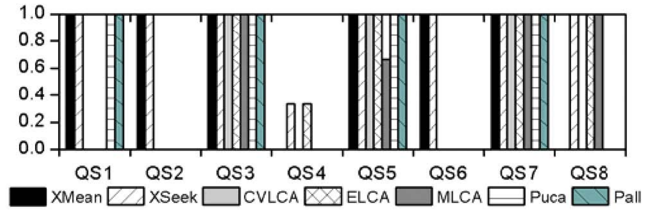
Generally, XMean has best overall performance among the methods. There are still some exceptions to XMean as illustrated by QS4, QS8, and QM8. By query QS4, the user intends to find issues of volume 28 number 2 or 3 or 4. XMean, as well as other methods, fails on this case because it adopts "AND" semantics between keywords. In fact, XSeek and ELCA find some answers for QS4 just by chance. QS8 is issued to see how "Jennifer Widom" and "Dan Suciu" are connected, e.g., whether they have common coauthors. Since "Jennifer Widom" and "Dan Suciu" appear in distinct entity nodes of the same type, they are not considered to be conceptually related in XMean. XMean fails on QM8 for similar reasons. Note that other methods with different requirements on keyword matches, i.e., CVLCA, $\mathcal{P}_{uca}$, and $\mathcal{P}_{all}$, also fail on these queries. The exceptions indicate that the accuracy of XMean can be further improved. Nevertheless, the exceptions on QS8 and QM8 can still be handled without modifying the semantics by allowing users to *search in previous results*. Take QS8 as an example, users can first issue a query "issuesTuple Jennifer Widom" to find the issues having "Jennifer Widom," and then issue another query "Dan Suciu" against the previously obtained results to get the final results. The drawback of this alternative is that users have to know how to perform such kind of search, and it may be inconvenient for users.

The measures on Mondial data set are shown in Appendix C, which can be found on the Computer Society Digital Library, which are similar to that on Sigmod Record.

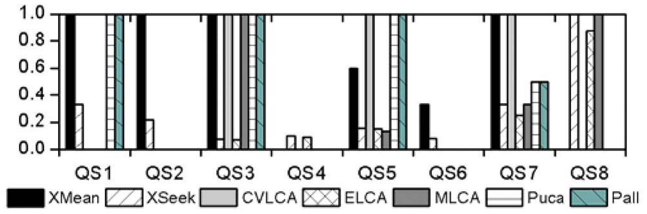
6.2 Effectiveness of the Methodology

In this section, we evaluate the retrieval effectiveness of clustering methods. Since this task involves two heterogeneous interfaces on the actually same set of answers, the traditional metrics such as recall, precision, and F-measure do not serve our purpose. For this purpose, we use the *reach time* model proposed in [19] to measure the retrieval effectiveness. Reach time is a modelization of the time taken to locate a relevant result in a certain interface. In a hierarchical interface, a typical user would inspect, from top to bottom, the node labels at each level and select one that he/she feels is most likely to contain relevant search results. The user would continue this process and drill down the hierarchy until a leaf node is reached. At this point, the user would scan the search results under that leaf node sequentially until the desired search result is found. Let h be the number of levels that must go through, s_k be the number of node labels that must be inspected at level k, and p_i be the position of the ith relevant result in the leaf node. The reach time for the ith result is

$$rt_{hierarchy} = \sum_{k=1}^{h} s_k + p_i.$$

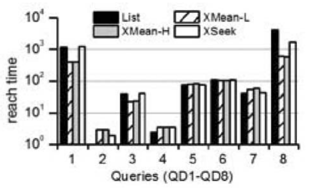


(a) Recall on Sigmod Record

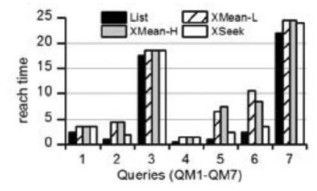


(b) Precision on Sigmod Record

Fig. 10. Measures on Sigmod Record data set.

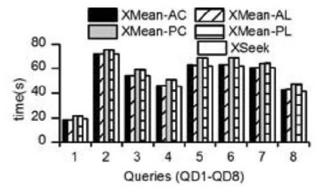


(a) reach time on DBLP

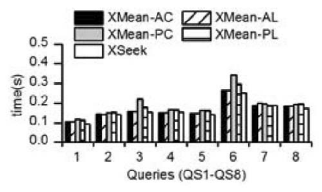


(b) reach time on Mondial

Fig. 11. Reach time on data sets.



(a) time on DBLP



(b) time on SigmodRecord

Fig. 12. Processing time on data sets.

The reach times are then averaged over the set of relevant results. For a list of results, the reach time for the ith result is simply i, so the average reach time for a list of N search results is

$$rt_{list} = \left(\sum_{i=0}^{N-1} i \right) \Big/ N = N/2.$$

Fig. 11 shows the reach times computed for the two sets of queries QD1-QD8 and QM1-QM7. We consider cluster-based interface and plain interface where clustering is not involved (denoted as **List**). For the former, we differentiate three interfaces: cluster hierarchy as described in Section 4 (denoted as **XMean-H**), cluster lists generated by our method (denoted as **XMean-L**), and that by XSeek [25] (denoted as **XSeek**). Note that we only compare with the automatic clustering method in [25], because the alternative method involves users' input, and the reach time depends on the input which may be arbitrary. To make these methods comparable, the same base sets of answers are used.

It can be observed from Fig. 11 that our clustering methods cause radical reduction on reach time compared with the List interface on QD1, QD3, and QD8. The reach times in the case of QD5-QD7 are comparable in List, XMean-L, and XMean-H methods. For QD2 and QD4, our clustering method seems to increase the reach time. The different performances are due to the different characteristics of the query. QD1, QD3, and QD8 are very ambiguous and selective. Take QD8 as an example, only one out of 57 clusters or 13.2 percent results are desired. In the List interface, users have to go through a lot of irrelevant results; but in the clustering interfaces, the irrelevant results are avoided, so the reach times are reduced. QD5-QD7 are not selective; in fact, their selectivity is above 86 percent, so clustering yields little gain. XMean achieves a perfect precision on QD2 and QD4, so the clusters are of no help, instead, they add up the reach times. The additional facility of cluster does not show its effect on Mondial data set. This is because the data set is so small and so few results are produced (no more than five for many queries).

The differences between XMean-L and XMean-H are not obvious, this is because the number of clusters for some queries is very small (less than four) so clusters are not aggregated; or because the difference is small in proportion to the reach time (QD8). Sometimes, the reach time of XMean-H is even greater than XMean-L (QD7, QM5), because the relevant answers are widely distributed among clusters, so cluster hierarchy does not show its advantage.

As for XSeek, except for QD8, the reach time on DBLP is comparable to that in List. The reason is that XSeek puts almost all results into one or two clusters for these queries, so the interface is almost the same as the plain list. The reach times of XSeek on Mondial are lower than XMean, because the numbers of answers are small, and XSeek generates few clusters than XMean, so it takes less time in browsing clusters.

From this set of experiments, we can see that our clustering method is very effective for ambiguous and selective queries on large data sets, which are common in real-world scenarios. On the other hand, the basic clustering method in XSeek is not so helpful for these cases, so users will have to explicitly specify a proper number of clusters. The performances on Mondial data set suggest that for small data sets, clustering of results has little effect on reducing browsing efforts. Nevertheless, users can still benefit from the clusters, e.g., can get a better understanding of results.

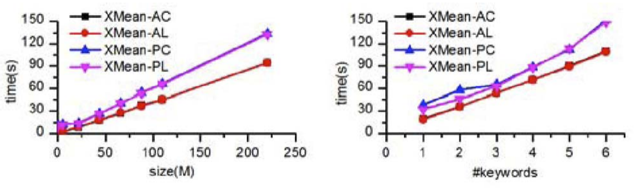
We do not report the results on Sigmod Record because they show similar characteristics.

6.3 Processing Time and Scalability

We evaluate the efficiency and scalability of proposed methods in this section.

We first evaluate the efficiency of clustering. We measure the total running times when search results are generated and clustered in passive approach or active approach, respectively. We then measure the times when search results are simply listed. Note that when results do not need to be clustered, we can still generate results using passive or active approaches. The only differences are that results are not assigned to clusters and duplicates are removed. We do not measure the delays of clustering directly because there are no separate phases of clustering in our algorithms; instead, the clustering is pushed into the search process. The efficiency of the clustering method in [25] is also investigated, where only the running time of the basic algorithm is considered. The other version of the algorithm that allows users to control the number of clusters should take more time than the basic algorithm.

The processing times of the four methods on DBLP and Sigmod Record are shown in Fig. 12, where the "P" and "A" in the suffixes stand for passive and active approaches, respectively; "C" and "L" refer to the methods generating clustered results and lists of results, respectively; and XSeek denotes the basic clustering algorithm in [25]. We can see from Fig. 12 that passive and active clustering do not lead to obvious delay in outputting search results. This is because the clustering approaches do not add much complexity to the corresponding listing approaches. Take the XMean-PC and XMean-PL methods as examples. When we get a potential answer r (i.e., an LCEA), both methods try to extract KMPs from r. The only difference is that, once a



(a) data size (b) number of keywords

Fig. 13. Processing time w.r.t. data size and number of keywords.

ground witness has been found for r, XMean-PL stops and just puts r in the answer list; on the other hand, XMean-PC has to extract all KMPs of r, and assign r to the corresponding clusters. The additional operations of XMean-PC do not lead to obvious delay, because 1) the KMPs of an answer should not be too many, so it does not need much time to extract all KMPs; and 2) we use a hash function to map a KMP to a unique cluster id, so it is very efficient to push an answer into a cluster.

Between the two approaches, the XMean-AC method is much more efficient than XMean-PC or even XMean-PL on most queries. On DBLP data set, the passive approaches are about 4-19 percent slower than their active counterparts. That figure increases to about 4-38 percent on Sigmod Record except for QS7. The efficiency of the clustering method in XSeek is comparable with the active methods, and is generally better than the passive methods. Since our answer semantics is more complex than XSeek, and our clustering methods generate much more clusters than the basic method in XSeek, we can draw a conclusion that the active clustering approach is very efficient.

The processing times of approaches on Mondial data set are shown in Appendix C, which can be found on the Computer Society Digital Library, which have similar characteristics as that on Sigmod Record.

Next, we draw a relationship between the processing time and data size. We use the XML generator in XMark [4] to generate several XML documents with size from 1 to 226 MB. Then, we evaluate query QX: "item payment creditcard" on these documents. As we can see from Fig. 13a, the differences between clustering methods and listing methods are so small that the two curves for each of two groups (the passive group and the active group) almost coincide with each other. The processing times of active methods grow at a much more steady rate compared with their passive counterparts. Generally, all methods show good scalability.

We also test the processing times of methods with increasing number of keywords. We randomly select a paper (article or inproceedings, etc.) from DBLP 230 MB document, and then extract six keywords which are not stopwords from its content. Then, we generate six queries containing one to six keywords, respectively, such that the longer queries subsume the shorter ones. The queries are then issued against DBLP document. We repeat the process above five times with different randomly selected titles. The running times of queries with length from 1 to 6 are then averaged, and is shown in Fig. 13b. We find that the scalabilities of the active methods are better than that of passive methods. The running times of passive methods are 10 to 90 percent greater than the active methods. As the queries become longer, the gaps between passive methods and active methods are more obvious. Between the two passive methods, PList shows better scalability than PCluster.

7 Related Work

There are extensive researches on XML keyword search as well as search results clustering, we review some of them in this section.

7.1 Identifying Meaningful Answers

A number of studies have discussed semantics in XML keyword search, which focus on how to effectively connect matches of keywords in a meaningful way. Most of them are based on the notion of LCA, e.g., ELCA [16], [31], SLCA [30], and MLCA [21]. These semantics are rough in that they do not consider the semantical relationships among keyword matches. XSEarch [13], [12] and VLCA [20] address this problem by introducing the concept of interconnection and homogenous nodes. These concepts, however, are not comprehensive enough.

The studies mentioned above do not have any restriction on the types of answers, assuming that each node is possible to be an answer. XSeek [23] differentiates nodes representing entities from nodes representing attributes, and generates answers based on the entities related to the keyword matches. XReal [7] differentiates *search for* node from *search via* node in XML documents, but there is no definite semantics for *search for* nodes. As a result, their method cannot guarantee the semantic meaningfulness of answers. In this paper, we follow the idea of XSeek [23] and consider XML documents as a repository of entities, which we believe is natural and reasonable.

7.2 Algorithms

Efficient algorithms for computing LCAs of two nodes on trees have been studied by previous work [5]. However, the algorithms are meant for main-memory resident data, and do not consider computing LCAs for sets of nodes. Schmidt et al. [28] propose the "meet" operator for querying XML document by computing the LCAs of nodes in XML trees. Their algorithm is thus not general enough. With LCA's variations (SLCA, ELCA, MLCA, VLCA, etc.) as query semantics, a set of algorithms [16], [20], [29], [30], [31] were proposed to compute results, which are either stack based or index based. XSeek [23] first computes SLCA and then finds the master entities. The algorithms proposed in this paper differ with those in that: 1) we adopt a different answer semantics, and many optimizations that are employed in SLCA (ELCA)-based algorithms are invalid in our context; and 2) the active approach is novel in that it transforms the vague search of XML data to exact query matching, which thus not only speeds up keyword search but also provides some additional benefits.

7.3 Understanding XML Search Results

The problem of generating result snippets for XML search is initiated in [18]. Recently, Liu et al. [26] addressed the problem of comparing and differentiating search results on structured data. Their emphasis is on the contents of XML fragments. In contrast, we concentrate on differences in matching semantics. Their work is orthogonal to our methods, e.g., results in each cluster can be further differentiated using their methods.

7.4 Clustering of Search Results and XML Documents

The clustering of search results has attracted considerable research interest [8]. However, little of them are targeting at XML search. To the best of our knowledge, the only work that addresses this problem is [25]. The differences between our method and that in [25] are analyzed and two methods have been compared experimentally. Generally, our method shows great usability whereas the method in [25] may require users' participation to be usable in many cases. Another related problem is the grouping of XML search results, which has been addressed in [17], [24], but grouping is different in nature from clustering. Recently, XML documents clustering has been the topic of many research papers [14], [22]. However, these techniques do not serve our purpose, because they are not query aware, thus are blind to query semantics, and cannot solve the problems introduced before, especially the *mixed results for ambiguous queries* and *Nonintuitive interface* problems.

8 Conclusion

In this paper, we investigate the problem of returning cluster-based search results for XML keyword search. We propose a new answer semantics for XML keyword query, which is based on a proposed *conceptually related* relationship between nodes. Then, we propose a novel clustering methodology based on the notion of keywords matching pattern. To realize the clustering methodology, we present two approaches: the first one is a conventional one, which does clustering in a post phase; the second one is novel in that it performs clustering in an active way, i.e., it first computes KMPs, then generates clustered search results using the KMPs. The generated clusters can be further improved by organizing clusters into a hierarchy. Experimental results verify the effectiveness and efficiency of our methods.

Acknowledgments

Xiping Liu and Changxuan Wan are partially supported by National Natural Science Foundation of China (NSFC) grant numbers 60763001 and 60803105. Lei Chen is partially supported by NSFC-RGC joint schema N_HKUST612/09, HKUST RPC10EG13, and NSFC Grant Nos. 60736013 and 60933011.

References

[1] "DBLP Bibliography," www.informatik.uni-trier.de/~ley/db/, 2011.

[2] http://www.cs.washington.edu/research/xmldatasets/, 2011.

[3] http://www.sigmod.org/publications/sigmod-record/xml-edition, 2011.

[4] http://www.xml-benchmark.org/, 2011.

[5] A.V. Aho, J.E. Hopcroft, and J.D. Ullman, "On Finding Lowest Common Ancestors in Trees," *Proc. Fifth Ann. ACM Symp. Theory of Computing,* 1973.

[6] S. Amer-Yahia, L.V.S. Lakshmanan, and S. Pandit, "FleXPath: Flexible Structure and Full-Text Querying for XML," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* 2004.

[7] Z. Bao, T.W. Ling, B. Chen, and J. Lu, "Effective XML Keyword Search with Relevance Oriented Ranking," *Proc. 25th Int'l Conf. Data Eng.,* 2009.

[8] C. Carpineto, S. Osiński, G. Romano, and D. Weiss, "A Survey of Web Clustering Engines," *ACM Computing Surveys,* vol. 41, no. 3, pp. 1-38, 2009.

[9] L. Chen and Y. Papakonstantinou, "Supporting Top-K Keyword Search in XML Databases," *Proc. 26th Int'l Conf. Data Eng.,* 2010.

[10] T. Chen, J. Lu, and T.W. Ling, "On Boosting Holism in XML Twig Pattern Matching Using Structural Indexing Techniques," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* 2005.

[11] J. Clark and S. DeRose, "XML Path Language (XPath) Version 1.0," W3C Recommendation, 1999.

[12] S. Cohen, Y. Kanza, B. Kimelfeld, and Y. Sagiv, "Interconnection Semantics for Keyword Search in XML," *Proc. ACM Int'l Conf. Information and Knowledge Management (CIKM),* 2005.

[13] S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv, "XSEarch: A Semantic Search Engine for XML," *Proc. 29th Int'l Conf. Very Large Data Bases,* 2003.

[14] S. Flesca, G. Manco, E. Masciari, L. Pontieri, and A. Pugliese, "Fast Detection of XML Structural Similarity," *IEEE Trans. Knowledge and Data Eng.,* vol. 17, no. 2, pp. 160-175, Feb. 2005.

[15] R. Goldman and J. Widom, "DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases," *Proc. 23rd Int'l Conf. Very Large Data Bases,* 1997.

[16] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram, "XRANK: Ranked Keyword Search over XML Documents," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* 2003.

[17] V. Hristidis, N. Koudas, Y. Papakonstantinou, and D. Srivastava, "Keyword Proximity Search in XML Trees," *IEEE Trans. Knowledge and Data Eng.,* vol. 18, no. 4, pp. 525-539, Apr. 2006.

[18] Y. Huang, Z. Liu, and Y. Chen, "Query Biased Snippet Generation in XML Search," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* 2008.

[19] K. Kummamuru, R. Lotlikar, S. Roy, K. Singal, and R. Krishnapuram, "A Hierarchical Monothetic Document Clustering Algorithm for Summarization and Browsing Search Results," *Proc. 13th Int'l Conf. World Wide Web,* 2004.

[20] G. Li, J. Feng, J. Wang, and L. Zhou, "Effective Keyword Search for Valuable LCAs over XML Documents," *Proc. 16th ACM Conf. Information and Knowledge Management,* 2007.

[21] Y. Li, C. Yu, and H.V. Jagadish, "Schema-Free XQuery," *Proc. 30th Int'l Conf. Very Large Data Bases,* 2004.

[22] W. Lian, D.W.-L. Cheung, N. Mamoulis, and S.-M. Yiu, "An Efficient and Scalable Algorithm for Clustering XML Documents by Structure," *IEEE Trans. Knowledge and Data Eng.,* vol. 16, no. 1, pp. 82-96, Jan. 2004.

[23] Z. Liu and Y. Chen, "Identifying Meaningful Return Information for XML Keyword Search," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* 2007.

[24] Z. Liu and Y. Chen, "Reasoning and Identifying Relevant Matches for XML Keyword Search," *Proc. VLDB Endowment,* vol. 1, no. 1, pp. 921-932, 2008.

[25] Z. Liu and Y. Chen, "Return Specification Inference and Result Clustering for Keyword Search on XML," *ACM Trans. Database Systems,* vol. 35, no. 2, pp. 1-47, 2010.

[26] Z. Liu, P. Sun, and Y. Chen, "Structured Search Result Differentiation," *Proc. VLDB Endowment,* vol. 2, no. 1, pp. 313-324, 2009.

[27] M. Necasky, "Conceptual Modeling for XML: A Survey," Technical Report No. 2006-3, Dept. of Software Eng., Faculty of Math. and Physics, Charles Univ., 2006, http://www.necasky.net/papers/tr2006.pdf.

[28] A. Schmidt, M. Kersten, and M. Windhouwer, "Querying XML Documents Made Easy: Nearest Concept Queries," *Proc. 17th Int'l Conf. Data Eng.,* 2001.

[29] C. Sun, C.-Y. Chan, and A.K. Goenka, "Multiway SLCA-Based Keyword Search in XML Data," *Proc. 16th Int'l Conf. World Wide Web,* 2007.

[30] Y. Xu and Y. Papakonstantinou, "Efficient Keyword Search for Smallest LCAs in XML Databases," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* 2005.

[31] Y. Xu and Y. Papakonstantinou, "Efficient LCA Based Keyword Search in XML Data," *Proc. 11th Int'l Conf. Extending Database Technology: Advances in Database Technology (EDBT),* 2008.

Xiping Liu received the PhD degree from Jiangxi University of Finance and Economics, China, in 2010. He is now a lecturer in Jiangxi University of Finance and Economics. His research interests include XML data management and information retrieval.

Changxuan Wan received the PhD degree in 2003 from Huazhong University of Science & Technology, China. He is a full professor in Jiangxi University of Finance and Economics. His research interests include web data management, data mining, and information retrieval.

Lei Chen received the PhD degree in computer science from the University of Waterloo, Canada, in 2005. He is now an associate professor in the Department of Computer Science and Engineering at Hong Kong University of Science and Technology. His research interests include probabilistic and uncertain databases, multimedia and time series databases, graph databases, social network privacy, and data management on sensor and peer-to-peer networks. He is a member of the IEEE.