Exploiting the Relationship between Keywords for Effective XML Keyword Search

Jiang Li[1], Junhu Wang[1], and Maolin Huang[2]

[1] School of Information and Communication Technology, Griffith University, Gold Coast, Australia
[2] Faculty of Engineering and Information Technology The University of Technology, Sydney, Australia

Abstract. XML keyword search provides a simple and user-friendly way of retrieving data from XML databases, but the ambiguities of keywords make it difficult to *effectively* answer keyword queries. In this paper, we tackle the keyword ambiguity problem by exploiting the relationship between keywords in a query. We propose an approach which infers and ranks a set of likely search intentions. Extensive experiments verified the better effectiveness of our approach than existing systems.

1 Introduction

Keyword search in XML databases has been extensively studied recently. However, the search effectiveness problem is far from solved. One of the main causes of the problem is the ambiguity of keywords. In particular, a word can have multiple meanings[1]. Consider the XML tree in Fig. 1. The word *16* appears as a text value of *volume* and *initPage* nodes, and the word *issue* exists as an XML tag name and a text value of *title* node. When a user types in such keywords, it is hard to know which meaning of the keyword the user wants.

In this work, we propose to tackle keyword ambiguities and infer users' search intention by exploiting the relationship between different keywords in a query. The basic observation is that users seldom issue a query arbitrarily. Instead, most of the time they construct queries logically. They usually place closely related keywords at *adjacent* positions. For example, if the user intends to retrieve the articles about database from issue 16, he is more likely to submit the query {**issue 16** database} than the query {**16** database **issue**} because the keywords "16" and "issue" are closely related. This intuition motivates us to infer a keyword's meaning by *preferentially* evaluating the relationship between this keyword and its adjacent keywords. In the example above, if the query is {**issue 16** database}, it is more likely to infer "issue" as the tag name *issue* by considering this keyword together with its adjacent keyword "16" than considering this keyword with "database".

[1] We use word type to represent a word's meaning. See Definition 5 for the definition of word type.

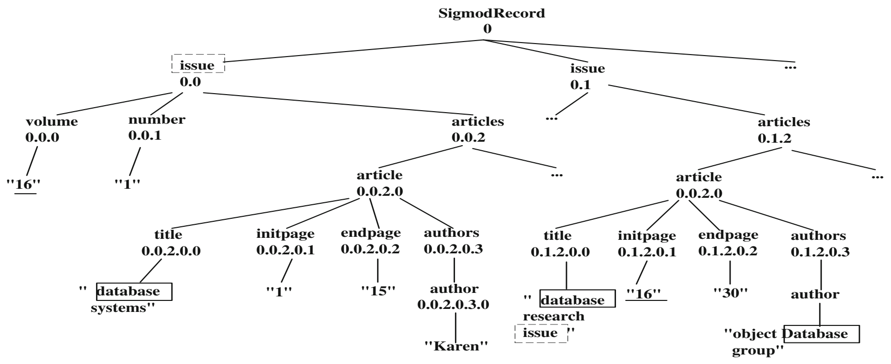


Fig. 1. A Sample XML data tree of SigmodRecord

We developed a system `XInfer` that infers a set of likely search intentions using the above intuition as well as keyword distribution in the data tree, and ranks them appropriately. For example, for the data tree in Fig. 1, if the user types the query {issue 16 database}, `XInfer` will infer that the most likely search intention as *articles in issue 16 whose title contains "database"*; if the query is {16 database issue}, it will infer the most likely search intention as *articles with initial page "16" whose title contains the words "database" and "issue"*; If the user query is {database issue 16}, then both of the afore-mentioned search intentions will be ranked highly, with the second one being ranked the highest.

Related Work. Works that are most closely related to ours include XReal [2], XBridge [3] and `XSeek` [4]. XReal uses statistics of the data (mainly term frequency) to find a *search-for node type (SNT)*, and XBridge [3] estimates the promising result types based on off-line synopsis of the XML tree (structural distribution and value distribution). `XSeek` tried to recognize the possible entities and attributes in the data tree, distinguish between search predicates and return specifications in the keywords, and return nodes based on the analysis of both XML data structures and keyword match patterns. Many other works on XML keyword search are based on variants of LCA (lowest common ancestor). One of the variants is maxMatch [5], which prunes irrelevant nodes from result subtrees obtained using the SLCA semantics. Due to page limit, we refer the readers to the full version of this paper[2] and the recent survey [6] for more details. To the best of our knowledge, no previous work has utilized the relationship between adjacent keywords when inferring the search intention.

Organization. After introducing the data model (Section 2), we design a formula to evaluate the relationship between two adjacent keywords without considering other keywords in the query, which takes into account the statistics and structural properties of a keyword's different meanings (Section 3). Then we propose the Pair-wise Comparison Strategy, which utilizes the inference results of pairs of adjacent keywords, to infer a set of likely search intentions and rank them

[2] Available from `http://www.ict.griffith.edu.au/~jw/report/xinfer.pdf`

appropriately (Section 4). In Section 5, we show how to generate result subtrees. We present our experiments in Section 6 and conclude the paper in Section 7.

2 Background

Data Model. An XML document is modeled as an unordered tree, called the *data tree*. Each *internal node* (i.e., non-leaf node) has a label, and each *leaf node* has a value. The internal nodes represent elements or attributes, while the leaf nodes represent the values of elements or attributes. Each node v in the data tree has a unique Dewey code. Fig. 1 shows an example data tree.

Entity Nodes. In reality, an XML document is usually a container of related entities. For instance, Fig. 1 is a collection of *issue* and *article* entities. We use an approach similar to that of [4] to identify entity nodes.

Definition 1. (**Entity Node**) *Let t be a data tree. A node v in t is said to be a* simple node *if it is a leaf node, or has a single child which is a leaf node. A node v is said to be an* entity node *if: (1) it corresponds to a $*$-node in the DTD (if DTD exists), or has siblings with the same tag name as itself (if DTD does not exist), and (2) it is not a simple node. The* entity type *of an entity node e refers to the node type of e. A node v in t is called a* grouping node *if it has children of entity nodes only.*

Consider the data tree in Fig. 1, nodes *issue* (0.0) and *article* (0.0.2.0) are entity nodes. The nodes *SigmodRecord* (0) and *articles* (0.0.2) are grouping nodes.

Keyword Query. A keyword query is a finite set of keywords $K = \{w_1, ..., w_n\}$. Given a keyword w and a data tree t, the search of w in t will check both the labels of internal nodes and values of leaf nodes for possible occurrences of w.

3 Inferring the Meaning of Two Adjacent Keywords

3.1 Preliminary Definitions

Definition 2. (**Node Type**) *Let* v *be a node in data tree* t. *The node type of* v *is the* sequence *of node labels on the path from the* root *to* v *if* v *is an internal node, and is denoted $l_1.l_2.\cdots.l_n$, where l_i ($1 \leq i \leq n$) is the label of the i^{th} node on the path. If* v *is a leaf node, its node type is the node type of its parent. The length of a node type is the number of nodes on the path.*

In Fig. 1, the node type of *author* (0.0.2.0.3.0) is `SigmodRecord.issue.articles.article.authors.author`. For simplicity, we will use the tag name of a node to denote the node type when there is no confusion.

Definition 3. (**Ancestor Node Type**) *Given a node type $T \equiv l_1.l_2.\cdots.l_n$, we say $l_1.l_2.\cdots.l_i$ is an* ancestor node type *of T, for any $i \in [1, n-1]$.*

We use $T_1 \prec T_2$ to denote that T_1 is an ancestor node type of T_2.

Definition 4. *Let $T_1, \cdots, T_n$ be node types. The* longest common ancestor *of $T_1, \cdots, T_n$, denoted* $\texttt{NtLCA}(T_1, \cdots, T_n)$, *is a node type V such that (1) $V \prec T_i$ for all $i \in [1, n]$; (2) there is no node type U such that $V \prec U \prec T_i$ for all $i \in [1, n]$.*

We now define word type, which is used to represent the meaning of a word.

Definition 5. (**Word Type**) *Let t be the data tree and w be a word that occurs in t. A node type is said to be a word type of w if some nodes of this node type* directly *contain w.*

Table 1 lists the word types of several words for the data tree in Fig. 1.

Table 1. Sample Word Types

Word	No.	Word Type
issue	1	`SigmodRecord.issue`
	2	`SigmodRecord.issue.articles.article.title`
16	3	`SigmodRecord.issue.volume`
	4	`SigmodRecord.issue.articles.article.initPage`
database	5	`SigmodRecord.issue.articles.article.title`
	6	`SigmodRecord.issue.articles.article.authors.author`

Definition 6. (**Search Intention**) *Given a keyword query $K = \{w_1, \cdots, w_n\}$ and their corresponding word type sets $\{WT_1, \cdots, WT_n\}$, a search intention of this query is a tuple of word types $(wt_1, \cdots, wt_n)$, where $wt_i \in WT_i$ $(1 \leq i \leq n)$.*

Once we have a search intention, we can find the result subtrees accordingly. The details of the result subtree will be given in Section 5.

3.2 Inferring the Word Types of Two Adjacent Keywords

If two adjacent keywords w_i and w_{i+1} are taken as a keyword query with two keywords, the number of possible search intentions is equal to $|WT_i| \times |WT_{i+1}|$. We design a formula to compute the *proximity score* between wt_i and wt_{i+1} which is used to evaluate how closely wt_i and wt_{i+1} are related, where $(wt_i, wt_{i+1}) \in WT_i \times WT_{i+1}$. The tuple (wt_s, wt_t) that achieves the *largest proximity score* is considered as the desired search intention of w_i and w_{i+1}.

Basically, the proximity score between two word types is affected by the distance between them and the statistics of them.

Distance between Two Word Types. The distance between two word types wt_1 and wt_2 is defined as the total number of edges from $\texttt{NtLCA}(wt_1, wt_2)$ to the ends of wt_1 and wt_2, which can be calculated using the following formula:

$$Dis(wt_1, wt_2) = len(wt_1) + len(wt_2) - 2 * clen(wt_1, wt_2) \qquad (1)$$

In the formula above, len is the length of a word type. $clen(wt_1, wt_2)$ is the length of the longest common ancestor of wt_1 and wt_2. Consider word types

2 and 6 in Table 1. According to Formula (1), the distance between them is $5 + 6 - 2 * 4 = 3$.

Intuitively, *the shorter the distance between two word types, the more closely these two word types are related.* In our work, we use $p^{Dis(wt_1,wt_2)}$ to formulate the intuition above, where p is a tuning parameter which is used to determine how much penalty should be given to the distance between two word types. In our experiments we found setting p to 0.87 achieves good results.

Statistics of Word Types. Another important factor that influences the proximity score is the statistics of word types. Given a pair of word types of two adjacent keywords, we formulate the influencing factor of the statistics of the two word types as follows:

$$Sta(wt_1, wt_2) = log_e(f_{w_1,wt_1}^{\mathtt{NtLCA}(wt_1,wt_2)} + f_{w_2,wt_2}^{\mathtt{NtLCA}(wt_1,wt_2)}) * R \tag{2}$$

where R is a reduction factor which will be explained shortly. $f_{w,wt}^{T}$ is the number of T-typed nodes that contain the keyword w with the word type wt in their subtrees. The intuition (similar to XReal) behind the statistics is as follows:

The more XML *nodes of node type* $\mathtt{NtLCA}(wt_1, wt_2)$ *contain the keywords with word types* wt_1 *and* wt_2*, the more likely these two word types are related through the nodes of node type* $\mathtt{NtLCA}(wt_1, wt_2)$ *and this kind of relationship between* wt_1 *and* wt_2 *is desired by the user.*

We now explain the reduction factor R. Due to the tree structure, the number of nodes at higher levels is usually significantly less than the number of nodes at lower levels. This may bring unfairness when collecting statistics. Fewer nodes at higher levels are usually caused by design, which may not reflect the real occurrences of data. Therefore, we put a reduction factor of depth to the formula to reduce the unfairness. A straightforward reduction factor of depth is $\frac{1}{Dep(\mathtt{NtLCA}(wt_1,wt_2))}$, but it reduces too much and too quickly when the depth increases. Instead, we use the following formula as the reduction factor:

$$R = \sqrt{\frac{1}{Dep(\mathtt{NtLCA}(wt_1, wt_2))}} \tag{3}$$

With the two influencing factors above, the proximity score between two word types is defined as follows:

$$P(wt_1, wt_2) = p^{Dis(wt_1,wt_2)} * Sta(wt_1, wt_2) \tag{4}$$

Desired Word Types. Given two adjacent keywords w_i and w_{i+1}, the preferred relationship between them achieves the largest proximity score among all $\mathtt{P}(wt_i, wt_{i+1})$, where $wt_i \in WT_i$ and $wt_{i+1} \in WT_{i+1}$. The word types that form this relationship are the desired word types of w_i and w_{i+1}. In the sequel, the largest proximity score between w_i and w_{i+1} will be denoted $\mathtt{HP}(w_i, w_{i+1})$, i.e.,

$$\mathtt{HP}(w_i, w_{i+1}) = \max\{\mathtt{P}(wt_i, wt_{i+1}) \mid wt_i \in WT_i, wt_{i+1} \in WT_{i+1}\} \tag{5}$$

4 Inferring Likely Search Intentions

4.1 One Keyword

If a query contains *only one* keyword, each word type of this keyword is considered as a search intention. We use the following formula to compute the likelihood that the word type wt is desired by the user.

$$\mathtt{C}(wt) = log_e(f_{k,wt}^{wt}) * \sqrt{\frac{1}{Dep(wt)}} \tag{6}$$

4.2 Two or More Keywords

When there are two or more keywords, we use a *pair-wise comparison strategy* (PCS) to infer a set of likely search intentions. We first explain the ideas and then present the detailed algorithms.

Inferring One Likely Search Intention. Given the keywords $\{w_1, \cdots, w_n\}$ ($n > 1$) and their corresponding word type sets $\{WT_1, \cdots, WT_n\}$, we infer a likely search intention as follows.

In the case $n = 2$, i.e., there are only two keywords w_1 and w_2 in the query, we will compute the largest proximity score between w_1 and w_2 (i.e., $\mathtt{HP}(w_1, w_2)$) using formula (5), and choose the word types of w_1 and w_2 that achieve $\mathtt{HP}(w_1, w_2)$ as the word types of w_1 and w_2.

In the case $n > 2$, we need to go through several iterations. In the first iteration, we scan the keywords from left to right pair by pair and compute $\mathtt{HP}(w_i, w_{i+1})$ ($1 \leq i < n$). We group w_j and w_{j+1} together if they achieve the largest HP value, that is, if $\mathtt{HP}(w_j, w_{j+1}) = max\{\mathtt{HP}(w_i, w_{i+1}) \mid i \in [1, n-1]\}$. Whenever a pair of adjacent keywords w_i, w_{i+1} are put into a group, the word types of w_i and w_{i+1} that achieve $\mathtt{HP}(w_i, w_{i+1})$ will be chosen as the word types of w_i and w_{i+1} respectively, and we will go to the next iteration. In each subsequent iteration, we scan the keywords or groups of keywords from left to right, grouping a pair of two keywords, or a keyword and a group, into the same group by using the largest proximity score in a way similar to the first iteration, except that for each group (that contains more than one keyword), a unique word type of each keyword in the group has been chosen as the word type for that keyword (and this unique word type will not be changed later), thus the computation of the largest proximity score between a keyword and a group, or two groups will use the word types of each keyword in the group as shown in formulae 7 to 8 below. This process continues until all keywords joins a group (so that its word type can be determined).

For a keyword w_i and a group $g = \{w_k, \ldots, w_s\}$ of keywords, suppose the word types of $w_k, \cdots, w_s$ are $wt_k, \cdots, wt_s$ respectively, then

$$\mathtt{HP}(w_i, g) = \mathtt{HP}(g, w_i) = \max\{\mathtt{P}(wt_i, wt) \mid wt_i \in WT_i, wt \in \{wt_k, \ldots, wt_s\}\} \tag{7}$$

For two neighboring groups $g_1 = \{w_j, \ldots, w_k\}$ and $g_2 = \{w_{k+1}, \ldots, w_t\}$ with word types $wt_j, \ldots, wt_k$ and $wt_{k+1}, \ldots, wt_t$ respectively, we define

$$\texttt{HP}(g_1, g_2) = P(wt_k, wt_{k+1}) \tag{8}$$

The HP score between two groups is not used to merge two groups. It is only used in the ranking of the returned search intentions, as we will discuss later.

Inferring a Set of Likely Search Intentions. To reduce the possibility of missing the real search intention, we generate a set of likely search intentions as described below.

Given the keywords $\{w_1, \ldots, w_n\}$ and their word type sets $\{WT_1, \ldots, WT_n\}$, we treat each word type in WT_i as the only word type of w_i, and use PCS to infer a search intention which is considered as a *likely search intention*. In total, we will produce $\sum_{1 \leq i \leq n} |WT_i|$ likely search intentions if there are no duplicates[3], which is much smaller than $\prod_{1 \leq i \leq n} |WT_i|$ (the number of all possible search intentions). Applying PCS against each word type guarantees a good coverage of different word types in the query. In other words, every word type of each keyword exists in at least one likely search intention.

Ranking Likely Search Intentions. In order to rank the inferred likely search intentions, we define their ranking scores as follows.

Definition 7. *(***Ranking Score***) Given a query $K = \{w_1, \cdots, w_n\}$ $(n \geq 2)$ and a search intention I inferred by* PCS*, the* ranking score *of I is defined as a vector $(r_1, \cdots, r_{n-1})$, where $r_i \geq r_{i+1}$ $(1 \leq i < n-1)$, and r_i is a* largest proximity score *between two keywords or between a keyword and a group that are grouped together in* PCS *(calculated by formulae (5) to (7)), or the largest proximity score between two neighboring groups (calculated by Formula (8)). Given two ranking scores A and B, $A = (a_1, \cdots, a_{n-1})$ is smaller than $B = (b_1, \cdots, b_n)$ iff the first a_i which is different from b_i is smaller than b_i.*

Example 1. Suppose the keywords in the query $\{w_1\ w_2\ w_3\ w_4\}$ are grouped as $\{(w_1, w_2), (w_3, w_4)\}$ and the inferred search intention is $I = (wt_1, wt_2, wt_3, wt_4)$. The ranking score of I is a sequence of proximity scores $\texttt{HP}(w_1, w_2)$, $\texttt{HP}(w_3, w_4)$ and $\texttt{P}(wt_2, wt_3)$, which will be sorted in descending order. The ranking score $(2.5, 2, 1.2)$ is larger than $(2.5, 1.8, 1)$.

Note that we use $(r_1, \cdots, r_{n-1})$ rather than $\sum_i r_i$ as the ranking score. The reason for this will become clear if one considers the data tree in Fig. 1 and the query {issue 16 database}. The correct search intention (1, 3, 5) (where 1,3, 5 refer to the word types in Table 3.1) will be ranked higher than the intention (2,4, 5) using our ranking scheme. However, if we use $\sum_i r_i$ as the ranking score, then the intention (1, 3, 5) will be ranked lower than (2,4,5).

[3] Note that there may exist duplicates in the inferred likely search intentions.

Algorithm 1. PCS (K,WT)

```
Input: Query $K = \{w_1, \cdots, w_n\}$, the corresponding word type sets $\{WT_1, \cdots, WT_n\}$
Output: a search intention: list $l = (l_1, \ldots, l_n)$ and it ranking vector $r$

1: let $l = (null, \ldots, null)$; $r = \emptyset$
2: while ExistUngroupedKeyword() do
3:     let $j = 1$; $k = 1 + w_1$; $largestScore = 0$
4:     while $k \leq n$ do
5:         if $not(w_j.num > 1 \wedge w_k.num > 1)$ then
6:             $(score_{j,k}, wt_j, wt_k) =$ computeHPscore$(w_j, w_k)$
7:             if $score_{j,k} > largestScore$ then
8:                 let $largestScore = score_{j,k}$; $s = j$; $t = k$
9:         let $j = k$; $k = j + w_j.num$
10:    group$(w_s, w_t)$
11:    add $largestScore$ to $r$
12:    let $l_s = wt_s$ if $w_s.num = 1$; let $l_t = wt_t$ if $w_t.num = 1$
13: add to $r$ the HP scores between neighboring groups, then sort $r$
```

Algorithms. We implemented PCS in Algorithm 1. Before explaining this algorithm, we first present some notation and functions used in the algorithm.

Notation. For each keyword w_i, we use an attribute $w_i.num$ to record the number of keywords in the group that w_i sits. If w_i is not grouped with others, $w_i.num = 1$. The function group(w_j, w_k) puts w_j and w_k into the same group if they are not grouped yet, or puts one into the same group as the other (if only one of them is grouped), and while doing this it also sets $w_i.num$ to $w_j.num + w_k.num$ for every keyword w_i in the group. The function computeHPscore(w_j, w_k) returns a score, which is the HP value calculated using formula 5 if neither w_j nor w_k is grouped with others, or calculated using formula 7 if only one of w_j, w_k is grouped with others. The corresponding word types of w_j, w_k that achieve the HP value are also returned as wt_j and wt_k respectively. The function ExistUngroupedKeyword() returns **true** iff there is a keyword in K that has not been grouped with others (A linear scan of each keyword in K, checking whether $w_i.num = 1$, will do).

Now we explain Algorithm 1. Line 1 initializes l and r. If there exists a keyword which is not grouped with others, lines 3 to 9 will compute the HP value between two keywords or between a keyword and a group, and find a pair of keywords (or a keyword and a group), represented by w_s and w_t, that achieves the largest HP value. Line 10 groups the pair together. The largest HP value is then added to r (line 11), and if w_s is not grouped before , we set its word type to wt_s, the word type in WT_s that achieves the above largest HP value (line 12). The same is done for w_t. Finally, line 13 computes the HP values between neighboring groups and add them to the ranking score r, and then sorts r.

Example 2. Suppose the user submits the query {*volume 11 article Karen*} over the real SigmodRecord data set obtained from [1]. In the first run of lines 3 to 12, HP$(volume, 11)$, HP$(11, article)$ and HP$(article, Karen)$ are computed, and because HP$(volume, 11)$ is greater than HP$(11, article)$ and HP$(article, Karen)$, we group *volume* and *11* together. Since HP$(volume, 11)$ is achieved by the word types *SimmodRecord.issue.volume* (for *volume*) and *SimmodRecord.issue.volume* (for *11*), these word types are chosen as the word types of *volume* and *11* (line 12), and $\texttt{HP}(volume, 11)$ is added into r. Since there are still keywords that are not grouped, lines 2 to 12 will execute again. This time, we compute $\texttt{HP}((volume, 11), article)$ and $\texttt{HP}(article, Karen)$, and since $\texttt{HP}(article, Karen)$ is larger and it is achieved by the word types *SigmodRecord.issue.articles.article* and *SigmodRecord.issue.articles.article.authors.author*. Thus we group *article* and *Karen* together and set their word types to the aforementioned word types. We add $\texttt{HP}(article, Karen)$ to r. Now every keyword is in a group (and thus has a word type chosen), the outer loop (lines 2 to 12) stops. Finally we add $P(SimmodRecord.issue.volume, SigmodRecord.issue.articles.article)$ into r and then sort r.

Time complexity. Given a keyword query $K = \{w_1, \cdots, w_n\}$, the worst case time complexity of Algorithm 1 is $O(n^2|WT_1||WT_2|)$, where WT_1 and WT_2 are the sets of word types of the two keywords which have the most word types. The detailed analysis of time complexity is as follows: Computing the HP score between keywords w_i and w_{i+1} needs $O(|WT_i||WT_{i+1}|)$. Computing the HP score between a keyword w_i and a group g needs $O(|WT_i||g|)$ (note that $|g| < n$). The loop (lines 2 to 12) runs at most n times, and each time, the HP score will be computed for at most $n-1$ pairs of keywords (and/or groups). The function `existUngroupedKeyword()` is also in $O(n)$. Computing the HP values between neighboring groups takes $O(n)$, and sorting r takes $O(n^2)$. Thus the algorithm takes $O(n^2|WT_1||WT_2|)$.

Algorithm for Inferring a Set of Likely Search Intentions. The algorithm for inferring the set of likely search intentions is rather simple. It simply calls PCS repeatedly, each time it chooses one keyword w_i and uses one word type as the word type set of w_i. The detailed steps are omitted in order to save space.

5 Generating Results

In this section, we explain how to retrieve result subtrees for a set of likely search intentions. Before defining result subtree, we need to define entity types of a node type.

Definition 8. (**Entity-type of a Node Type**) *If a node type T is the node type of some entity node, its entity-type is itself; otherwise, its entity-type is its ancestor node type T' such that (1) T' is the node type of some entity node, and (2) T' is the longest among all ancestor node types of T satisfying condition (1).*

Note that every node type in the data tree owns one and only one entity-type. For example, in Fig. 1, the entity-type of node type *initPage* is the node type *article*. Node type *article*'s entity-type is itself.

The result subtree is defined as follows.

Definition 9. *(***Result Subtree***) Given a keyword query $K = \{w_1, \cdots, w_n\}$ and a search intention $\{wt_1, \cdots, wt_n\}$, a subtree of t is a* result subtree *iff: (1) its root has the node type $\texttt{ENtLCA}(wt_1, \cdots, wt_n)$, (2) it contains all of the keywords in K, and at least one of the occurrences of keyword w_i has the word type wt_i.*

Algorithm 2. GenerateResults(L)

```
Input: A set of likely search intentions $L$
Output: result subtrees
1: for each $l$ in $L$ do
2:     $rt = GetNodeTypeofRoot(l)$ // $rt$: node type of the root
3:     $rl = RetrieveInvertedList(root)$
4:     for $1 \leq i \leq |l|$ do
5:         $IL_i = RetrieveInvertedList(l_i)$
6:     while $rl.isEnd() = false$ do
7:         for $1 \leq i \leq |l|$ do
8:             $IL_i.Moveto(rl.getCurrent())$
9:             if $isAncestor(rl.getCurrent(), IL_i.getCurrent()) == false$ then
10:                break
11:        if $i > |l|$ then
12:            $resultList.insert(rl.getCurrent())$
13:        $rl.movetoNext()$
14: Build result subtrees rooted at the nodes in $resultList$ and exclude irrelevant entities.
15: Return result subtrees to the user
```

The purpose of using `ENtLCA` instead of `NtLCA` in the definition above is to make the returned result subtrees more informative and meaningful.

Excluding Irrelevant Entities. Sometimes, a retrieved result subtree may contain lots of irrelevant information. Consider the query {issue 16 database}. Suppose our approach is to return the result subtrees rooted at *issue* nodes. However, if we return the whole subtrees rooted at *issue* nodes, a number of articles that are not related to database are also returned to the user. Therefore, we should exclude these irrelevant entities. In order to achieve this goal, we first find the entity-type of each keyword's word type, then we know the keywords and their word types that an entity should contain. If an entity does not contain the keywords with the inferred word types, it will be excluded. In the example above, an article entity should contain the keyword "database" with the word type No.5 in Table 1. The entities that do not satisfy this condition will be excluded.

Algorithm. The algorithm for generating results is shown in Algorithm 2. We refer the readers to the full version of the paper for a detailed explanation of the algorithm.

6 Experiments

In this section, we present the experimental results on the effectiveness of our approach against `XReal` [2], `XBridge` [3] and `MaxMatch` [5]. `XReal` and `XBridge` are the most up-to-date XML keyword search system which utilize statistics of data to infer the major search intention of a query. Comparisons of these systems with `XSeek` can be found in [2] and [3].

6.1 Experimental Setup

We implemented `XReal`, `XBridge` and our system `XInfer` in C++. All the experiments were performed on an Intel Pentium-M 1.7G laptop with 1G RAM.

Note that XBridge only provides information on how to suggest the promising result types, so we extend XBridge with the part of generating result subtrees. Similar to XReal, the subtrees that are rooted at the nodes of the suggested result types and contain all of the keywords are considered as the result subtrees. The executable file of MaxMatch was kindly provided by its authors. We used the following three data sets that are obtained from [1] for evaluation:

DBLP: The structure of this data set is wide and shallow. It has many different types of entities. Many words in this data set have multiple word types.
SigmodRecord: This data set has a little more complicated structure than DBLP, but has less entity types. Fig. 1 provides a similar sample.
WSU: Similar to DBLP, the structure of this data set is also wide and shallow. However, it has only one entity type and many words have only one word type.

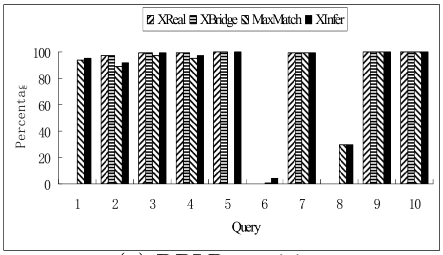
The queries we use for evaluation are listed in Table 2. These queries were chosen by three student users who were given the data sets.

Table 2. Queries

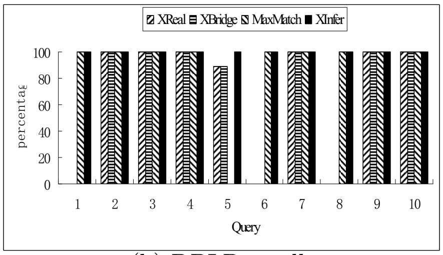
dataset	ID	Query
DBLP	QD1	{Automated Software Engineering}
	QD2	{Han data mining}
	QD3	{WISE 2000}
	QD4	{Jeffrey XML}
	QD5	{author Jim Gray}
	QD6	{Relational Database Theory}
	QD7	{article spatial database}
	QD8	{Wise database}
	QD9	{Han VLDB 2000}
	QD10	{twig pattern matching}
SigmodRecord	QS1	{Database Design}
	QS2	{title XML}
	QS3	{article Database}
	QS4	{Karen Ward}
	QS5	{author Karen Ward}
	QS6	{volume 11 database}
	QS7	{issue 11 database}
	QS8	{Anthony 11}
	QS9	{Anthony issue 11}
	QS10	{issue 21 article semantics author Jennifer}
WSU	QW1	{CAC 101}
	QW2	{title ECON}
	QW3	{instructor MCELDOWNEY}
	QW4	{FINITE MATH}
	QW5	{prefix MATH}
	QW6	{place TODD}
	QW7	{COST ACCT enrolled}
	QW8	{CELL BIOLOGY times}
	QW9	{ECON days times place}
	QW10	{prefix ACCTG instructor credit}

6.2 Effectiveness

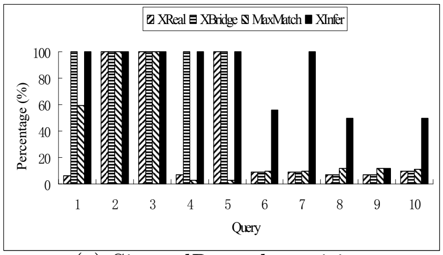
We conducted a user survey on the search intentions of the queries in Table 2. 19 graduate students participated in the survey. We used the search intentions selected by the majority of people to determine the relevant matches. We evaluate



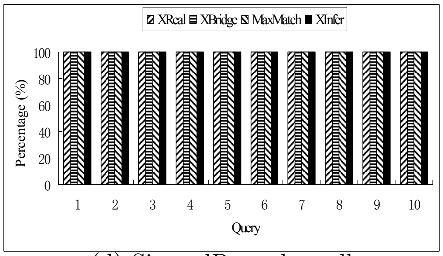
(a) DBLP precision



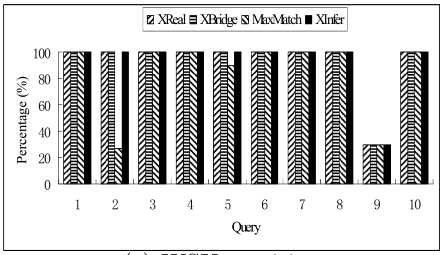
(b) DBLP recall



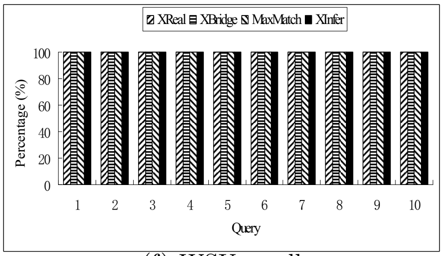
(c) SigmodRecord precision



(d) SigmodRecord recall



(e) WSU precision



(f) WSU recall

Fig. 2. Precision and recall

the effectiveness of `XReal`, `XBridge`, `MaxMatch` and `XInfer` based on *precision*, *recall* and *F-measure*. Precision is the percentage of retrieved results that are desired by users. Recall is the percentage of relevant results that can be retrieved. F-measure is the weighted harmonic mean of precision and recall.

As shown in Fig. 2(a), `XReal` and `XBridge` have a very low precision on the queries QD1, QD6 and QD8. This is mainly because `XReal` and `XBridge` infer undesired Search-for Node Types (`SNTs`). For these queries, `XReal` and `XBridge` infer the same `SNTs`, even though they use different strategies. Suppose the user submits QD1 to retrieve the articles from the journal of Automated Software Engineering. `XReal` and `XBridge` just return the *inproceedings* that are related to the automated software engineering, but both `MaxMatch` and `XInfer` return the articles from the journal of Automated Software Engineering besides the inproceedings about automated software engineering. Suppose the user submits the query QD8 to retrieve the publications written by Wise. `XReal` and `XBridge`

Table 3. Comparison on F-Measure

F-Measure	XReal	XBridge	MaxMatch	XInfer
DBLP	0.69	0.69	0.79	0.90
SigmodRecord	0.51	0.65	0.48	0.88
WSU	0.96	0.96	0.91	0.96

return the *inproceedings* of WISE conference. MaxMatch and XInfer return the inproceedings of WISE conference as well as Wise's publications. Suppose the user wants to search the book called Relational Database Theory and submits the query QD6. XReal and XBridge does not return this book. MaxMatch and XInfer return this book but have low precisions because they return much irrelevant information at the same time (e.g., the inproceedings about relational database theory, etc). On Query QD2-QD4, XReal and XBridge achieve a little higher precisions than MaxMatch and XInfer because XReal and XBridge just infer one search-for node type which reduces the irrelevant information in the results. Actually this is not a serious problem for XInfer because XInfer rank the desired search intention as the top-1 search intention, so it is very easy for the user to find their desired results. Suppose the user wants to retrieve the publications written by Jim Gray and submits query QD5. XReal, XBridge and XInfer return the desired results, but MaxMatch just returns the author nodes, which means the returned information is too limited. Therefore, MaxMatch has a very low precision on this query. Fig. 2(b) presents the recalls of XReal, XBridge, MaxMatch and XInfer on the query QD1-QD10. XReal and XBridge have very low recalls on the query QD1, QD6 and QD8 because they do not return the relevant results as we explained above. MaxMatch has a very low recall on the query QD5 because it just returns the subtrees rooted at *author* nodes.

As shown in Fig. 2(c), XInfer achieves higher precision than XReal, XBridge and MaxMatch for the SigmodRecord dataset. For the queries QS1 and QS4, XReal shows low precision mainly because it infers *issue* as the search-for node type for these two queries, which results in many irrelevant articles being returned to the user. For example, according to the survey, the user intends to retrieve articles about database design with the query QS1. However, lots of articles that are not related to database design are also returned to the user in XReal. XBridge and XInfer returns the articles about database design for QS1 and the articles written by Karen Ward for QS4, which are desired by the user. XInfer, XReal and MaxMatch achieve good precisions on the queries QS2 and QS3. MaxMatch gets very low precision on the queries QS4 and QS5. Most participants think the user wants the articles written by Karen Ward with the queries QS4 or QS5, but MaxMatch only returns *author* nodes, which do not provide much information desired by the user. In order to retrieve the articles about database from the issues of volume 11, the user submits the query QS6 or QS7. XReal and XBridge return lots of irrelevant articles (including the articles that are not about database, the database articles whose initPage is 11, etc) to the user. XInfer infers five search intentions on QS6 and one search intention on QS7. Compared with QS6, QS7 adds a new keyword "issue" which is used to specify the meaning of "11". XInfer notices this difference, and correctly infers the user's search intention. For the recall, as shown in Fig. 2(d), all of these three approaches present good recalls.

For data set WSU, Fig. 2(e) shows that all of these approaches generally achieve good precision. This is mainly because WSU has a simple and shallow structure compared with the data set SigmodRecord. For the query QW9, they present a relatively low precision. The user intends to retrieve the days, times and place of the courses whose titles contain "ECON", but the systems return the courses whose prefix contain "ECON" as well and give them the highest ranks because most words "ECON" appear in the *prefix* nodes. MaxMatch has a low precision on query QW2 because it also returns the courses whose prefix contain "ECON" even though the user adds a describing word "title". As shown in Fig. 2(f), all of the four approaches present good recalls.

We calculated the average F-measure of the queries over each data set and they are listed in Table 3. It can be seen that XInfer achieves higher F-measure than XReal, XBridge and MaxMatch over all three datasets.

Other Metrics. More experimental results are provided in the full version of this paper. These include the categorized efficiency, scalability of efficiency, index structures and index size, the number of returned likely search intentions, ranking effectiveness, and the effect of individual factors (distance, statistics) on the search quality.

7 Conclusion

In this paper, we presented a method to improve the effectiveness of XML keyword search by exploiting the relationship between different keywords in a query. We proposed the Pair-wise Comparison Strategy to infer and rank a set of likely search intentions. We developed an XML keyword search system called XInfer which realizes the techniques we propose. The better search quality of XInfer was verified by our experiments.

References

1. http://www.cs.washington.edu/research/xmldatasets
2. Bao, Z., Ling, T.W., Chen, B., Lu, J.: Effective XML keyword search with relevance oriented ranking. In: ICDE, pp. 517–528 (2009)
3. Li, J., Liu, C., Zhou, R., Wang, W.: Suggestion of promising result types for XML keyword search. In: EDBT, pp. 561–572 (2010)
4. Liu, Z., Chen, Y.: Identifying meaningful return information for XML keyword search. In: SIGMOD Conference, pp. 329–340 (2007)
5. Liu, Z., Chen, Y.: Reasoning and identifying relevant matches for XML keyword search. PVLDB 1(1), 921–932 (2008)
6. Liu, Z., Chen, Y.: Processing keyword search on XML: a survey. World Wide Web 14(5-6), 671–707 (2011)