IEEE International Conference on Data Engineering (ICDE2009):
517-528, March 29- April 3, 2009, Shanghai, China

Effective XML Keyword Search with Relevance Oriented Ranking

Zhifeng Bao #, Tok Wang Ling #, Bo Chen #
#*School of Computing*
National University of Singapore
{baozhife, lingtw, chenbo}@comp.nus.edu.sg

Jiaheng Lu *
**School of Information and DEKE, MOE*
Renmin University of China
jiahenglu@gmail.com

***Abstract*— Inspired by the great success of information retrieval (IR) style keyword search on the web, keyword search on XML has emerged recently. The difference between text database and XML database results in three new challenges: (1) Identify the user search intention, i.e. identify the XML node types that user wants to search for and search via. (2) Resolve keyword ambiguity problems: a keyword can appear as both a tag name and a text value of some node; a keyword can appear as the text values of different XML node types and carry different meanings. (3) As the search results are sub-trees of the XML document, new scoring function is needed to estimate its relevance to a given query. However, existing methods cannot resolve these challenges, thus return low result quality in term of query relevance.**

In this paper, we propose an IR-style approach which basically utilizes the statistics of underlying XML data to address these challenges. We first propose specific guidelines that a search engine should meet in both search intention identification and relevance oriented ranking for search results. Then based on these guidelines, we design novel formulae to identify the search for nodes and search via nodes of a query, and present a novel XML TF*IDF ranking strategy to rank the individual matches of all possible search intentions. Lastly, the proposed techniques are implemented in an XML keyword search engine called XReal, and extensive experiments show the effectiveness of our approach.

I. Introduction

The extreme success of web search engines makes keyword search the most popular search model for ordinary users. As XML is becoming a standard in data representation, it is desirable to support keyword search in XML database. It is a user friendly way to query XML databases since it allows users to pose queries without the knowledge of complex query languages and the database schema.

Effectiveness in term of result relevance is the most crucial part in keyword search, which can be summarized as the following three issues in XML field.

Issue 1: It should be able to effectively identify the type of target node(s) that a keyword query intends to search for. We call such target node as a *search for node*.

Issue 2: It should be able to effectively infer the types of condition nodes that a keyword query intends to search via. We call such condition nodes as *search via nodes*.

Issue 3: It should be able to rank each query result in consideration of the above two issues.

The first two issues address the search intention problem, while the third one addresses the relevance based ranking problem w.r.t. the search intention. Regarding to Issue 1 and Issue 2, XML keyword queries usually have ambiguities in interpreting the search for node(s) and search via node(s), due to two reasons below.

- **Ambiguity 1**: A keyword can appear both as an XML tag name and as a text value of some other nodes.
- **Ambiguity 2**: A keyword can appear as the text values of different types of XML nodes and carry different meanings.

For example see the XML document in Figure 1, keywords *customer* and *interest* appear as both an XML tag name and a text value (e.g. value of the title for book B1), and *art* appears as a text value of interest, address and name node.

Regarding to Issue 3, the search intention for a keyword query is not easy to determine and can be ambiguous, because the search via condition is not unique; so how to measure the confidence of each search intention candidate, and rank the individual matches of all these candidates are challenging.

Although many research efforts have been conducted in XML keyword search [1], [2], [3], [4], [5], none of them has addressed and resolved the above three issues yet. For instance, one widely adopted approach so far is to find the smallest lowest common ancestor (SLCA) of all keywords [3]. Each SLCA result of a keyword query contains all query keywords but has no subtree which also contains all the keywords. Since [4], [5] etc. are variations of SLCA, we use SLCA as a typical existing approach in the rest discussion. Those SLCA-based approaches only take the tree structure of XML data into consideration, without considering the semantics of the query and XML data.

In particular, regarding to Issue 1 and 2, SLCA may introduce answers that are either irrelevant to user search intention, or answers that may not be meaningful or informative enough. E.g. when a query "Jim Gray" that intends to find Jim Gray's publications on DBLP [6] is issued, SLCA returns only the *author* elements containing both keywords. Besides, SLCA also returns publications written by two authors where "Jim" is a term in 1st author's name and "Gray" is a term in 2nd author, and publications with *title* containing both keywords. It is reasonable to return such results because search intention may not be unique; however they should be given a lower rank, as they are not matches of the major search intention. Regarding to Issue 3, no existing approach has studied the problem of relevance oriented result ranking in depth yet. Moreover, they don't perform well on pure keyword query

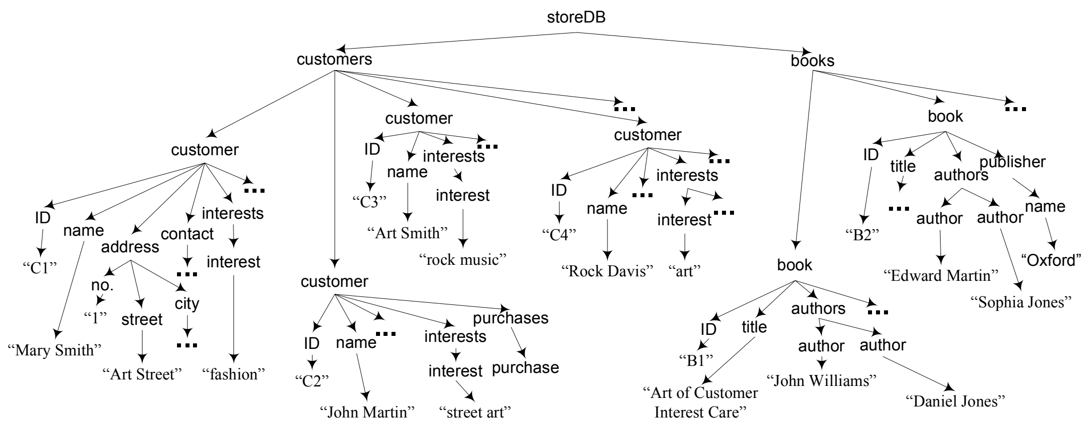


Fig. 1. Portion of data tree for an online bookstore XML database

when the schema information of XML data is not available [4]. The actual reason is, none of them can solve the keyword ambiguity problems, i.e. Ambiguity 1 and Ambiguity 2, as demonstrated by the following example.

Example 1: Consider a keyword query "*customer interest art*" issued on the bookstore data in Figure 1, and most likely it intends to find the customers who are interested in art.

If adopting SLCA, we will get 5 results, which include the title of book B1 and the customer nodes with IDs from C1 to C4 (as these four customer nodes contain "customer", "interest" and "art" in either the tag names or node values) in Figure 1. Since SLCA cannot well address the search intention, all these 5 SLCA results are returned without any ranking applied. However, only C4 is desired which should be put as the top ranked one, and C2 is less relevant, as his interest is "street art" rather than "art", while C1 and C3 are irrelevant.□

Inspired by the great success of IR approach on web search (especially its distinguished ranking functionality), we aim to achieve similar success on XML keyword search, to solve the above three issues without using any schema knowledge. The main challenge we are going to solve is how to extend the keyword search techniques in text databases (IR) to XML databases, because the two types of databases are different. *First*, the basic data units in text databases searched by users are flat documents. For a given query, IR systems compute a numeric score for each document and rank the document by this score. In XML databases, however, information is stored in hierarchical tree structures. The logical unit of answers needed by users is not limited to individual leaf nodes containing keywords, but a subtree instead. *Second*, unlike text database, it is difficult to identify the (major) user search intention in XML data, especially when the keywords contain ambiguities mentioned before. *Third*, effective ranking is a key factor for the success of keyword search. There may be dozens of candidate answers for an ordinary keyword query in a medium-sized database. E.g. in Example 1, five subtrees can be the query answers, but they are not equally useful to user. Due to the difference in basic answer unit between document search and database search, in XML database we need to assign a single ranking score for each subtree of certain category with a fitting size, in order to rank the answers effectively.

Statistics is a mathematical science pertaining to the collection, analysis, interpretation or explanation of data; it can be used to objectively *model a pattern* or *draw inferences* about the underlying data being studied. Although keyword search is a subjective problem that different people may have different interpretations on the same keyword query, statistics provides an objective way to distinguish the major search intention(s).

This motivates us to design a best efforts heuristic approach that provides an objective way to measure the query result relevance; thus we model the search engine as a domain expert who automatically interprets user's all possible search intention(s) through analyzing the statistics knowledge of underlying data. In this paper we propose a novel IR-style approach which well captures XML's hierarchical structure, and works well on pure keyword query independent of any schema information of XML data. In particular, the original TF*IDF similarity [7] is extended to handle both semi-structured and unstructured data, and a keyword search system prototype called XReal is implemented to achieve effective identification of user search intention and relevance oriented ranking for the search results in the presence of keyword ambiguities.

Example 2: We use the query in Example 1 again to explain how XReal infers user's desired result and puts it as a top-ranked answer. XReal interprets that user desires to search for customer nodes, because all three keywords have high frequency of occurrences in customer nodes. Similarly, since keywords "interest" and "art" have high frequency of occurrences in subtrees rooted at interest nodes, it is considered with high confidence that this query wants to search via interest nodes, and incorporate this confidence into our ranking formula. Besides, customers interested in "art" should be ranked before those interested in (say) "street art". As a result, C4 is ranked before C2, and further before customers with address in "art street"(e.g. C1) or named "art" (e.g. C3).□

To our best knowledge, we are the first that exploit the statistics of underlying XML database to address search intention identification, result retrieval and relevance oriented ranking as a single problem for XML keyword search. Our main contributions are summarized as follows:

1) This is the first work that addresses the keyword ambiguity problem. We also identify three crucial issues that an effective XML keyword search engine should meet.
2) We define our own XML TF (term frequency) and XML DF (document frequency), which are cornerstones of all formulae proposed later.
3) We propose three important guidelines in identifying the user desired *search for* node type, and design a formula to compute the confidence level of a certain node type to be a desired *search for* node based on the guidelines.
4) We design formulae to compute the confidence of each candidate node type as the desired *search via* node to model natural human intuitions, in which we take into account the pattern of keywords co-occurrence in query.
5) We propose a novel relevance oriented ranking scheme called *XML TF*IDF similarity* which can capture the hierarchical structure of XML, and resolve Ambiguity 1 and Ambiguity 2 in a heuristic way; and also distinguish the similarity computation for leaf nodes and internal nodes in XML data. Moreover, our approach is able to handle both semi-structured and unstructured data.
6) We implement the proposed techniques in a keyword search engine prototype called XReal. Extensive experiments show its effectiveness, efficiency and scalability.

The rest of the paper is organized as follows. We present the related work in Section II, and preliminary on IR and data model in Section III. Section IV infers user search intention, and Section V discusses relevance oriented ranking. Section VI presents the search algorithms. Experimental evaluation is given in Section VII and we conclude in Section VIII.

II. Related Work

Extensive research efforts have been conducted in XML keyword search to find the smallest sub-structures in XML data that each contains all query keywords in either the tree data model or the directed graph (i.e. digraph) data model.

In tree data model, LCA (lowest common ancestor) semantics is first proposed and studied in [8], [2] to find XML nodes, each of which contains all query keywords within its subtree. Subsequently, SLCA (smallest LCA [9], [3]) is proposed to find the smallest LCAs that do not contain other LCAs in their subtrees. GDMCT (minimum connecting trees) [5] excludes the subtrees rooted at the LCAs that do not contain query keywords. Sun et al. [10] generalize SLCA to support keyword search involving combinations of AND and OR boolean operators. XSeek [4] generates the return nodes which can be explicitly inferred by keyword match pattern and the concept of entities in XML data. However, it addresses neither the ranking problem nor the keyword ambiguity problem. Besides, it relies on the concept of entity (i.e. object class) and considers a node type t in DTD as an entity if t is "*"-annotated in DTD. As a result, customer, phone, interest, book in Figure 1, are identified as object classes by XSeek. However, it causes the multi-valued attribute to be mistakenly identified as an entity, causing the inferred return node not as intuitive as possible. E.g. phone and interest are not intuitive as entities. In fact, the identification of entity is highly dependent on the semantics of the underlying database rather than its DTD, so it usually requires the verification and decision from database administrator. Therefore, the adoption of entities for keyword search should be optional although this concept is very useful.

In digraph data model, previous approaches are heuristics-based, as the reduced tree problem on graph is as hard as NP-complete. Li et al. [11] show the reduction from minimal reduced tree problem to the NP-complete Group Steiner Tree problem on graphs. BANKS [12] uses bidirectional expansion heuristic algorithms to search as small portion of graph as possible. BLINKS [13] proposes a bi-level index to prune and accelerate searching for top-k results in digraphs. Cohen et al. [14] study the computation complexity of interconnection semantics. XKeyword [15] provides keyword proximity search that conforms to an XML schema; however, it needs to compute candidate networks and thus is constrained by schemas.

On the issue of result ranking, XRANK [2] extends Google's PageRank to XML element level, to rank among the LCA results; but no empirical study is done to show the effectiveness of its ranking function. XSEarch [1] adopts a variant of LCA, and combines a simple tf*idf IR ranking with size of the tree and the node relationship to rank results; but it requires users to know the XML schema information, causing limited query flexibility. EASE [16] combines IR ranking and structural compactness based DB ranking to fulfill keyword search on heterogenous data. Regarding to ranking methods, TF*IDF similarity [7] which is originally designed for flat document retrieval is insufficient for XML keyword search due to XML's hierarchical structure and the presence of Ambiguity 1 and Ambiguity 2. Several proposals for XML information retrieval suggest to extend the existing XML query languages [17], [18], [19] or use XML fragments [20] to explicitly specify the search intention for result retrieval and ranking.

III. Preliminaries

A. TF*IDF cosine similarity

TF*IDF (Term Frequency * Inverse Document Frequency) similarity is one of the most widely used approaches to measure the relevance of keywords and document in keyword search over flat documents. We first review its basic idea, then address its limitations for keyword search in XML. The main idea of TF*IDF is summarized in the following three rules.

- **Rule 1**: A keyword appearing in many documents should not be regarded as being more important than a keyword appearing in a few.
- **Rule 2**: A document with more occurrences of a query keyword should not be regarded as being less important for that keyword than a document that has less.
- **Rule 3**: A normalization factor is needed to balance between long and short documents, as Rule 2 discriminates against short documents which may have less chance to contain more occurrences of keywords.

To combine the intuitions in the above three rules, the TF*IDF similarity is designed:

$$\rho(q,d) = \frac{\sum_{k\in q\cap d} W_{q,k} * W_{d,k}}{W_q * W_d} \tag{1}$$

where q represents a query, d represents a flat document and k is a keyword appearing in both q and d. A larger value of $\rho(q,d)$ indicates q and d are more relevant to each other. $W_{q,k}$ and $W_{d,k}$ represent the weights of k in query q and document d respectively; while W_q and W_d are the weights of query q and document d. Among several ways to express $W_{q,k}$, $W_{d,k}$, W_q and W_d, the followings are the conventional formulae:

$$W_{q,k} = \ln\left(N/(f_k+1)\right) \tag{2}$$
$$W_{d,k} = 1 + \ln\left(f_{d,k}\right) \tag{3}$$
$$W_q = \sqrt{\sum_{k\in q} W_{q,k}^2} \tag{4}$$
$$W_d = \sqrt{\sum_{k\in d} W_{d,k}^2} \tag{5}$$

where N is the total number of documents, and document frequency f_k in Formula 2 is the number of documents containing keyword k. Term frequency $f_{d,k}$ in Formula 3 is the number of occurrences of k in document d.

$W_{q,k}$ is monotonical decreasing w.r.t. f_k (*Inverse Document Frequency*) to reflect Rule 1; while $W_{d,k}$ is monotonical increasing w.r.t. $f_{d,k}$ (*Term Frequency*) to reflect Rule 2. The logarithms used in Formula 2 and 3 are designed to normalize the raw document frequency f_k and raw term frequency $f_{d,k}$. Finally, W_q and W_d are increasing w.r.t. the size of q and d, playing the role of normalization factors to reflect Rule 3.

However, the original TF*IDF is inadequate for XML, because it is not able to fulfill the job of search intention identification or resolve keyword ambiguities resulted from XML's hierarchical structure, as Example 3 shows.

Example 3: Suppose a keyword query *"art"* is issued to search for *customer*s interested in "art" in Figure 1's XML data. Ideally, the system should rank *customer*s who do have "art" in their nested *interest* nodes before those who do not have. Moreover, it is desirable to give customer (*A*) who is only interested in art a higher rank than another customer (*B*) who has many interests including art (e.g. C4 in Figure 1).

However, it causes two problems if directly adopting original TF*IDF to XML data. (1) If the structures in *customer* nodes are not considered, customer *A* may have a lower rank than *B* if *A* happens to have more keywords in its subtrees (analog to long document in IR) than *B*. (2) Even worse, suppose a customer *C* is not interested in art but has *address* in "art street". If C has less number of keywords than A and B in XML data, then C may have higher rank than A and B.

B. Data model

We model XML document as a rooted, labeled tree, such as the one in Figure 1. Our approach exploits the prefix-path of a node rather than its tag name for result retrieval and ranking. Note that the existing works [4], [21] rely on DTD while our approach works without any XML schema information.

Definition 3.1: (**Node Type**) The type of a node n in an XML document is the prefix path from root to n. Two nodes are of same node type if they share the same prefix path.

In Definition 3.1, the reason that two nodes need to share same prefix path instead of their tag name is, there may be two or more nodes of the same tag name but of different semantics (i.e. in different contexts) in one document. E.g. In Figure 1, the *name* of publisher and the *name* of customer are of different node types, as they are in different contexts.

To facilitate our discussion later, we use the tag name instead of the prefix path of a node to denote the node type in all examples throughout this paper. Besides, we distinguish an XML node into either a value node or a structural node, to separate the content part from leaf node.

Definition 3.2: (**Value Node**) The text values contained in the leaf node of XML data (i.e. #PCDATA) is defined as a *value node*.

Definition 3.3: (**Structural Node**) An XML node labeled with a tag name is called a *structural node*. A structural node that contains other structural nodes as its children is called an *internal node*.

Definition 3.4: **(Single-valued Type)** A given node t is of *single-valued type* if each node of type t has at most one occurrence within its parent node.

Definition 3.5: **(Multi-valued Type)** A given node t is of *multi-valued type* if each node of type t has more than one occurrence within its parent node.

Definition 3.6: **(Grouping Type)** An internal node t is defined as a *grouping type* if each node of type t contains child nodes of only one multi-valued type.

XML nodes of *single-valued type* and *multi-valued type* can be easily identified when parsing the data. A node of single-valued (or multi-valued, or grouping) type is called a single-valued (or multi-valued, or grouping) node. E.g. in Figure 1, address is a *single-valued node*, while interest is a multi-valued node and interests is a grouping node for interest.

In this paper, for ease of presentation later, we assume every multi-valued node has a grouping node as its parent, as we can easily introduce a dummy grouping node in indexing without altering the data. Note a grouping node is also a single-valued node. Thus, the children of an internal node are either of same multi-valued type or of different single-valued types.

C. XML TF & DF

Inspired by the important role of data statistics in IR ranking, we try to utilize it to resolve ambiguities for XML keyword search, as it usually provides an intuitionistic and convincing way to model and capture human intuitions.

Example 4: When we talk about *"art"* in the domain of database like Figure 1, we in the first place consider it as a value in interest of customer nodes or category (or title) of book nodes. However, we seldom first consider it as a value of other node types (e.g. street with value "Art Street").

The reason for this intuition is, usually there are many nodes of interest type and category type containing "art" in their text values (or subtrees) while "art" is usually infrequent in street nodes. Such intuition (based on domain knowledge) always can be captured by statistics of the underlying database.

Similarly, when we talk about *"interest"* here, we in the first place consider it as a node type instead of a value of the title of book nodes with intuition thinking. Besides the simple reason that "interest" matches the XML tag interest, it can also be explained from statistical point of view, i.e. all interest nodes contain keyword "interest" in their subtrees.

The importance of statistics in XML keyword search is formalized as follows.

Intuition 1: The more XML nodes of a certain type T (and their subtrees) contain a query keyword k in either their text values or tag names, it is more intuitive that nodes of type T are more closely related to the query w.r.t. keyword k.

In this paper, we maintain and exploit two **important** basic statistics terms, $f_{a,k}$ and f_k^T.

*Definition 3.7: **(XML TF)*** $f_{a,k}$: The number of occurrences of a keyword k in a given *value node a* in the XML database.

*Definition 3.8: **(XML DF)*** f_k^T: The number of T-typed nodes that contain keyword k in their subtrees in the XML database.

Here, $f_{a,k}$ and f_k^T are defined in an analogous way to term frequency $f_{d,k}$ (in Formula 3) and document frequency f_k (in Formula 2) used in original TF*IDF similarity; except that we use f_k^T to distinguish statistics for different node types, as the granularity on which to measure similarity in XML scenario is a subtree rather than a document. Therefore, $f_{a,k}$ and f_k^T can be directly used to measure the similarity between a value node (with parent node of type T) and a query based on the intuitions of original TF*IDF. Besides, f_k^T is also useful in resolving ambiguities, as Intuition 1 shows. We will discuss how these two sets of statistics are used for relevance oriented ranking for XML keyword search in presence of ambiguities.

IV. Inferring Keyword Search Intention

In this section, we discuss how to interpret the search intentions of keyword query according to the statistics in XML database and the pattern of keyword co-occurrence in a query.

A. Inferring the node type to search for

The desired node type to search for is the first issue that a search engine needs to address in order to retrieve the relevant answers. Given a keyword query q, a node type T is considered as the desired node to search for only if the following three guidelines hold:

Guideline 1: T is intuitively related to every query keyword in q, i.e. for each keyword k, there should be some (if not many) T-typed nodes containing k in their subtrees.

Guideline 2: XML nodes of type T should be informative enough to contain enough relevant information.

Guideline 3: XML nodes of type T should not be overwhelming to contain too much irrelevant information.

Guideline 2 prefers an internal node type T at a higher level to be the returned node, while Guideline 3 prefers that the level of T-typed node should not be very near to the root node. For instance let's refer to Figure 1: according to Guideline 2, leaf nodes of type interest, street etc. are usually not good candidates for desired returned nodes, as they are not informative. According to Guideline 3, nodes of type customers and books are not good candidates as well, as they are too overwhelming as a single keyword search result.

By incorporating the above guidelines, we define $C_{for}(T, q)$, which is the confidence of a node type T to be the desired search for node type w.r.t. a given keyword query q as follows:

$$C_{for}(T, q) = \log_e(1 + \prod_{k \in q} f_k^T) * r^{depth(T)} \quad (6)$$

where k represents a keyword in query q; f_k^T is the number of T-typed nodes that contain k as either values or tag names in their subtrees; r is some reduction factor with range (0,1] and normally chosen to be 0.8, and $depth(T)$ represents the depth of T-typed nodes in document.

In Formula 6, the first multiplier (i.e. $\log_e(1 + \prod_{k \in q} f_k^T)$) actually models Intuition 1 to address *Guideline 1*. Meanwhile, it effectively addresses *Guideline 3*, since the candidate overwhelming nodes (i.e. the nodes that are near the root) will be assigned a small value of $\prod_{k \in q} f_k^T$, resulting in a small confidence value. The second multiplier $r^{depth(T)}$ simply reduces the confidence of the node types that are deeply nested in the XML database to address *Guideline 2*. In addition, we use product rather than sum of f_k^T (i.e. $\prod_{k \in q} f_k^T$) in the first multiplier to combine statistics of all query keywords for each node type T. The reason is, the search intention of each query usually has a unique desired node type to search for, so using product ensures that a node type needs to be intuitively related to all query keywords in order to have a high confidence as the desired type. Therefore, if a node type T cannot contain all keywords of the query, its confidence value is set to 0.

Example 5: Given a query *"customer interest art"*, node type customer usually has high confidence as the desired node type to search for, because the values of three statistics $f^{\text{customer}}_{\text{"customer"}}$, $f^{\text{customer}}_{\text{"interest"}}$ and $f^{\text{customer}}_{\text{"art"}}$ (i.e. the number of subtrees rooted at customer nodes containing "customer", "interest" and "art" in either nested text values or tags respectively) are usually greater than 1. In contrast, node type customers doesn't have high confidence since $f^{\text{customers}}_{\text{"customer"}} = f^{\text{customers}}_{\text{"interest"}} = f^{\text{customers}}_{\text{"art"}} = 1$. Similarly, node type interest doesn't have high confidence since $f^{\text{interest}}_{\text{"customer"}}$ usually has small value. E.g. in Figure 1's XML data, $f^{\text{interest}}_{\text{"customer"}} = 0$.

Finally, with the confidence of each node type being the desired type, the one with the highest confidence is chosen as the desired search for node, when the highest confidence is significantly greater than the second highest. However, when several node types have comparable confidence values, either users can be offered a choice to decide the desired one, or the system will do a search for each convincing candidate node. Although not always fully automatic, our inference approach still provides a guidance for the system-user interaction for ambiguous keyword queries in absence of syntax.

B. Inferring the node types to search via

Similar to inferring the desired search for node, *Intuition 1* is also useful to infer the node types to search via. However, unlike the search for case which requires a node type to be related to all keywords, it is enough for a node type to have high confidence as the desired search via node if it is closely related to some (not necessarily all) keywords, because a query may intend to search via more than one node type. E.g. we can search for customer(s) named "Smith" and interested in "fashion" with query *"name smith interest fashion"*. In this case, the system should be able to infer with high confidence that name and interest are the node types to search via, even if keyword "interest" is probably not related to name nodes.

Therefore, we define $C_{via}(T,q)$, which is the confidence of a node type T to be a desired type to search via as below:

$$C_{via}(T,q) = \log_e(1 + \sum_{k \in q} f_k^T) \tag{7}$$

where variables k, q and T have the same meaning as those in Formula 6. Compared to Formula 6, we use sum of f_k^T instead of product, as it is sufficient for a node type to have high confidence as the search via node if it is related to some of the keywords. In addition, if all nodes of a certain type T do not contain any keyword k in their subtrees, f_k^T is equal to 0 for each k in q, resulting in a zero confidence value, which is also consistent with the semantics of SLCA. Then, the confidence of each possible node type to search via will be incorporated into *XML TF*IDF similarity* (which will be discussed in Section V-B) to provide answers of high quality.

C. Capturing keyword co-occurrence

While statistics provide a macro way to compute the confidence of a node type to search via; it alone is not adequate to infer the likelihood of an individual value node to search via for a given keyword in the query.

Example 6: Consider a query "*customer name Rock interest Art*" searching for customers whose name includes "Rock" and interest includes "Art". Based on statistics, we can infer that name and interest-typed nodes have high confidence to *search via* by Formula 7, as the frequency of keywords "name" and "interest" are high in node types name and interest respectively. However, statistics is not adequate to help the system infer that the user wants "Rock" to be a value of name and "Art" to be a value of interest, which is intuitive with the help of keyword co-occurrence in the query. Therefore, purely based on statistics, it is difficult for search engine to differ customer C4 (with name "Art" and interest "Rock") from C3 (with name "Rock" and interest "Art") in Figure 1.

Motivated from the above example, the pattern of keyword co-occurrence in a query provides a micro way to measure the likelihood of an individual value node to search via, as a compliment of statistics. Therefore, for each query-matching value node in XML data, in order to capture the co-occurrence of keywords matching the node types and keywords matching the value nodes, the following distances are defined.

Given a keyword query q and a certain value node v, if there are two keywords k_t and k in q, such that k_t matches the type of an ancestor node of v and k matches a keyword in v, then we define the following distances.

Definition 4.1: (**In-Query Distance (IQD)**) The *In-Query Distance* $Dist_q(q,v,k_t,k)$ between keyword k and node type k_t in query q with respect to a value node v is defined as the position distance between k_t and k in q if k_t appears before k in q; Otherwise, $Dist_q(q,v,k_t,k) = \infty$.

Note the position distance of two keywords k_1 and k_2 in a query q is the difference of k_1's position and k_2's position in the query. The above definition assumes there is no repeated k_t and k in a query q. When there are multiple occurrences of k_t and/or k (e.g. *query "name smith address smith street"*), we define $Dist_q(q,v,k_t,k)$ as the minimal value for all possible combinations of each occurrence of k_t and k.

Definition 4.2: (**Structural Distance (SD)**) The *structural Distance* $Dist_s(q,v,k_t,k)$ between k_t and k w.r.t. a value node v is defined as the depth distance between v and the nearest k_t-typed ancestor node of v in XML document.

Definition 4.3: (**Value-Type Distance (VTD)**) The *Value-Type Distance* $Dist(q,v,k_t,k)$ between k_t and k w.r.t. a value node v is defined as

$$\max(Dist_q(q,v,k_t,k), Dist_s(q,v,k_t,k)).$$

In general, the smaller the value of $Dist(q,v,k_t,k)$ is, it is more likely that q intends to search via the node v with value matching keyword k. Therefore, we define the confidence of a value node v as the node to search via w.r.t. a keyword k appearing in both query q and v as follows.

$$C_{via}(q,v,k) = 1 + \sum_{k_t \in q \cap ancType(v)} \frac{1}{Dist(q,v,k_t,k)} \tag{8}$$

Example 7: Consider the query q in Example 6 again with same search intention. Let n_3 and i_3 represent the value nodes under name (i.e. Art Smith) and interest (i.e. rock music) respectively of customer C3. Similarly, let n_4 and i_4 be the values nodes under name and interest of customer C4. Now $Dist_q(q,n_3,\text{name},Art) = 3$; $Dist_s(q,n_3,\text{name},Art) = 1$; as a result $Dist(q,n_3,\text{name},Art) = 3$ and $C_{via}(q,n_3,Art)$ = 4/3. Similarly, $C_{via}(q,i_3,Rock)$ = 1; $C_{via}(q,n_4,Rock)$ = 2; and $C_{via}(q,i_4,Art)$ = 2. We can see that the values of customer C4 are larger than those of customerC3.

V. Relevance Oriented Ranking

In this section, we first summarize some unique features of keyword search in XML, and address the limitations of traditional TF*IDF similarity for XML. Then we propose a novel XML TF*IDF similarity which incorporates the confidence formulae we have designed in Section IV, to resolve the keyword ambiguity problem in relevance oriented ranking.

A. Principles of keyword search in XML

Compared with flat documents, keyword search in XML has its own features. In order for an IR-style ranking approach to smoothly apply to it, we present three principles that the search engine should adopt.

Principle 1: When searching for XML nodes of desired type D via a *single-valued node type* V, ideally, only the values and structures nested in V-typed nodes can affect the relevance of D-typed nodes as answers, whereas the existence of other typed nodes nested in D-typed nodes should not. In other words, the size of the subtree rooted at a D-typed node d (except the subtree rooted at the search via node) shouldn't affect d's relevance to the query.

Example 8: When searching for customer nodes via street nodes using a keyword query "*Art Street*", a customer node (e.g. customer $C1$ in Figure 1) with the matching keyword *"street"* shouldn't be ranked lower than another customer node (e.g. customer $C3$ in Figure 1) without the matching keyword *"street"*, regardless of the sizes, values and structures of other nodes nested in $C1$ and $C3$. Note this is different from the original TF*IDF similarity that has strong intuition to normalize the relevance score of each document with respect to its size (i.e. to normalize against long documents).

Principle 2: When searching for the desired node type D via a *multi-valued node type* V', if there are many V'-typed nodes nested in one node d of type D, then the existence of one query-relevant node of type V' is usually enough to indicate, d is more relevant to the query than another node d' also of type D but with no nested V'-typed nodes containing the keyword(s). In other words, the relevance of a D-typed node which contains a query relevant V'-typed node should not be affected (or normalized) too much by other query-irrelevant V'-typed nodes.

Example 9: Consider when searching for customers interested in art using the query "*art*", a customer with "art"-interest along with many other interests (e.g. $C4$ in Figure 1) should not be regarded as less relevant to the query than another customer who doesn't have "art"-interest but has "art street" in address (e.g. $C1$ in Figure 1).

Principle 3: The `order` of keywords in a query is usually important to indicate the search intention.

The first two principles look trivial if we know exactly the search via node. However, when the system doesn't have exact information of which node type to search via (as user issues pure keyword query in most cases), they are important in designing the formula of XML TF*IDF similarity; we will utilize them in designing Formula for W_a^q in section V-B.2.

B. XML TF*IDF similarity

We propose a recursive Formula 9, which captures XML's hierarchical structure, to compute XML TF*IDF similarity between an XML node of the desired type to search for and a keyword query. It first (*base case*) computes the similarities between leaf nodes l of XML document and the query, then (*recursive case*) it recursively computes the similarities between internal nodes n and the query, based on the similarity value of each child c of n and the confidence of c as the node type to search via, until we get the similarities of search for nodes:

$$\rho_s(q,a) = \begin{cases} \frac{\sum_{k \in q \cap a} W_{q,k}^{T_a} * W_{a,k}}{W_q^{T_a} * W_a} & \text{(a) } a \text{ is value node (base case)} \\ \frac{\sum_{c \in chd(a)} \rho_s(q,c) * C_{via}(T_c, q)}{W_a^q} & \text{(b) } a \text{ is internal node (recursive case)} \end{cases} \quad (9)$$

where q represents a keyword query; a represents an XML node; and the result $\rho_s(q,a)$ represents the similarity value between q and a.

We first discuss the intuitions behind Formula 9 briefly.
(1) In the base case, we compute the similarity values between XML leaf nodes and a given query in a similar way to original TF*IDF, since leaf nodes contain only keywords with no further structure.
(2) In the recursive case: on one hand, if an internal node a has more query relevant child nodes while another internal node a' has less, then it is likely that a is more relevant to the query than a'. This intuition is reflected as the numerator in Formula 9(b). On the other hand, we should take into account the fan-out (size) of the internal node as normalization factor, since the node with large fan-out has a higher chance to contain more query relevant children. This is reflected as the denominator of Formula 9(b).

Next, we will illustrate how each factor in Formula 9 contributes to the XML structural similarity in Section V-B.1 (for base case) and V-B.2 (for recursive case).

*1) Base case of XML TF*IDF:* Since XML leaf nodes contain keywords with no further structure, we can adopt the intuitions of original TF*IDF to compute the similarity between a leaf node and a keyword query by using statistics terms f_k^T and $f_{a,k}$ which have been explained in Section III-C.

However, unlike Rule 1 in original TF*IDF which models and assigns the same weight to a query keyword w.r.t. all documents (i.e. $W_{q,k}$ in Formula 2), we model and distinguish the weights of a keyword w.r.t. different XML leaf node types (i.e. $W_{q,k}^{T_a}$ in Formula 10).

Example 10: Keyword *"road"* may appear quite frequently in street nodes of Figure 1 while infrequently in other nodes. Thus it is necessary to distinguish the (low) weight of "road" in address from its (high) weight in other nodes. Similarly, we distinguish the weights of a query w.r.t. different XML node types (i.e. $W_q^{T_a}$), rather than fixed weight for a given query for all flat documents.

Now let's take a detailed look at Formula 9. In the base case for XML leaf nodes, each k represents a keyword appearing in both query q and value node a; T_a is the type of a's parent node; $W_{q,k}^{T_a}$ represents the weight of keyword k in q w.r.t. node type T_a. $W_{a,k}$ represents the weight of k in leaf node a; $W_q^{T_a}$ represents the weight of q w.r.t. node type T_a; and W_a represents the weight of a. Following the conventions of original TF*IDF, we propose the formulas for $W_{q,k}^{T_a}$, $W_{a,k}$, $W_q^{T_a}$ and W_a in Formula 10, 11, 12 and 13 respectively:

$$W_{q,k}^{T_a} = C_{via}(q,a,k) * \log_e\left(1 + N_{T_a}/(1 + f_k^{T_a})\right) \quad (10)$$

$$W_{a,k} = 1 + \log_e(f_{a,k}) \quad (11)$$

$$W_q^{T_a} = \sqrt{\sum_{k \in q} (W_{q,k}^{T_a})^2} \quad (12)$$

$$W_a = \sqrt{\sum_{k \in a} W_{a,k}^2} \quad (13)$$

In Formula 10, N_{T_a} is the total number of nodes of type T_a while $f_k^{T_a}$ is the number of T_a-typed nodes containing keyword k; $C_{via}(q,a,k)$ is the confidence of node a to be a search via node w.r.t. keyword k (explained in Section IV-C). In Formula 11, $f_{a,k}$ is the number of occurrences of k in value node a. Similar to Rule 1 and Rule 2 in original TF*IDF, $W_{q,k}^{T_a}$ is monotonical decreasing w.r.t. $f_k^{T_a}$, while $W_{a,k}$ is monotonical increasing w.r.t. $f_{a,k}$. W_a is normally increasing w.r.t. the size of a, so put it as part of denominator to play a role of normalization factor to balance between leaf nodes containing many keywords and those with a few keywords.

*2) Recursive case of XML TF*IDF:* The recursive case of Formula 9 recursively computes the similarity value between an internal node a and a keyword query q in a bottom-up way based on two intuitions below.

Intuition 2: An internal node a is relevant to q, if a has a child c such that the type of c has high confidence to be a *search via* node w.r.t. q (i.e. large $C_{via}(T_c, q)$), and c is highly relevant to q (i.e. large $\rho_s(q, c)$).

Intuition 3: An internal node a is more relevant to q if a has more query-relevant children when all others being equal.

In the recursive case of Formula 9, c represents one child node of a; T_c is the node type of c; $C_{via}(T_c, q)$ is the confidence of T_c to be a search via node type presented in Formula 7; $\rho_s(q, c)$ represents the similarity between node c and query q which is computed recursively; W_a^q is the overall weight of a for the given query q.

Next, we explain the similarity design of an internal node a in Formula 9: we first get a weighted sum of the similarity values of all its children, where the weight of each child c is the confidence of c to be a *search via* node w.r.t. query q. This weighted sum is exactly the numerator of formula 9, which also follows *Intuition 2* and *3* mentioned above. Besides, since *Intuition 3* usually favors internal nodes with more children, we need to normalize the relevance of a to q. That naturally leads to the use of W_a^q (Formula 14) as the denominator.

3) Normalization factor design:

$$W_a^q = \begin{cases} \sqrt{\sum_{c \in chd(a)} (C_{via}(T_c, q) * B + DW(c))^2} & \text{(a) if } a \text{ is grouping node} \\ \sqrt{\sum_{T \in chdType(T_a)} C_{via}(T, q)^2} & \text{(b) otherwise} \end{cases} \quad (14)$$

Formula 14 presents the design of W_a^q, which is used as a normalization factor in the recursive case of XML TF*IDF similarity formula. W_a^q is designed based on *Principle 1* and *Principle 2* pointed out in section V-A.

Formula 14(a) presents the case that internal node a is a *grouping node*; then for each child c of a (i.e. $c \in chd(a)$), B is considered as a Boolean flag: $B = 1$ if $\rho_s(q, c) > 0$ and $B = 0$ otherwise; $DW(c)$ is a small value as the default weight of c which we choose $DW(c) = 1/\log_e(e - 1 + |chd(a)|)$ if $B = 0$ and $DW(c) = 0$ if $B = 1$, where $|chd(a)|$ is the number of children of a, so that W_a^q for grouping node a grows with the number of query-irrelevant child nodes, but grows very slowly to reflect *Principle 2*. Note $DW(c)$ is usually insignificant as compared to $C_{via}(T_c, q)$.

Now let's explain the reason that we design Formula 14(a). The intuition for the formula of *grouping node* a comes from *Principle 2*, so we don't count $C_{via}(T_c, q)$ in the normalization unless c contains some query keywords within its subtree. In this way, the similarity of a to q will not be significantly normalized (or affected) even if a has many query-irrelevant child nodes of the same type. At the same time, with the default weight $DW(c)$, we still provide a way to distinguish and favor a grouping node with small number of children from another grouping node with many children, in case that the two contain the same set of query-relevant child nodes. Informally speaking, the more compact the meaningful answer is, the higher the rank it is given.

When internal node a is a *non-grouping node*, we compute W_a^q based on the type of a rather than each individual node. In Formula 14(b), $chdType(T_a)$ represents the node types of the children of a, and it computes the same W_a^q for all a-typed nodes even if each individual a-typed node may have different set of child nodes (e.g. some customer nodes have nested address while some do not have).

This design has two advantages. *First*, it models *Principle 1* to achieve a normalization that the size of the subtree of individual node a does not affect the similarity of a to a query.

Example 11: Given a query q *"customer Art Street"*, since address has high confidence to be searched via (i.e. $C_{via}(address, q)$), $C1$ (with address in "Art Street") will be ranked before $C2$ (with interest in "street art") according to the normalization in Formula 14(b). However, if we compute the normalization factor based on the size of each individual node, then the high confidence for address node doesn't contribute to the normalization factor of $C2$ (who even doesn't have address and street nodes etc.). As a result, $C2$ has a good chance to be ranked before $C1$ due to its small size which results in small normalization factor.

Second, Formula 14(b)'s design has advantage in term of computation cost. With W_a^q for non-grouping node computed based on node types instead of data nodes, we only need to compute W_a^q for all a-typed nodes once for each query, instead of repeatedly computing W_a^q for each a-typed node in the data.

Note in the base case, a keyword k is less important in T-typed nodes if more T-typed nodes contain k. However, now we consider T-typed nodes are more important for keyword k (i.e. larger $C_{via}(T, k)$). These two, which seem contradictive, are in fact the key to accurate relevance based ranking.

Example 12: Consider when searching for customers with query "*customer art road*", statistics will normally give more weights to address than other node types because of the high frequency of keyword "road" in address. But if no customer node has address in "art road" but some have address in "art street", then these customer nodes will be ranked before customers with address containing "road" without "art". Because the keyword "road" has a lower weight than "art" in address nodes due to its much higher frequency.

VI. Algorithms

A. Data processing and index construction

We parse the input XML document during which we collect the following information for each node n visited: (1) assign a Dewey label $DeweyID$ [22] to n; (2) store the prefix path $prefixPath$ of n as its node type in a global hash table, so that any two nodes sharing the same $prefixPath$ have the same node type; (3) in case n is a leaf node, we create a value node a (mentioned in section III-B) as its child and summarize two basic statistics data $f_{a,k}$ (in Definition 3.7) and W_a (in Formula 13) at the same time. Besides, we also build two indices in order to speedup the keyword query processing.

The first index built is called keyword inverted list, which retrieves a list of value nodes in document order whose values contain the input keyword. In particular, we have designed and evaluated three candidates for the inverted list: (1) Dup, the most basic index which stores only the dewey id and XML TF $f_{a,k}$; (2) DupType, which stores an extra node type (i.e. its prefix path) compared to Dup; (3) DupTypeNorm, which stores an extra normalization factor W_a (in Formula 13) associated with this value node compared to DupType. DupTypeNorm provides the most efficient computation of XML TF*IDF, as it costs the least index lookup time; in contrast Dup and DupType need extra index lookup to gather the value of $W_{a,k}$ (see formula 11) to compute W_a online.

Given a keyword k, the inverted list returns a set of nodes a in document order, each of which contains the input keyword and is in form of a tuple $<DeweyID$, $prefixPath$, $f_{a,k}$, $W_a>$. Each term here has been explained as above. In order to facilitate the explanations of the algorithm, we name such tuple as "*Node*". It supports the following operations:

- getDeweyID(a,k) returns the Dewey id of value node a.
- getPrefix(a,k) returns the prefix path of a in XML data.
- getFrequency(a,k) returns the value of $f_{a,k}$.

Algorithm 1: KWSearch($keywords[m]$, $IL[m]$, $F[m]$)

```
1  Let max = 0; $T_{for}$ = null
2  List L$_{for}$ = getAllNodeTypes()
3  foreach $T_n \in L_{for}$ do
4      $C_{for}(T_n, keywords)$ = getSearchForConfidence($T_n$,keywords)
5      if ($C_{for}(T_n) >$ max) then
6          max = $C_{for}(T_n)$; $T_{for}$ = $T_n$
7  LinkedList rankedList
8  $N_{for}$ = getNext($T_{for}$)
9  while (!end($IL[1]$) || ... || (!end($IL[m]$))) do
10     Node $a$ = getMin(IL[1],IL[2],...,IL[m])
11     if (!isAncestor($N_{for}$, a)) then
12         $\rho_s$(keywords,$N_{for}$) = getSimilarity($N_{for}$,keywords)
13         rankedList.insert($N_{for}$, $\rho_s$(keywords,$N_{for}$))
14         $N_{for}$ = getNext($T_{for}$)
15     if (isAncestor($N_{for}$, a)) then
16         $\rho_s(keywords, a)$ = getSimilarity(a,keywords)
17     else
18         $\rho_s(keywords, a)$ = 0
19 return rankedList;
```

The second index built is called frequency table, which stores the frequency f_k^T for each combination of keyword k and node type T in XML document. Its worst case space complexity is O(K*T), where K is the number of distinct keywords and T is the number of node types in XML database. Since the number of node types in a well designed XML database is usually small (e.g. 100+ in DBLP 370MB and 500+ in XMark 115MB), the frequency table size is comparable to inverted list. It is indexed by keywords using Berkeley DB B+-tree [23], so the index lookup cost is O(log(K)). It supports getFrequency(T,k) which returns the value of f_k^T.

Note that values returned by these operations are important to compute the result of the formulae presented in Section V.

B. Keyword search & ranking

Algorithm 1 presents a flowchart of keyword search and result ranking. The input parameters $keywords[m]$ is a keyword query containing m keywords. Based on the inverted lists built after pre-processing the XML document, we extract the corresponding lists $IL[1]$, ..., $IL[m]$ for each keyword in the query. F is the frequency table mentioned in section VI-A. In particular, Algorithm 1 executes in three steps.

First, it identifies the search intention of the user, i.e. to identify the most desired search for node type (line 1-6). In particular, it first collects all distinct node types in XML document (line 2). Then for each node type, we compute its confidence to be a search for node through Formula 6, and choose the one with the maximum confidence as the desired search for node type T_{for} (line 3-6).

Second, for each search for node candidate N_{for}, it computes the XML TF*IDF similarity between n and the given keyword query (line 7-18). We maintain a $rankedList$ to contain the similarity of each search for node candidate (line 7). N_{for} is initially set to the first node of type T_{for} in document order (line 8). The computation of XML TF*IDF similarity between an XML node and the given query is computed recursively in a bottom-up way (line 9-18): for each N_{for}, we first extract node a which occurs first in document order (line 10), then compute the similarity of all leaf nodes a by calling Function $getSimilarity()$, then go one level up to compute the similarity of the lowest *internal node* (line 15-18), until it reaches up to N_{for}, which is actually the root of all nodes computed before. Then it computes the similarity between current N_{for} and the query (line 12), insert a pair (N_{for}, ρ) into $rankedList$ (line 13), and move the cursor to next N_{for} by calling function $getNext()$ and calculate the similarity of next N_{for} in the same way (line 14).

Third, it returns the ranked list of all search for node candidates by their similarity to the query (line 19).

Function `getSimilarity`(*Node a, q[n]*)

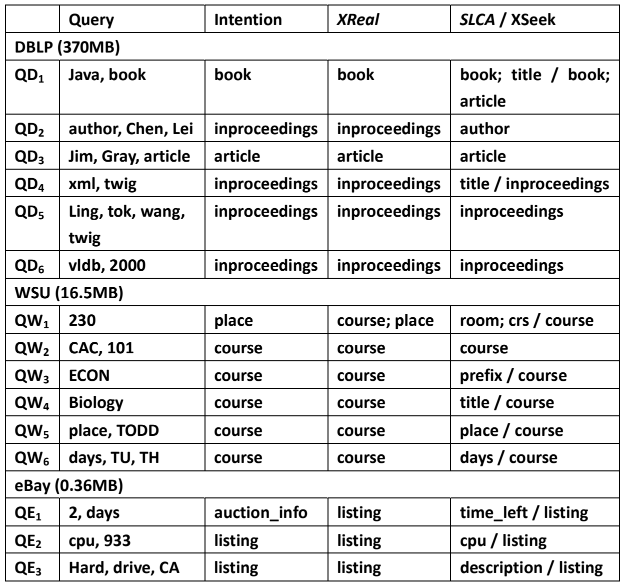
```
1  if (isLeafNode(a)) then
2      foreach $k \in q \bigcap a$ do
3          $C_{via}(q, a, k)$ = getKWCo-occur(q,a,k);
4          $W_{q,k}^{T_a}$ = getQueryWeight(q,k,a);
5          $W_{q,k}^{T_a}$ = $C_{via}(q, a, k)$ * $W_{q,k}^{T_a}$;
6          $W_{a,k}$ = 1+log$_e$(f$_{a,k}$);
7          sum += $W_{q,k}^{T_a}$ * $W_{a,k}$;
8      $\rho_s(q, a)$ = sum/($W_q^{T_a}$*getWeight(a));
9  if (isInternalNode(a)) then
10     $W_a^q$ = getQWeight(a,q);
11     foreach c∈child(a) do
12         $T_c$ = getNodeType(c);
13         $C_{via}(T_c$,q) = getSearchViaConfidence();
14         sum += $getSimilarity(c, q)$ * $C_{via}(T_c$,q);
15     $\rho_s(q, a)$ = sum/$W_a^q$;
16 return $\rho_s(q, a)$;
```

Function $getSimilarity()$ presents the procedure of computing XML TF*IDF similarity between a document node a and a given query q of size n. There are two cases to consider. `Case 1`: a is a leaf node (line 1-8). For each keyword k in both a and q, we first capture whether k co-occurs with keyword k_t matching some node type. Line 3-8 present the calculation details of $\rho_s(q, a)$ in Formula 9(a). The statistics in line 3,5,6 are illustrated in Formula 8, 10 and 11 respectively. `Case 2`: a is an internal node (line 9-15). We compute a's similarity $\rho_s(q, a)$ w.r.t. query q by exactly following Formula 9(b). $\rho_s(q, a)$ is computed by a sum of the product of the similarity of each of its child c and the confidence value of c as a search via node (line 11-14). Finally, $\rho_s(q, a)$ is normalized by a factor W_a^q (line 15), which is the weight of internal node a w.r.t. q. Lastly, we return the similarity value (line 16).

Moreover, XReal can work on both semi-structured and unstructured data, since unstructured data is a special case of semi-structured data with no structure, and XML TF*IDF ranking formula 9(a) for value node can be easily simplified to original TF*IDF Formula 1 by ignoring the node type.

VII. Experiments

We have performed comprehensive experiments to compare the effectiveness, efficiency and scalability of XReal with SLCA and XSeek. XReal and SLCA are implemented in Java and run on a 3.6GHz Pentium 4 machine with 1GB RAM running Windows XP; the binary file of XSeek is generously provided by its author. We have tested both synthetic and real datasets. The synthetic dataset is generated using XMark benchmark [24] with size 115MB; WSU and eBay from Washington XML Data Repository [25] and DBLP 370MB are used as real datasets. Berkeley DB Java Edition [23] is used to store the keyword inverted lists and frequency table.



	Query	Intention	*XReal*	*SLCA* / XSeek
DBLP (370MB)				
QD_1	Java, book	book	book	book; title / book; article
QD_2	author, Chen, Lei	inproceedings	inproceedings	author
QD_3	Jim, Gray, article	article	article	article
QD_4	xml, twig	inproceedings	inproceedings	title / inproceedings
QD_5	Ling, tok, wang, twig	inproceedings	inproceedings	inproceedings
QD_6	vldb, 2000	inproceedings	inproceedings	inproceedings
WSU (16.5MB)				
QW_1	230	place	course; place	room; crs / course
QW_2	CAC, 101	course	course	course
QW_3	ECON	course	course	prefix / course
QW_4	Biology	course	course	title / course
QW_5	place, TODD	course	course	place / course
QW_6	days, TU, TH	course	course	days / course
eBay (0.36MB)				
QE_1	2, days	auction_info	listing	time_left / listing
QE_2	cpu, 933	listing	listing	cpu / listing
QE_3	Hard, drive, CA	listing	listing	description / listing

Fig. 2. Test on inferring the *search for* node

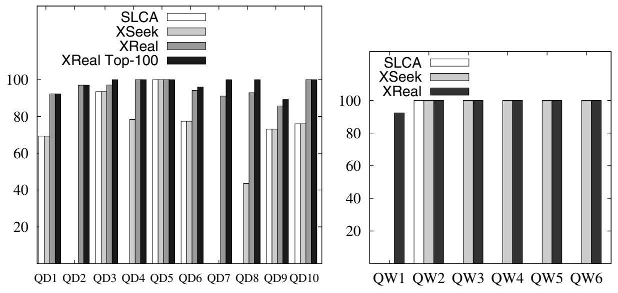
The effectiveness test contains two parts: (1) the quality of inferring the desired *search for* node; (2) the quality of our ranking approach.

A. *Search effectiveness*

1) Infer the search for node: To test XReal's accuracy in inferring the desired search for node, we make a survey of 20 keyword queries, most of which do not contain an explicit search for node. To get a fairly objective view of user search intentions in real world, we post this survey online and ask for 46 people to write down their desired search for and search via nodes. We summarize their answers and choose the queries that more than 80 percentage of users agree on a same search intention. The final queries are shown in Figure 2, and some queries contain ambiguities: e.g. QD_1 and QD_3 have both Ambiguity 1 and Ambiguity 2; QD_2, QD_6 and QW_1 have Ambiguity 2. The 4th column contains the search for node inferred by XReal while the 5th column contains the majority node types returned by SLCA and XSeek, as the semantics of SLCA cannot guarantee all results are of the same node type.

We find XReal is able to infer a desired search for node in most queries, especially when the search for node is not given explicitly in the query (e.g. QD_2, QD_4, QW_2, QE_1), or its choice is not unique (e.g. QD_1, QD_3), or both cases such as QW_1. XSeek just blindly infers the return nodes of individual keyword matches case by case, rather than addressing the major search intention(s), whereas XReal does so before it goes to find individual matches.

In addition, if more than one candidate have comparable confidence to be a search for node, XReal returns all possible candidates (for user to decide) or returns a ranked list for each such candidate if user interaction is not preferred. E.g. in QW_1, both place and course can be the return node, as the frequency of "230" in subtrees of course and place are comparable. The search for node models a real world object, so we choose to return sub-trees rooted at the desired search for node, and provide links to the descendants of subtrees for user interested in particular parts of the subtree to explore.



(a) DBLP (b) WSU

Fig. 3. Precision Comparison(%)

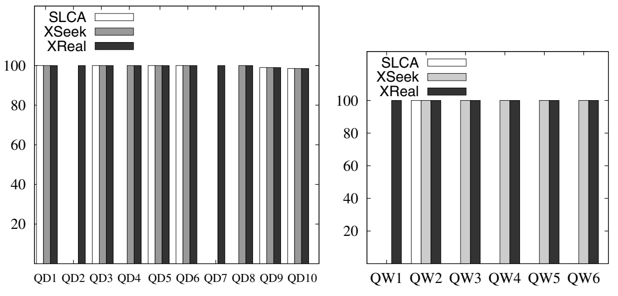
2) Precision, Recall & F-measure: To measure the search quality, we evaluate all queries in Figure 2, and summarize two metrics, i.e. precision and recall borrowed from IR field. The results are shown in Figure 3 and 4. Precision measures the percentage of the output subtrees that are desired; recall measures the percentage of the desired subtrees that are output. We obtain the correct answers by running the schema-aware XQuery and additionally verifying the correctness manually.

To evaluate XReal's performance on large real datasets, we include four more queries for DBLP: QD_7 "Philip Bernstein"; QD_8 "WISE"; QD_9 "ER 2005"; QD_{10} "LATIN 2006". Each of these queries have Ambiguity 2 problem, e.g. "WISE" can be the *booktitle*, *title* of inproceedings, or a value of *author*.

As users are always interested in top-k results, so we compute XReal's *top-100* precision besides the overall precision of SLCA, XSeek and XReal on DBLP and WSU, shown in Figure 3; results on eBay are not shown due to space limit. We have four main observations as below.

(1) XReal achieves higher precision than SLCA and XSeek for the queries that contain ambiguities (e.g. QD_1-QD_4, QD_6-QD_{10}, QW_1). E.g. in QD_3 which intends to find articles written by author "Jim Gray". Since "article" can be either a tag name or a value of title node, and "Jim" and "Gray" can appear in one author or two different authors, SLCA and XSeek generate many false positive results and lead to low accuracy, while XReal addresses these ambiguities well. As another example in QD_9 which intends to find the inproceedings of *ER* conference in year 2005. As "ER" appears in both *booktitle* and *title*, and "2005" appears in both *title* and *year*, XSeek returns not only the intended results, but also other inproceedings whose title contains both keywords; but XReal correctly interprets the search intention.

(2) We find SLCA suffers a zero precision and recall from the pure keyword value query, e.g. QD_4, QD_7, QD_8, QW_1 and QW_3 etc. Because the result returned by SLCA contains nothing relevant except the SLCA node. E.g. for QD_8 SLCA returns the *booktitle* or *title* nodes containing "WISE", while user wants to find the inproceedings of "WISE" conference. However, XReal detects keyword "WISE" has large occurrences as a *booktitle* of inproceedings and correctly captures the search intention. XSeek suffers a zero precision in QD_2 and QD_7, mainly because it mistakenly decides "*author*" as an entity, while the query intends to find the publications.



(a) DBLP (b) WSU

Fig. 4. Recall Comparison(%)

(3) XReal Performs as well as XSeek (in both recall and precision) when queries have no ambiguity in XML data (e.g. QD_5, QW_4-QW_6, etc).

(4) By comparing the overall and *top-100* precision of XReal on DBLP in Figure 3(a), *XReal Top-100* has a higher precision, which indirectly proves our ranking strategy works well w.r.t. the user search intention on large datasets.

TABLE I

F-MEASURE COMPARISON

F-measure	*SLCA*	*XSeek*	*XReal*	*XReal top-100*
DBLP	0.272	0.3461	0.4748	0.4799
WSU	0.0083	0.4162	0.4967	0.497
EBAY	0	0.4002	0.4002	0.4002

Furthermore, we adopt *F-measure* used in IR as the weighted harmonic mean of precision and recall. We compute the average precision and recall of all queries in Figure 2 (plus QD_7-QD_{10}) for each dataset, adopting formula $F = precision * recall/(precision + recall)$ to get F-measure in Table I. We find XReal beats SLCA and XSeek on all datasets, and achieves almost a perfect value of F which is 0.5 on WSU.

TABLE II

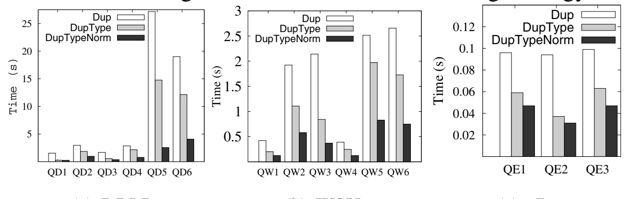
RANKING PERFORMANCE OF XREAL

Dataset	*Top-1 Number/Total Number*	*R-Rank*	*MAP*
DBLP	27/30	0.946	0.925
WSU	8/10	0.85	0.803
eBay	9/10	0.9	0.867
XMark	7/10	0.791	0.713

B. *Ranking effectiveness*

To evaluate the effectiveness of XReal's ranking strategy, we use three measures widely adopted in IR field. (1) *Number of top-1 answers that are relevant.* (2) *Reciprocal rank* (R-rank). For a given query, the reciprocal rank is 1 divided by the rank at which the first correct answer is returned, or 0 if no correct answer is returned. (3) *Mean Average Precision (MAP).* A precision is computed after each relevant answer is retrieved, and MAP is the average value of such precisions. The first two measure how good the system returns one relevant answer, while the third one measures the overall effectiveness for top-k answers returned, k=40 for DBLP (as DBLP data has very large size) and k=20 for others.

We evaluate a set of 30 randomly generated queries on DBLP, and 10 queries on WSU, eBay and XMark, with an average of 3 keywords. The average values of these metrics are recorded in Table II. We find XReal has an average R-rank greater than 0.8 and even over 0.9 on DBLP. Besides, XReal returns the relevant result in its top-1 answer in most queries, which shows high effectiveness of our ranking strategy.



(a) DBLP (b) WSU (c) eBay

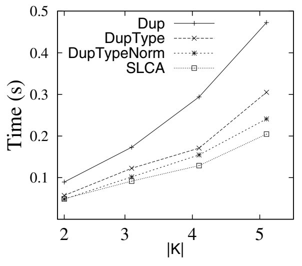
Fig. 5. Response time on individual queries

C. *Efficiency*

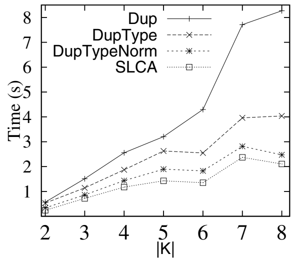
We compare the query response time of XReal adopting three indices for keyword inverted list mentioned in section VI-A, i.e. *Dup*, *DupType* and *DupTypeNorm*, measured by the timestamp difference between a query is issued and result is returned. Throughout section VII, XReal refers to the one adopting *DupTypeNorm*. Figure 5 shows the time on hot cache for queries listed in Figure 2. *DupTypeNorm* outperforms the other two on all three real datasets, about 2 and 4 times faster than *DupType* and *Dup* respectively. Because *DupTypeNorm* stores the dewey id, node type and normalization factor (for value nodes) together, thus it needs less number of index lookups to fulfill the similarity computation in Formula 9. Such advantage is significant when the number of keywords is large or query result size is large, e.g. QD_5 and QD_6 in Figure 5(a).

D. *Scalability*

Among the existing keyword search systems SLCA[3], GDMCT[5] and XSEarch[1], SLCA is recognized as the most efficient one so far, so we compare XReal with SLCA on DBLP and XMark. For each dataset, we test a set of 50 randomly generated queries, each guarantees to have at least one SLCA result and contains $|K|$ number of keywords, where $|K|$ = 2 to 8 for DBLP and $|K|$ = 2 to 5 for XMark. The response time is average time of the corresponding 50 queries in four executions on hot cache, as shown in Figure 6. From Figure 6(a) and 6(b), we find XReal is nearly 20% slower than SLCA on both datasets which is acceptable, because SLCA only find all the SLCA nodes resulting in low accuracy, while XReal does extra search intention identification, precise result retrieval and ranking; and XReal finds extra results (satisfying the boolean OR semantics). So this overhead is worthwhile. We also find, the response time of each proposed index increases as number of keywords increases. In particular, the one using *DupTypeNorm* index costs less time than *DupType*, in turn less than *Dup*. XReal adopting *DupTypeNorm* index scales as well as SLCA, especially when $|K|$ varies from 5 to 8 for DBLP (Figure 6(b)).



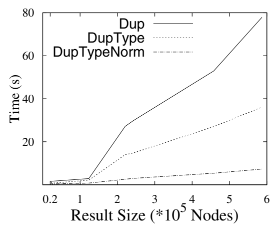
(a) XMark



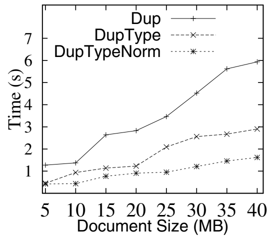
(b) DBLP

Fig. 6. Response time on different number of keywords |K|

Besides, we evaluate the scalability of those indices by drawing the relationship between the response time and query result size (in term of number of nodes returned). A range of 15 queries with various result sizes run over DBLP, and the result is shown in Figure 7(a). We can see *DupTypeNorm* again outperforms the other two, and scales linearly w.r.t. the query result size. Similarly, we test the response time of a query "*location united states item*" on XMark data of size 5MB up to 40MB. As shown in Figure 7(b), both *DupTypeNorm* and *DupType*'s response time increases linearly w.r.t. the data size.



(a) DBLP



(b) XMark

Fig. 7. Response time w.r.t. result/document size

VIII. Conclusion

In this paper, we study the problem of effective XML keyword search which includes the identification of user search intention and result ranking in the presence of keyword ambiguities. As statistics provide an objective way to model patterns and draw inferences on the underlying data, we utilize them to infer user's search intention and rank the query results. In particular, we define XML TF (term frequency) and XML DF (document frequency). Based on these two statistic terms, we design formulas to compute the confidence level of each candidate node type to be a *search for/search via node*. Then, we propose a novel *XML TF*IDF similarity* ranking scheme which takes the above confidence levels and the co-occurrence of keywords into consideration, and well captures the hierarchical structure of XML document. Moreover, we are the first paper to investigate and solve the keyword ambiguity problem. As a result, we implement an XML keyword search engine prototype called XReal, which exploits only data statistics to combine search intention identification, search result retrieval and relevance oriented ranking together as a single problem in XML keyword search. Extensive experiment results show XReal is much more effective than the existing approaches. For future work, we are now investigating the ranking strategy for XML documents with ID/IDREFs.

References

[1] S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv, "XSEarch: A semantic search engine for XML," in *Proc. of VLDB Conference*, 2003, pp. 45–56.
[2] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram, "XRANK:ranked keyword search over XML documents," in *SIGMOD*, 2003.
[3] Y. Xu and Y. Papakonstantinou, "Efficient keyword search for smallest LCAs in XML databases," in *SIGMOD*, 2005, pp. 537–538.
[4] Z. Liu and Y. Chen, "Identifying meaningful return information for xml keyword search," in *SIGMOD Conference*, 2007.
[5] V. Hristidis, N. Koudas, Y. Papakonstantinou, and D. Srivastava, "Keyword proximity search in XML trees," in *TKDE*, 2006, pp. 525–539.
[6] M. Ley, "Dblp computer science bibliography record," http://www.informatik.uni-trier.de/ ley/db/.
[7] G. Salton and M. J. McGill, *Introduction to Modern Information Retrieval*. New York, NY, USA: McGraw-Hill, Inc., 1986.
[8] A. Schmidt, M. L. Kersten, and M. Windhouwer, "Querying xml documents made easy: Nearest concept queries." in *ICDE*, 2001, pp. 321–329.
[9] Y. Li, C. Yu, and H. V. Jagadish, "Schema-free XQuery," in *VLDB2004*.
[10] C. Sun, C. Y. Chan, and A. K. Goenka, "Multiway slca-based keyword search in xml data," in *WWW*, 2007, pp. 1043–1052.
[11] W. S. Li, K. S. Candan, Q. Vu, and D. Agrawal, "Retrieving and organizing web pages by information unit," in *WWW*, 2001, pp. 230–244.
[12] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar, "Bidirectional expansion for keyword search on graph databases," in *Proc. of VLDB Conference*, 2005, pp. 505–516.
[13] H. He, H. Wang, J. Yang, and P. S. Yu, "Blinks: ranked keyword searches on graphs," in *SIGMOD Conference*, 2007, pp. 305–316.
[14] S. Cohen, Y. Kanza, B. Kimelfeld, and Y. Sagiv, "Interconnection semantics for keyword search in xml," in *CIKM*, 2005, pp. 389–396.
[15] V. Hristidis, Y. Papakonstantinou, and A. Balmin, "Keyword proximity search on XML graphs," in *ICDE*, 2003, pp. 367–378.
[16] G. Li, B. C. Ooi, J. Feng, J. Wang, and L. Zhou, "Ease: Efficient and adaptive keyword search on unstructured, semi-structured and structured data," in *SIGMOD*, 2008.
[17] N. Fuhr and K. Großjohann, "Xirql: A query language for information retrieval in xml documents," in *SIGIR*, 2001, pp. 172–180.
[18] S. Amer-Yahia, L. V. S. Lakshmanan, and S. Pandit, "Flexpath: flexible structure and full-text querying for xml," in *SIGMOD conference*, 2004.
[19] A. Theobald and G. Weikum, "The index-based xxl search engine for querying xml data with relevance ranking," in *EDBT*, 2002, pp. 477–495.
[20] D. Carmel, Y. S. Maarek, M. Mandelbrod, Y. Mass, and A. Soffer, "Search xml documents via xml fragments," in *SIGIR*, 2003, pp. 151–158.
[21] G. Li, J. Feng, J. Wang, and L. Zhou, "Effective keyword search for valuable lcas over xml documents," in *CIKM*, 2007, pp. 31–40.
[22] V. Vesper., "Let's do dewey," http://www.mtsu.edu/ vvesper/dewey.html.
[23] "Berkeley DB," http://www.sleepycat.com/.
[24] "http://www.xml-benchmark.org/."
[25] "http://www.cs.washington.edu/research/xmldatasets."