Towards an Effective XML Keyword Search

Zhifeng Bao, Jiaheng Lu, Tok Wang Ling, *Senior Member*, *IEEE*, and Bo Chen

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; a keyword can appear as the tag name of different XML node types with different meanings. 3) As the search results are subtrees 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. To complement our result ranking framework, we also take the popularity into consideration for the results that have comparable relevance scores. Lastly, extensive experiments have been conducted to show the effectiveness of our approach.

Index Terms—XML, keyword search, ranking.

1 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 terms 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 three reasons.

- **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.
- **Ambiguity 3**: A keyword can appear as an XML tag name in different contexts and carry different meanings.

For example, see the XML document in Fig. 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); *art* appears as a text value of interest, address, and name node; *name* appears as the tag name of the name of both customer and publisher.

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 [8], [10], [29], [22], [12], [23], 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 [29]. Each SLCA result of a keyword query contains all query keywords but has no subtree which also contains all the keywords.

In particular, regarding to Issues 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. For example, when a query "Jim Gray" that intends to find Jim Gray's publications on DBLP [17] 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 first author's name and "Gray" is a term in second author, and publications with *title* containing both keywords. It is

- *Z. Bao, T.W. Ling, and B. Chen are with the School of Computing, National University of Singapore, Computing 1, 13 Computing Drive, Singapore 117417, Republic of Singapore. E-mail: {baozhife, lingtw, chenbo}@comp.nus.edu.sg.*
- *J. Lu is with the Key Lab of Data Engineering and Knowledge Engineering, MOE, Renmin University of China, Beijing, 100872, China. E-mail: jiahenglu@ruc.edu.cn.*

Manuscript received 15 May 2009; revised 20 Oct. 2009; accepted 7 Dec. 2009; published online 12 Apr. 2010.
Recommended for acceptance by Y. Ioannidis, D. Lee, and R. Ng.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDESI-2009-05-0431.
Digital Object Identifier no. 10.1109/TKDE.2010.63.

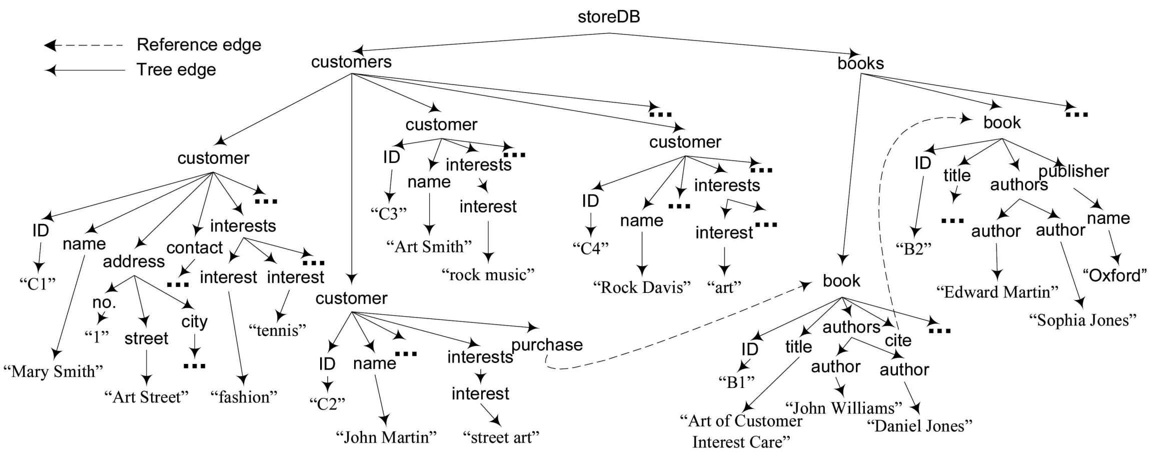


Fig. 1. Portion of data tree for an online bookstore XML database.

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 when the schema information of XML data is not available [22]. The actual reason is, none of them can solve the above keyword ambiguity problems, as demonstrated by the following example:

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

If adopting SLCA, we will get five 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 Fig. 1. Since SLCA cannot well address the search intention, all these five 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 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. For example, 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).

It motivates us to 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. As a result, 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. A search engine 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. The major 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 term frequency (TF) and XML document frequency (DF), 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 1-3 in a heuristic way. Besides, the popularity of query results is designed to distinguish the results with comparable relevance scores.
6. We implement the proposed techniques in a keyword search engine prototype called XReal. Extensive experiments show their effectiveness, efficiency, and scalability.

The rest of the paper is organized as follows: We present the related work in Section 2 and data model in Section 3. Section 4 infers user search intention. Section 5 discusses the ranking scheme. Section 7 presents the search algorithms. Experiment is discussed in Section 8, and we conclude in Section 9.

2 Related Work

Extensive research efforts have been conducted in XML keyword search to find the smallest substructures 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, lowest common ancestor (LCA) semantics is first proposed and studied in [25], [10] to find XML nodes, each of which contains all query keywords within its subtree. Subsequently, SLCA (smallest LCA [21], [29]) is proposed to find the smallest LCAs that do not contain other LCAs in their subtrees. GDMCT (minimum connecting trees) [12] excludes the subtrees rooted at the LCAs that do not contain the query keywords. Sun et al. [26] generalize SLCA to support keyword search involving combinations of AND and OR boolean operators. XSeek [22] 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, and book in Fig. 1 are identified as object classes by XSeek. However, it causes the multivalued attribute to be mistakenly identified as an entity, causing the inferred return node not as intuitive as possible. For example, phone and interest are not intuitive as entities. In fact, the identification of entity is highly dependent on the semantics of underlying database rather than its DTD, so it usually requires the verification and decision from database administrator. Liu and Chen [23] propose an axiomatic way to judge the completeness and correctness of a certain keyword search semantics.

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. [20] show the reduction from minimal reduced tree problem to the NP-complete Group Steiner Tree problem on graphs. BANKS [16] uses bidirectional expansion heuristic algorithms to search as small portion of graph as possible. BLINKS [11] proposes a bilevel index to prune and accelerate searching for top-k results in digraphs. Cohen et al. [7] study the computation complexity of interconnection semantics. XKeyword [13] 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 [10] 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 [8] 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 [19] combines IR ranking and structural compactness based DB ranking to fulfill keyword search on heterogenous data. Regarding to ranking methods, TF*IDF similarity [24] 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 1-3. Several proposals for XML information retrieval suggest to extend the existing XML query languages [9], [4], [27] or use XML fragments [6] to explicitly specify the search intention for result retrieval and ranking.

As an extension of [5], we have several major updates: 1) We complement our ranking framework by adding the popularity of a query result into consideration in Section 6. 2) Accordingly, an enhanced data model is built in Section 3.2, new index and efficient algorithm are designed to compute the popularity score in Section 7, and more experiments are conducted. 3) A new Principle 3 (in Section 5.1) is proposed to take into account the proximity of keywords, and accordingly a new In-Query Distance is designed in Definition 4.1.

3 Preliminaries

3.1 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 (2) is the number of documents containing keyword k. Term frequency $f_{d,k}$ in (3) is the number of occurrences of k in document d.

$W_{q,k}$ is monotonically decreasing w.r.t. f_k (*Inverse Document Frequency*) to reflect Rule 1; while $W_{d,k}$ is monotonically increasing w.r.t. $f_{d,k}$ (*Term Frequency*) to reflect Rule 2. The logarithm in (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 Fig. 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 should 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 Fig. 1).

However, it causes two problems if directly adopting the original TF*IDF to XML data. 1) If the structure in *customer* nodes is not considered, customer A may have a lower rank than B if A happens to have more keywords in its subtree (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 the underlying XML data, then C may have a higher rank than A and B.

3.2 Data Model

We model XML document as a rooted, labeled tree plus a set of directed IDRef edges between XML nodes, such as the one in Fig. 1. In contrast to general directed graph model, the containment edge and IDRef edge are distinguished in our model. Our approach exploits the prefix path of a node rather than its tag name for result retrieval and ranking. Note that the existing works [22], [18] 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 the same node type if they share the same prefix path.*

In Definition 3.1, the reason that two nodes need to share the 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. For example, in Fig. 1, the *name* of publisher and the *name* of customer are of different node types, which are storeDB/books/book/publisher/name and storeDB/customers/customer/name, respectively. Besides, when XML database contains multiple XML documents, the node type should also include the file name.

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, in order to separate the content part from leaf node, we distinguish an XML node into either a *data node* or a *structural node*.

Definition 3.2 (Data Node). *The text values that are contained in the leaf node of XML data and have no tag name are defined as data 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; otherwise, it is called a leaf node.*

In this paper, we do not consider the case that an *internal node n* contains both *data nodes* and *structural nodes*, as we can easily avoid it by adding a dummy structural node with a tag name say "*value*" between n and the data nodes during node indexing without altering the XML data.

With the above two definitions, the value part and structure part of the XML data is separated. For example, within the subtree of customer C1 in Fig. 1, address is an *internal node*, street is a *leaf node*, and "Art Street" is a *data node*.

Definition 3.4 (Single-Valued Type). *A structural 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 (Multivalued Type). *A structural node t is of multivalued type if some 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 multivalued type.*

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

In this paper, for ease of presentation later, we assume every multivalued 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 multivalued type or of different single-valued types.

In our data model, a directed edge is classified into a containment edge or an IDRef edge. A containment edge $u \rightarrow v$ denotes that u is the parent of node v. An IDRef edge from node u pointing to node v is denoted as $u \rightarrow v$, where u's attribute of type IDRef has a value equal to the ID-typed attribute of node v. For example, in Fig. 1, the directed solid lines represent containment edges; the dotted line from *purchase* (which is an attribute of *customer*) to *book* is an IDRef edge from *customer* C2 to *book* B1.

Definition 3.7 (Reference Connection). *Node v has a reference connection (RC) from node u, denoted as $RC(u, v)$, if there exists a directed path P: $u \rightarrow \cdots \rightarrow v$ from u to v, where each edge in P is an IDRef edge. The distance of a certain reference connection path P from u to v is defined as the number of IDRef edges involved in P.*

For example, in Fig. 1, the distance from customer C2 to book B2 is 2, as two IDRef edges are involved.

3.3 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 Fig. 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, while "art" is infrequent in street nodes. Such intuition (based on domain knowledge) always can be captured by statistics of underlying data. Similarly, when we talk about *"interest,"* intuitionally, we in the first place consider it as a node type instead of a value of the title of book nodes. Besides the reason that "interest" matches the XML tag interest, it can 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 3.8 ((XML TF) $f_{a,k}$). *The number of occurrences of a keyword k in a given data node a in XML data.*

Definition 3.9 ((XML DF) f_k^T). *The number of T-typed nodes that contain keyword k in their subtrees in XML data.*

Here, $f_{a,k}$ and f_k^T are defined in an analogous way to term frequency $f_{d,k}$ (in (3)) and document frequency f_k (in (2)) used in the original TF*IDF similarity, respectively; except that we use f_k^T to distinguish statistics for different node types, as the granularity on which to measure similarity in XML 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 data 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.

4 Inferring Keyword Search Intention

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

4.1 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, as the search target in a keyword query may not be specified explicitly like in structured query language. 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 Fig. 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:

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

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 (as explained in Section 3.3 to reflect Intuition 1); r is a 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 (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. Furthermore, when the schema of XML data is available, the entity can be inferred (by adopting XSeek [22]) and used to constrain the search for node candidates produced by (6), as users are usually interested in the real world entities. Similar to all the existing works [10], [29], [22], [12], in this paper, we assume each query keyword has at least one occurrence in the XML document being queried.

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 $f^{\text{customer}}_{\text{"customer"}} = f^{\text{customer}}_{\text{"interest"}} = f^{\text{customer}}_{\text{"art"}} = 1$. Similarly, node type interest doesn't have high confidence since $f^{\text{interest}}_{\text{"customer"}}$ usually has a small value. For example, in Fig. 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. Regarding to the threshold for comparableness judgment, we adopt the results from our empirical study: when the difference percentage of the scores of these node types is within 10 percent, they are viewed as "comparable." 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. For example, the search engine can provide a guidance for users to browse and select their desired node type(s) in case that the keyword queries are ambiguous, before adopting the ranking strategy to rank the individual matches.

4.2 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. For example, 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:

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

where variables k, q, and T have the same meaning as those in (6). Compared to (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 5.2) to provide answers of high quality.

4.3 Capturing Keyword Co-Occurrence

In this section, we discuss the search via confidence for a *data node*. Although statistics provide a macro way to compute the confidence of a *structural node* type to search via, it alone is not adequate to infer the likelihood of an individual *data 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-typed and interest-typed nodes have high confidence to *search via* by (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 captured. Thus, if purely based on statistics, it is difficult for a search engine to differ customer C4 (with name "Art" and interest "Rock") from C3 (with name "Rock" and interest "Art") in Fig. 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 data node to search via, as a compliment of statistics. Therefore, for each query-matching data node v in XML data, in order to capture the co-occurrence of keyword k_t matching the node types of an ancestor node of v and keyword k matching a value in v (if they do exist in the query) in both query and XML data, respectively, the following distances are defined.

The design of In-Query Distance (IQD) is motivated by an observation: when users want to specify both the predicate k_t and its value k in a keyword query, they always put k_t and k close to each other, regardless of the search habits of different users, i.e., no matter whether k is specified *before/after* k_t for a particular user.

Definition 4.1 (IQD). *The In-Query Distance $Dist_q(q, k_t, k)$ between keyword k and node type k_t in a query q is defined as the absolute value of the position distance between k_t and k in q; otherwise, $Dist_q(q, k_t, k) = \infty$.*

Note that, the above definition assumes there is no repeated k_t and k in a query q, and 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.

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 data node v is defined as the depth distance between v and the nearest k_t-typed ancestor node of v in XML data.*

IQD and SD are designed to capture the closeness of such node type k_t and keyword k in the input user query and underlying XML data, respectively. With intuition thinking, a data node v is favored when such k_t and k associated with it appear closely to each other in both the query and XML data, as stated in Intuition 2 and captured in Definition 4.3.

Intuition 2. *For a data node v, if the keyword k_t matching its associated node type and keyword k covered by v appear closely to each other in both the user query and XML data, it is more intuitive that v has a high confidence to be searched via w.r.t. keywords k_t and k.*

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 data node v is defined as*

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

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 a value matching keyword k. Note that, any monotonic function can be applied in Definition 4.3 to fulfill such intuition, while *max* is one of them. Therefore, we define the confidence of a data node v as the node to search via w.r.t. a keyword k appearing in both query q and v as follows:

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

Example 7. Consider the query q in Example 6 again. Let n_3 and i_3 represent the data nodes under name (i.e., Art Smith) and interest (i.e., rock music) of customer C3. Similarly, let n_4 and i_4 be the data nodes under name and interest of customer C4. Now, the in-query distance between name and Art is 3, i.e., $Dist_q(q, \mathrm{name}, Art) = 3$; $Dist_s(q, n_3, \mathrm{name}, Art) = 1$; as a result $Dist(q, n_3, \mathrm{name}, Art) = 3$ 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 find, the two predicates of customer C4 have a larger confidence to be searched via than those of customer C3. Intuitively, C4 should be more preferred than C3 as the result of q. We will discuss how to incorporate these values into our XML TF*IDF similarity in Section 5.2.1.

5 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 4, to resolve the keyword ambiguity problem in relevance oriented ranking.

5.1 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 Fig. 1) with the matching keyword "*street*" shouldn't be ranked lower than another customer node (e.g., customer $C3$ in Fig. 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 *multivalued 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 Fig. 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 Fig. 1).

Compared to the existing works which blindly exploit the compactness of the query results in result ranking [10], [8], [19], a significant difference of the above two principles is: the internal structure of a query result should be exploited as a critical factor to reflect the real relevance of the query results.

Principle 3: The *proximity* 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 5.2.2.

5.2 XML TF*IDF Similarity

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

We propose a recursive (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 the leaf nodes l of XML data 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.

In (9), q represents a keyword query; a represents an XML node; $\rho_s(q,a)$ represents the similarity value between q and a. We first discuss the intuitions behind (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 the 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 (9b). 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 (9b).

Next, we will illustrate how each factor in (9) contributes to the XML structural similarity in Section 5.2.1 (for base case) and Section 5.2.2 (for recursive case).

5.2.1 Base Case of XML TF*IDF

Since XML leaf nodes contain keywords with no further structure, we can adopt the intuitions of the 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 3.3.

However, unlike Rule 1 in the original TF*IDF which assigns the same weight to a query keyword w.r.t. all documents (i.e., $W_{q,k}$ in (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 (10)), as shown in Example 10.

Example 10. Keyword *street* may appear quite frequently in address nodes of Fig. 1 while infrequently in other nodes. Thus, it is necessary to distinguish the (low) weight of *street* 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 a fixed weight for a given query for all flat documents.

Now let's take a detailed look at (9). In the base case for XML leaf nodes, each k represents a keyword appearing in both query q and data 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 data 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 the original TF*IDF, we propose the formulas for $W_{q,k}^{T_a}$, $W_{a,k}$, $W_q^{T_a}$, and W_a in (10)-(13), respectively:

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

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

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

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

In (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 4.3).

In (11), $f_{a,k}$ is the number of occurrences of k in data node a. Similar to Rule 1 and Rule 2 in original TF*IDF, $W_{q,k}^{T_a}$ is monotonically decreasing w.r.t. $f_k^{T_a}$, while $W_{a,k}$ is monotonically 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.

5.2.2 Recursive Case of XML TF*IDF

The recursive case of (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 3. *An internal node a is relevant to q, if a has a child c such that the type of c has a 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 4. *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 (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 (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 (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 (9), which also follows Intuitions 3 and 4 mentioned above. Besides, since Intuition 4 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 (14) as the denominator.

5.2.3 Normalization Factor Design

Equation (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 5.1.

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

Grouping node case. Formula (14a) 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 (14a).

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-relevant 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. In other words, the *result specificity* is taken into account in this case.

Nongrouping node case. When internal node a is a *nongrouping node*, we compute W_a^q based on the type of a rather than each individual node. In (14b), $chdType(T_a)$ 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 sets 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 (14b). 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, (14b)'s design has advantage in term of computation cost. With W_a^q for nongrouping 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 that, the normalization factor in (14b) potentially favors nodes with more nested node types. However, the existence of one or a few nodes containing query keywords but with low confidence to be searched via is usually insufficient to outweigh a query-relevant search via node with high confidence. In addition, we do not choose the same normalization factor for all nodes of the same type, because we have to prevent the similarity of internal nodes (up to the search for node) from increasing monotonically from the base case of the recursive XML TF*IDF formula (i.e., (9a)), in order to avoid discriminating against nodes that are nested near the nodes to be searched for.

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 contradictory, 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.

5.2.4 Advantages of XML TF*IDF

Compatibility. The *XML TF*IDF similarity* can work on both semistructured and unstructured data, because unstructured data is a simpler kind of semistructured data with no structure, and XML TF*IDF ranking (9a) for *data node* can be easily simplified to the original TF*IDF (1) by ignoring the node type.

Robustness. Unlike existing methods which require a query result to cover all keywords [22], [29], [12], [10], we adopt a heuristic-based approach that does not enforce the occurrence of all keywords in a query result; instead, we rank the results according to their relevance to the query. In this way, more relevant results can be found, because a user query may often be an imperfect description of his real information need [15]. Users never expect an empty result to be returned even though no result can cover all keywords; fortunately, our approach is still able to return the most relevant results to users.

6 Popularity Score by IDRef

To date, our XML TF*IDF similarity only reflects the relevance of a search for node instance N_{for} in XML data. However, when two or more instances of the search for node have comparable relevance scores, is there any way to distinguish them? The answer is IDRef, which reflects the popularity of a query result to a certain extent. From user's perspective, when there are many results with comparable relevance scores, it is desired the most popular result is returned first, thus, saving much user effort in navigating all those results until finding their desired ones. The relevance and popularity score of a query result should not be trivially combined, as they are regarded to be orthogonal essentially. For example, a highly relevant query result may have a very low popularity score, probably resulting in a low overall ranking score which is undesired.

Regarding to the popularity of a search for node instance B_T of type T (i.e., a subtree rooted at the desired search for node of type T), the following guidelines are proposed:

Guideline 4: The more relevant the subtree rooted at the node that has a reference connection to N_T (or its descendants) is, the more popular N_T should be.

Guideline 5: The more the number of reference connections to N_T is, the more popular N_T should be.

Guideline 6: The closer the reference connection to N_T is, the more popular N_T should be.

Intuitively speaking, Guideline 4 favors the query result which is also referred by another highly relevant subtree. It also explains why Definition 3.7 restricts only the (direct or indirect) incoming references of v as its reference connection, while its outgoing references are omitted. Guideline 5 favors the query result which is referred by as many referring nodes as possible; Guideline 6 favors a direct reference connection.

Example 13. Consider a query $Q =$ "XML keyword search" issued on StoreDB in Fig. 1. Suppose n books $B_1, \ldots, B_n$ have comparable relevance scores. A book B_i is said to be popular if B_i is cited by as many other books as possible (Guideline 5), and those books citing B_i are also highly relevant to Q (Guideline 4); lastly, direct citation is expected (Guideline 6), as it reflects a closer relationship between the book cited and citing.

By incorporating the above three guidelines, the popularity $p(q, a)$ of a search for node instance a (which is of node type T) w.r.t. a keyword query q is defined in (15).

$$p(q, a) = \log_e \sum_{u \in idref(a)} \frac{\rho_s(q, T_u)}{d_u * h(a, v)}, \tag{15}$$

where $idref(a)$ returns a set of nodes u, each of which has a reference connection to v, where v is either a or its descendant; d_u denotes the distance of this reference connection by Definition 3.7; $h(a, v)$ denotes the height distance between node a and v. The numerator part $\rho_s(q, T_u)$ collects the XML TF*IDF similarity of (the subtree rooted at) node T_u w.r.t. query q, to address Guideline 4. Here, T_u is the parent node of u. As a decay factor, the denominator d_u reduces the contribution to the popularity of node a from node T_u via an indirect IDRef relationship, $h(a, v)$ deduces the contribution from the descendant of a, both of which effectively address Guideline 6. The monotonic feature of the summation function $\sum_{u \in idref(a)}$ is used to address Guideline 5, while logarithm function is used to normalize the effect of raw relevance score from each such participant u.

7 Algorithms

7.1 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 $DeweyID$ [28] 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 data node a (mentioned in Section 3.2) as its child and summarize two basic statistics data $f_{a,k}$ (in Definition 3.8) and W_a (in (13)) at the same time. Besides, we also build two indexes in order to speed up the keyword query processing.

The first index built is called keyword inverted list, which retrieves a list of data nodes in document order whose values contain the input keyword; moreover, an index (e.g., B+-Tree) is built on top of each inverted list for probing purpose. 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 (13)) associated with this data 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 (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 a "*Node*." It supports the following operations:

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

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 the space complexity is O($K * N$), where K is the number of distinct keywords and N 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 370 MB and $500+$ in XMark 115 MB), the frequency table size is comparable to inverted list. It is indexed by keywords using Berkeley DB B+-Tree [1], so the index lookup cost is O(log(K)). It supports getFrequency(T, k) which returns the value of f_k^T. The values returned by these operations are important to compute the result of formulae in Section 5.

Lastly, a connection table CT is built to record the direct reference connection between nodes in XML data in data parsing. Each entry in CT is in form of <Dewey(v), $cList(v)$>, where $cList(v)$ stores a list of dewey labels of nodes n in document order, where $RC(n,v)$ holds with distance $\mathrm{d} = 1$ by Definition 3.7. CT supports the operation getCList(v) which is to retrieve $cList(v)$. A B+-Tree index (with Dewey(v) as its key) is built on top of CT for fast retrieval of $cList(v)$.

7.2 Keyword Search and Ranking

Algorithm 1 presents a flowchart of keyword search and result ranking. The input parameter $Q[m]$ is a keyword query containing m keywords. Based on the inverted lists built after preprocessing the XML document, we extract the corresponding inverted lists $IL[1], \ldots, IL[m]$ for each keyword in the query. Each inverted list IL contains a set of tuples in form of <$DeweyID, prefixPath, f_a^k, W_a$>. F is the frequency table mentioned in Section 7.1. 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 (6), and choose the one with the maximum confidence as the desired search for node type T_{for} (lines 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 (lines 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 (lines 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* (lines 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), inserts a pair (N_{for}, ρ) into $rankedList$ (line 13), and moves the cursor to next N_{for} by calling function $getNext()$ and calculates the similarity of next N_{for} in the same way (line 14). Function $isAncestor(N_1, N_2)$ returns true if N_1 is an ancestor of N_2.

Third, it collects the results in $rankedList$, where their relevance difference is less than a specified threshold (lines 19 and 20), and computes their popularities by calling Function 3, and adjust their ranking positions in $rankedList$ (lines 21-23).

Algorithm 1. KWSearch($Q[m]$, $IL[m]$, $F[m]$)

1 Let $\max = 0$; $T_{for} = \mathrm{null}$
2 List L$_{for}$ = getAllNodeTypes()
3 **foreach** $T_n \in L_{for}$ **do**
4 $\quad C_{for}(T_n, Q) =$ getSearchForConfidence(T_n, Q)
5 $\quad$ **if** ($C_{for}(T_n) > max$) **then**
6 $\quad\quad \max = C_{for}(T_n)$; $T_{for} = T_n$
7 LinkedList rankedList
8 $N_{for} =$ getNext(T_{for})
9 **While** (!$end(IL[1])$ $\| \cdots \|$ (!$end(IL[m])$)) **do**
10 $\quad$ Node $a = \mathrm{getMin}(\mathrm{IL}[1], \mathrm{IL}[2], \ldots, \mathrm{IL}[m])$
11 $\quad$ **if** (!$isAncestor(N_{for}$, a)) **then**
12 $\quad\quad \rho_s(\mathrm{Q}, N_{for}) =$ getSimilarity(N_{for},Q)
13 $\quad\quad$ rankedList.insert(N_{for}, ρ_s(Q,N_{for}))
14 $\quad\quad N_{for} =$ getNext(T_{for})
15 $\quad$ **if** ($isAncestor(N_{for}$, a)) **then**
16 $\quad\quad \rho_s(Q, a) =$ getSimilarity(a,Q)
17 $\quad$ **else**
18 $\quad\quad \rho_s(Q, a) = 0$
19 **foreach** *two neighboring ordered results* r_1 *and* r_2 *in rankedList* **do**
20 $\quad$ **if** $((\rho_s(r_1, Q)$-$\rho_s(r_2, Q))/\rho_s(r_2, Q) < \sigma)$ **then**
21 $\quad\quad$ **foreach** *such* r_i **do**
22 $\quad\quad\quad$ p(Q, r_i) = getPopularity(r_i, Q, CT, L)
23 $\quad\quad$ re-rank those r_i in rankedList according to their p(Q, r_i)
24 return rankedList;

Function `getSimilarity` ($Nodea, q[n]$)

1 **if** ($isLeafNode(a)$) **then**
2 $\quad$ **foreach** $k \in q \bigcap a$ **do**
3 $\quad\quad C_{via}(q, a, k) =$ getKWCo-occur(q,a,k);
4 $\quad\quad W_{q,k}^{T_a} =$ getQueryWeight(q,k,a);
5 $\quad\quad W_{q,k}^{T_a} = C_{via}(q, a, k) * W_{q,k}^{T_a}$;
6 $\quad\quad W_{a,k} = 1 + \log_e(\mathrm{f}_{a,k})$;
7 $\quad\quad \mathrm{sum} \mathrel{+}= W_{q,k}^{T_a} * W_{a,k}$;
8 $\quad \rho_s(q, a) = \mathrm{sum}/(W_q^{T_a} * \mathrm{getWeight(a)})$;
9 **if** ($isInternalNode(a)$) **then**
10 $\quad W_a^q =$ getQWeight(a,q);
11 $\quad$ **foreach** $c \in child(a)$ **do**
12 $\quad\quad T_c =$ getNodeType(c);
13 $\quad\quad C_{via}(T_c, \mathrm{q}) =$ getSearchViaConfidence();
14 $\quad\quad$ sum += $getSimilarity(c, q) * C_{via}(T_c$,q);
15 $\quad \rho_s(q, a) = \mathrm{sum}/W_a^q$;
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 (lines 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. Lines 3-8 present the calculation details of $\rho_s(q, a)$ in (9a). The statistics in lines 3, 5, and 6 are illustrated in (8), (10), and (11), respectively.

`Case 2`: a is an internal node (lines 9-15). We compute a's similarity $\rho_s(q, a)$ w.r.t. query q by exactly following (9b).

TABLE 1
Data and Index Sizes

Data	Data Size	Dup	DupType	DupTypeNorm	CT	Index Time
DBLP	370MB	1.96GB	2.05GB	2.23GB	2MB	2.3 hours
XMark	115MB	1.26GB	1.3GB	1.32GB	13MB	58 minutes
WSU	15.6MB	13.1MB	13.4MB	14.1MB	0	91 seconds
eBay	350KB	718KB	732KB	803KB	0	10 seconds

$\rho_s(q, a)$ 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 (lines 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).

Function $getPopularity()$ computes the popularity of node a w.r.t. query q in two steps. First, it calls Function $getRCList$ to retrieve a list of nodes u that a or its descendants have reference connection RC with (lines 1-3), which are kept in $nodeList$. $getRCList$ finds those u by a depth-limited search on CT in a recursive way. Here, L is an upper limit for the distance of RC, and all variables in the above two functions have the same meaning as described in (15). Second, it computes a's popularity by (15) (lines 4-6). The similarity of node n_u can be computed by $getSimilarity()$ with slight adaptation. The detail is omitted due to space limit.

Function `getPopularity` (*Node* a, q, CT, L)
1 Let $\mathrm{d} = 1$; Let $\mathrm{p} = 0$; Let nodeList be an empty list of node labels;
2 **foreach** $v \in$ self-or-descendant(a) **do**
3 getRCList(v, CT, d, L, nodeList);
4 **foreach** $u \in nodeList$ **do**
5 p += getSimilarity(n_u,q)/(d_v * $h(a, u)$);
6 return log_ep;

Function `getRCList` (*Node* v, CT, d, L, $nodeList$)
1 **if** ($d \leq L$) **then**
2 Let tempList = <CT.getCList(), d>;
3 nodeList.merge(tempList); d++;
4 **foreach** $Node\ n \in tempList$ **do**
5 getRCList(n, CT, d, L, nodeList);
6 return;

In order to locate the descendants of a efficiently (see line 2 of $getPopularity$), we build a *trie* data structure to store the keys of connection table CT, i.e., the dewey labels of the nodes that have direct incoming IDRef edges; thus, it costs only O(m) time to find the descendant of a node a, where m is the depth of XML data in worst case.

The above search methods can be gracefully adapted to handle unstructured data, which provide an easy way to incorporate our ranking techniques in a standard text indexing system to handle both unstructured and semi-structured data.

8 Experiments

We have performed comprehensive experiments to compare the effectiveness, efficiency, and scalability of XReal with SLCA and XSeek, all implemented in Java and run on a 3.6 GHz Pentium 4 machine with 1 GB RAM running Windows XP. We tested both synthetic and real data sets. XMark [3] is used as synthetic data set; WSU, eBay from [2] and DBLP are used as real data sets. The size of the data, the three indexes, and the connection table CT (proposed in Section 7.1), and the total indexing time are reported in Table 1. Berkeley DB Java Edition [1] is used to store the keyword inverted lists, frequency table, and connection table CT.

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

	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

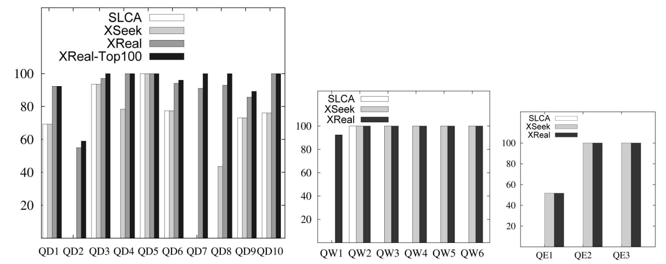
Fig. 2. Test on inferring the *search for* node.

8.1 Search Effectiveness

8.1.1 Infer the Search for Node

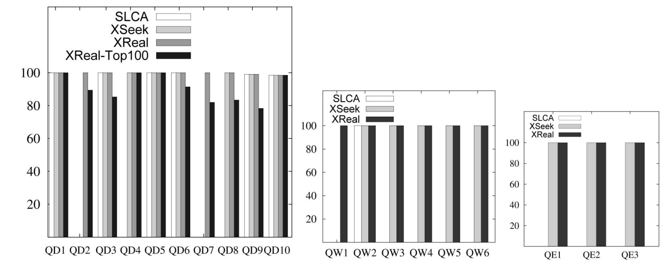
To test XReal's accuracy in inferring the desired search for node, we make a survey of 20 keyword queries, most of which do not contain an explicit search for node. To get a fairly objective view of user search intentions in real world, we post this survey online and ask for 46 people to write down their 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 Fig. 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 fourth column contains the search for node inferred by XReal while the fifth 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 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



(a) (b) (c)

Fig. 3. Precision comparison (percent). (a) DBLP, (b) WSU, and (c) EBAY.



(a) (b) (c)

Fig. 4. Recall comparison (percent). (a) DBLP, (b) WSU, and (c) EBAY.

for node, XReal returns all possible candidates (for user to decide), or returns a ranked result list for each such candidate in parallel if user interaction is not preferred. For example, 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.

8.1.2 Precision, Recall, and F-Measure

To measure the search quality, we evaluate all queries in Fig. 2, and summarize two metrics borrowed from IR field: precision and recall. 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 with an additional manual verification. As most queries on DBLP have more than 100 results, we compute XReal's *top-100* precision and top-100 recall besides the overall ones; since SLCA and XSeek do not provide any ranking function, we only compute their overall precision and recall. Besides, as there are less than 100 results for each query issued on WSU and eBay, we do not show the top-100 precision and recall in Figs. 3b and 3c, and Figs. 4b and 4c.

To evaluate XReal's performance on large real data sets, 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 has Ambiguity 2 problem, e.g., "WISE" can be the *booktitle*, *title* of inproceedings, or a value of *author*.

From Figs. 3 and 4, we have four main observations.

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). For example, in QD_3 which intends to find the 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 some 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, since "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 titles contain both keywords; but XReal correctly interprets the search intention.
2. SLCA suffers a zero precision and recall from the pure keyword value query, e.g., QD_4, QD_7, QD_8, QW_1, and QE_1-QE_3, as the SLCA results contain nothing relevant except the SLCA node. For example, for QD_8 SLCA returns the *booktitle* or *title* nodes containing "WISE," while user wants the inproceedings of "WISE" conference. In contrast, XReal 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.
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, QE_1-QE_3). XReal has a low precision on QD_2, as there are more than one person called Lei Chen in DBLP, while the users are only interested in one of them.
4. For queries that have more than 100 results on DBLP such as QD_3, QD_6-QD_9, *XReal Top-100* has a higher precision (and lower recall) than overall XReal, which indirectly proves our ranking strategy works well on large data sets.

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 Fig. 2 for each data set (plus QD_7-QD_{10}), adopting formula $F = precision * recall/(precision + recall)$ to get F-measure in Table 2. We find XReal beats SLCA and XSeek on all data sets, and achieves almost a perfect value of F which is 0.5 on WSU.

8.2 Ranking Effectiveness

To evaluate the effectiveness of XML TF*IDF alone, 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

TABLE 2
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	0.4967
EBAY	0	0.4002	0.4002	0.4002

TABLE 3
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

TABLE 4
Average CG for Top-20 Query Results

DataSet	Variants	$CG[5]$	$CG[10]$	$CG[15]$	$CG[20]$
DBLP	Before	13.47	23.56	33.32	41.05
	After	13.68	23.74	33.51	41.16
XMark	Before	12.75	23.40	31.58	38.54
	After	13.02	23.35	31.60	38.54

effectiveness for top-k answers returned, $\mathrm{k} = 40$ for DBLP (as DBLP data has very large size) and $\mathrm{k} = 20$ for others (if they do exist).

We evaluate a set of 30 randomly generated queries on DBLP, and 10 queries on WSU, eBay, and XMark, with an average of three keywords. The average values of these metrics are recorded in Table 3. 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.

Effects of popularity score. Here, we test the effects of popularity score P in distinguishing the order of the results that have comparable relevance scores. According to our empirical study, a threshold $\sigma = 2$ percent is a good choice, as usually there are more than 10 results whose ratio of relevance score difference is within 2 percent. The upper limit for the reference connection distance d_u in (15) is set to 2.

As P does not affect the overall precision of Top-K results, R-Rank and MAP which adopt a binary judgment (i.e., relevant or irrelevant) cannot be used to test the effects of P. Therefore, we adopt a comprehensive evaluation method *Cumulated Gain-based evaluation* (CG) [14], which is aware of the fact that the results are not of equal relevance to users, and allows user to specify the degree of relevance of results at a four-point scale:

1. irrelevant,
2. marginally relevant,
3. fairly relevant, and
4. highly relevant.

In this way, users can specify the degree of relevance in a more precise way. In particular, given a ranked result list retrieved by search engine, Järvelin and Kekäläinen [14] turn the list to a gained value vector G, where G[i] denotes the relevance score of the ith result given by a user; then a cumulated gain vector is defined recursively as below: $\mathrm{CG[i]} = \mathrm{CG[i-1]} + \mathrm{G[i]}$ for $\mathrm{i} > 1$, and $\mathrm{CG[1]} = \mathrm{G[1]}$. As a result, CG[k], in fact, reflects the accumulated relevance of the top-k results retrieved. Note that in this experiment, we use moderate relevance scores (i.e., 0-1-2-3) for the above four-point scale, as our users are assumed to be patient to dig down the low-ranked results.

DBLP and XMark are chosen as the data set (DBLP stores citations in the *cite* subelements of each paper), and the queries are the same as the above experiment, top-30 and top-20 results are extracted for DBLP and XMark, respectively. Seven people are asked for result relevance judgment. Table 4 shows the average CG values by these seven users for Top-K results generated by our ranking method before and after applying the popularity scoring function, where $\mathrm{K} = 5, 10, 15, 20$.

As evident from CG[5]'s value in Table 4, after taking the result popularity into account, the overall degrees of relevance of Top-5 results improve than before, for both XMark and DBLP. Similar observations for Top-10, 15, and 20 results. Moreover, by comparing the difference between CG[10] and CG[5] for DBLP, we find the results among top 10-15 ranks, which are even more relevant to user's search intention, are ranked within the top-10 results via the adjustment based on popularity score. Similar effects are reflected on XMark.

8.3 Efficiency

We compare the query response time of XReal adopting three indexes for keyword inverted list mentioned in Section 7.1, i.e., *Dup*, *DupType*, and *DupTypeNorm*, measured by the time stamp difference between when a query is issued and result is returned. Throughout Section 8, XReal refers to the one adopting *DupTypeNorm*. Fig. 5 shows the time on hot cache for queries listed in Fig. 2. *DupTypeNorm* outperforms the other two on all three real data sets, about two and four times faster than *DupType* and *Dup*, respectively. Because *DupTypeNorm* stores the dewey id, node type, and normalization factor (for data nodes) together, thus, it needs less number of index lookups to fulfill the similarity computation in (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 Fig. 5a.

8.4 Scalability

Among the existing keyword search methods [29], [12], [8], SLCA is recognized as the most efficient one so far, so we compare XReal with SLCA on DBLP and XMark. We also test the one that incorporates results' popularity computation into DupTypeNorm, denoted as *DTN+Pop*. For each data set, 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 $|K| = 2$-8 for DBLP and $|K| = 2$-5 for XMark. The response time is the average time of the corresponding 50 queries in four executions on hot cache, as shown in Fig. 6.

From Figs. 6a and 6b, we find XReal is nearly 20 percent slower than SLCA on both data sets which is acceptable, because XReal does extra search intention identification, precise result retrieval and ranking; and XReal finds extra

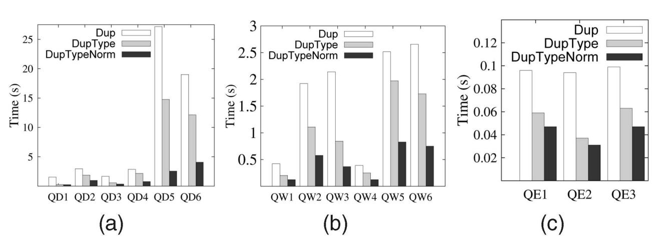


Fig. 5. Response time on individual queries. (a) DBLP, (b) WSU, and (c) eBay.

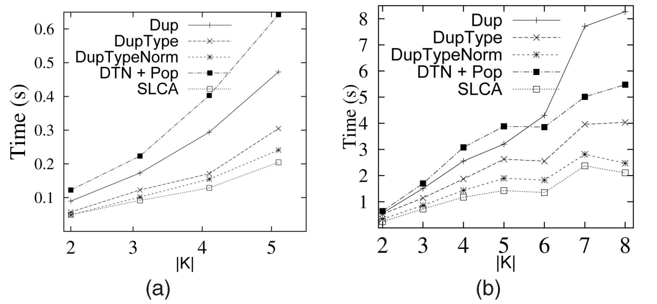


Fig. 6. Response time on different number of keywords $|K|$. (a) XMark and (b) DBLP.

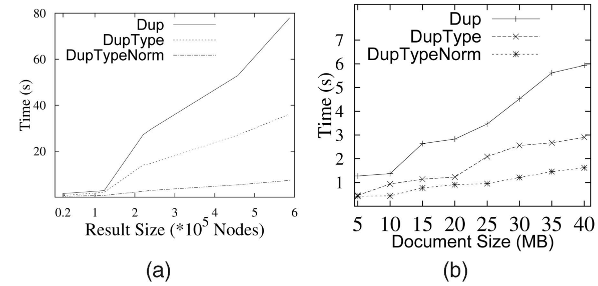


Fig. 7. Response time w.r.t. result/document size. (a) DBLP and (b) XMark.

results; so this overhead is worthwhile. We also find, the response time of each proposed index increases as $|K|$ increases. In particular, the one with *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 (Fig. 6b). $DTN + Pop$ costs about 2.2 and 3 times longer than $DupTimeNorm$ for DBLP and XMark; because XMark contains a lot of IDRef edges which need more time to compute the similarity of the nodes having reference connection with a certain query result, while the citation in DBLP is few and incomplete.

Besides, we evaluate the scalability of those indexes by drawing the relationship between the response time and query result size (in terms of number of nodes returned). A range of 15 queries with various result sizes run over DBLP, and the result is shown in Fig. 7a. 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 5 MB up to 40 MB. As shown in Fig. 7b, both *DupTypeNorm* and *DupType*'s response time increase linearly w.r.t. the data size.

9 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. We utilize statistics to infer user search intention and rank the query results. In particular, we define XML TF and XML DF, based on which we design formulae to compute the confidence level of each candidate node type to be a *search for/search via node*, and further propose a novel *XML TF*IDF similarity* ranking scheme to capture the hierarchical structure of XML data. Lastly, the popularity of a query result (captured by IDRef relationships) is considered to handle the case that multiple results have comparable relevance scores. In future, we would like to extend our approach to handle the XML document conforming to a highly recursive schema as well.

ACKNOWLEDGMENTS

This paper was partially supported by 863 National High-Tech Research Plan of China (No: 2009AA01Z133), National Science Foundation of China (NSFC) (No. 60903056), Key Project in Ministry of Education (No: 109004) and SRFDP Fund for the Doctoral Program (No. 20090004120002). Jiaheng Lu was the contact author.

REFERENCES

[1] Berkeley DB, http://www.sleepycat.com/, 2010.
[2] http://www.cs.washington.edu/research/xmldatasets, 2010.
[3] http://www.xml-benchmark.org/, 2010.
[4] S. Amer-Yahia, L.V.S. Lakshmanan, and S. Pandit, "Flexpath: Flexible Structure and Full-Text Querying for XML," *Proc. ACM SIGMOD Conf.,* 2004.
[5] Z. Bao, B. Chen, T.W. Ling, and J. Lu, "Effective XML Keyword Search with Relevance Oriented Ranking," *Proc. IEEE Int'l Conf. Data Eng. (ICDE),* pp. 517-528, 2009.
[6] D. Carmel, Y.S. Maarek, M. Mandelbrod, Y. Mass, and A. Soffer, "Search XML Documents via XML Fragments," *Proc. ACM SIGIR,* pp. 151-158, 2003.
[7] S. Cohen, Y. Kanza, B. Kimelfeld, and Y. Sagiv, "Interconnection Semantics for Keyword Search in XML," *Proc. ACM Int'l Conf. Information and Knowledge Management (CIKM),* pp. 389-396, 2005.
[8] S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv, "XSEarch: A Semantic Search Engine for XML," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* pp. 45-56, 2003.
[9] N. Fuhr and K. Großjohann, "XIRQL: A Query Language for Information Retrieval in XML Documents," *Proc. ACM SIGIR,* pp. 172-180, 2001.
[10] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram, "XRANK: Ranked Keyword Search over XML Documents," *Proc. ACM SIGMOD Conf.,* 2003.
[11] H. He, H. Wang, J. Yang, and P.S. Yu, "Blinks: Ranked Keyword Searches on Graphs," *Proc. ACM SIGMOD Conf.,* pp. 305-316, 2007.
[12] V. Hristidis, N. Koudas, Y. Papakonstantinou, and D. Srivastava, "Keyword Proximity Search in XML Trees," *IEEE Trans. Knowledge and Data Eng.,* vol. 18, no. 4, pp. 525-539, Apr. 2006.
[13] V. Hristidis, Y. Papakonstantinou, and A. Balmin, "Keyword Proximity Search on XML Graphs," *Proc. IEEE Int'l Conf. Data Eng. (ICDE),* pp. 367-378, 2003.
[14] K. Järvelin and J. Kekäläinen, "Cumulated Gain-Based Evaluation of IR Techniques," *ACM Trans. Information Systems,* vol. 20, pp. 422-446, 2002.
[15] R. Jones, B. Rey, O. Madani, and W. Greiner, "Generating Query Substitutions," *Proc. Int'l Conf. World Wide Web (WWW),* 2006.
[16] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar, "Bidirectional Expansion for Keyword Search on Graph Databases," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* pp. 505-516, 2005.
[17] M. Ley DBLP, http://www.informatik.uni-trier.de/ley/db/, 2009.
[18] G. Li, J. Feng, J. Wang, and L. Zhou, "Effective Keyword Search for Valuable LCAs over XML Documents," *Proc. ACM Int'l Conf. Information and Knowledge Management (CIKM),* pp. 31-40, 2007.
[19] 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," *Proc. ACM SIGMOD Conf.,* 2008.
[20] W.S. Li, K.S. Candan, Q. Vu, and D. Agrawal, "Retrieving and Organizing Web Pages by Information Unit," *Proc. Int'l Conf. World Wide Web (WWW),* 2001.
[21] Y. Li, C. Yu, and H.V. Jagadish, "Schema-Free XQuery," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* 2004.
[22] Z. Liu and Y. Chen, "Identifying Meaningful Return Information for XML Keyword Search," *Proc. ACM SIGMOD Conf.,* 2007.

[23] Z. Liu and Y. Chen, "Reasoning and Identifying Relevant Matches for XML Keyword Search," *Proc. Int'l Conf. Very Large Data Bases (VLDB)* vol. 1, no. 1, pp. 921-932, 2008.

[24] G. Salton and M.J. McGill, *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., 1986.

[25] A. Schmidt, M.L. Kersten, and M. Windhouwer, "Querying XML Documents Made Easy: Nearest Concept Queries," *Proc. IEEE Int'l Conf. Data Eng. (ICDE)*, pp. 321-329, 2001.

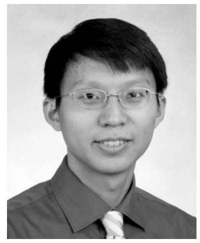
[26] C. Sun, C.Y. Chan, and A.K. Goenka, "Multiway SLCA-Based Keyword Search in XML Data," *Proc. Int'l Conf. World Wide Web (WWW)*, pp. 1043-1052, 2007.

[27] A. Theobald and G. Weikum, "The Index-Based XXL Search Engine for Querying XML Data with Relevance Ranking," *Proc. Int'l Conf. Extending Database Technology (EDBT)*, 2002.

[28] V. Vesper, Let's Do Dewey, http://www.mtsu.edu/vvesper/dewey.html, 2009.

[29] Y. Xu and Y. Papakonstantinou, "Efficient Keyword Search for Smallest LCAs in XML Databases," *Proc. ACM SIGMOD*, pp. 537-538, 2005.

Zhifeng Bao received the BS degree from the Department of Computer Science, School of Computing, National University of Singapore, in 2006. He is currently working toward the PhD degree in the Department of Computer Science, School of Computing, National University of Singapore. His research interests include XML structured query processing, XML keyword search, and data stream analysis.



Jiaheng Lu received the PhD degree from the National University of Singapore and his advisor is Prof. Tok Wang Ling. He is an associate professor in Renmin University of China. His research interests include XML data management, keyword search, and cloud data management. He has published more than 20 papers in top conferences and journals. He serves as a program member in top conferences including SIGMOD and VLDB.

Tok Wang Ling received the PhD degree in computer science from the University of Waterloo (Canada). He is a professor of the Department of Computer Science at the National University of Singapore. His research interests include data modeling, ER approach, normalization theory, and semistructured data model and XML query processing. He has published more than 180 papers, coauthored a book, and coedited nine conference proceedings. He is an ACM distinguished scientist and a senior member of the IEEE.

Bo Chen received the BS and MA degrees from the Department of Computer Science, School of Computing, National University of Singapore, in 2005 and 2008, respectively. His research interests include XML structured query processing and XML keyword search.