XSeek: A Semantic XML Search Engine Using Keywords

Ziyang Liu
Arizona State University
Ziyang.Liu@asu.edu

Jeffrey Walker
Arizona State University
Jeffrey.Walker@asu.edu

Yi Chen
Arizona State University
Yi@asu.edu

1. MOTIVATION

Keyword search provides a user-friendly information discovery mechanism for web users to easily access XML data without the need of learning a structured query language or studying possibly complex and evolving data schemas. However, due to the lack of expressivity and inherent ambiguity, there are two main challenges in performing keyword search on XML data intelligently.

1. First, unlike XQuery, where the connection among data nodes matching a query is specified precisely using variable bindings and *where* clauses, we need to automatically connect the keyword matches in a meaningful way.

2. Second, unlike XQuery, where the data nodes to be returned are specified using a *return* clause, we should effectively identify the desired return information.

Several attempts have been made to address the first challenge [3, 2, 9, 7] by selecting and connecting keyword matches through a variant concept of lowest common ancestor, named as *VLCA* (such as SLCA [9], MLCA [7], interconnection [2], etc.). However, it is an open problem of how to automatically and effectively infer *return nodes*, the names of the data nodes that are the goal of user searches.

There are two baseline approaches for determining return nodes adopted in the existing work. One is to return the subtrees rooted at VLCA nodes [3, 9], named as *Subtree Return*. Alternatively, we can return the paths in the XML tree from each VLCA node to its descendants that match an input keyword, as described in [1, 4], named as *Path Return*. However, neither approach is effective in identifying return information as the following examples show.

Let us look at the sample queries listed in Figure 2 on XML data in Figure 1(a). For Q_1, it is likely that a user is interested in the information about `Rockets`. Both *Subtree Return* and *Path Return* first compute the VLCA of keyword matches, which is the match node itself: `Rockets` (0.2.0.0), then output this node. However, to echo print the user input without any additional information is not informative. Ideally, we would like to return the subtree rooted at the `team` node with ID 0.2 for information about `Rockets`.

Now let's consider Q_2 and Q_3. By issuing Q_2, the user is likely to be interested in the information about the player whose name is `Mutombo` and who is a `center` in a team. Therefore the subtree rooted at the `player` node with ID 0.2.4.0 is desired output. In contrast, Q_3 indicates that the user is interested in a particular piece of information: the `position` of `Mutombo`.

As we can see, the input keyword can specify a *predicate* for the search, or specify desired *return* nodes. However, existing approaches fail to differentiate these two types of keywords. In particular, *Path Return* approach returns the paths from the VLCA `player` node (0.2.4.0) to `Mutombo` and to `center` for Q_2, and the path from `player` (0.2.4.0) to `Mutombo` and `position` for Q_3, respectively. On the other hand, since Q_2 and Q_3 have the same VLCA node, `player` (0.2.4.0), *Subtree Return* outputs the subtree rooted at this node for both queries, though the user indicates that only `position` information is of interest in Q_3.

Now let's look at a more complex query Q_4, intending to find information about the player who is a `center` in the `team Rockets`. The desired query result is shown in Figure 1(b). The *Subtree Return* approach outputs the whole tree rooted at `team` (0.2), and requires the user him/herself to search the relevant player information in this big tree. On the other hand, the *Path Return* approach outputs the path from `team` to `Rockets` and to `center`, without providing any additional information about `player` (0.2.4.0).

As we can see from the above sample queries, existing approaches fail to effectively identify relevant return nodes and suffer a low precision and/or recall.

The only work that has considered the problem of identifying return nodes is [5, 6]. Both require the schema information, and require users and/or system administrators to specify the schema of output information.

In this demo, we present an XML keyword search system XSeek [8] that addresses the problem of identifying return nodes for XML keyword search. It allows users to search XML documents using keywords, and generates meaningful return information as exemplified in the above examples without schema or user preference solicitation.

2. CONTRIBUTIONS

The XSeek system has several significant features compared with existing keyword search engines for XML data.

1. To the best of our knowledge, XSeek is the first XML

VLDB '07, September 23-28, 2007, Vienna, Austria.
Copyright 2007 VLDB Endowment, ACM 978-1-59593-649-3/07/09.

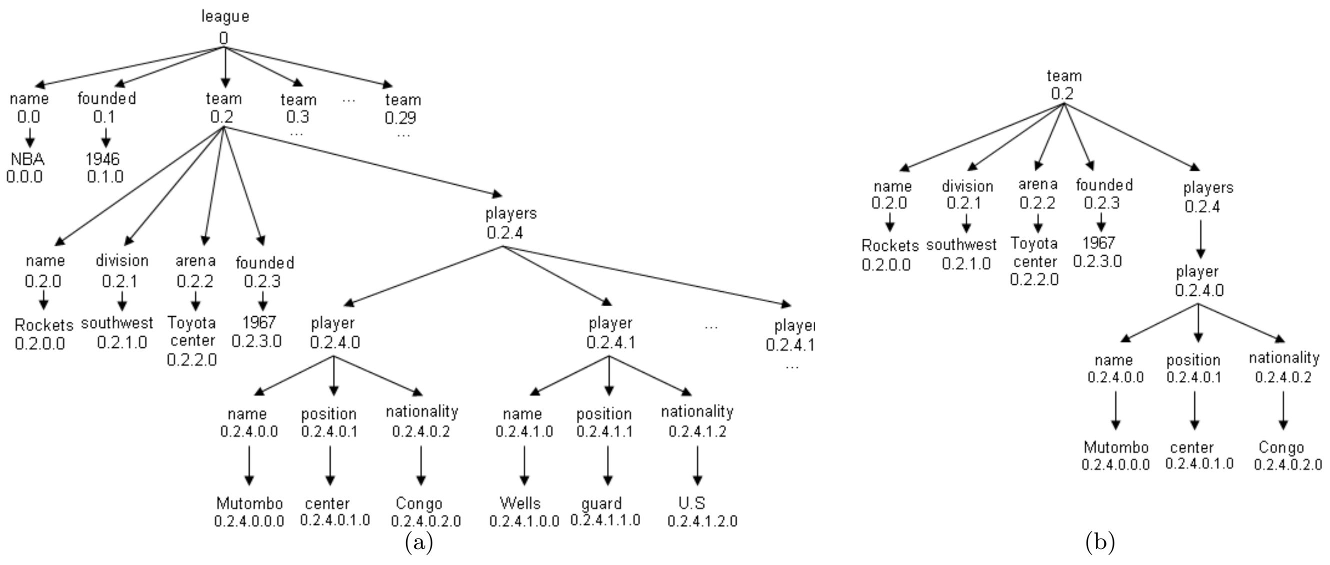

Figure 1: Sample XML Document(a) and Search Result for Q_4(b)

keyword search engine that automatically infers desirable return node information to form query results.

2. XSeek identifies and generates return nodes of two types: *explicit return nodes* that are optionally specified in the keywords; *implicit return nodes* that are not part of the input keywords, but can be inferred from XML data.

3. To determine explicit return nodes, XSeek analyzes the input keyword patterns and classifies keywords into two categories: the ones that specify search predicates, and the ones that indicate the return information that the user is seeking for.

4. To infer implicit return nodes, XSeek analyzes the structure of XML data and differentiates three types of information: entities in the real world, attributes of entities, and connection nodes.

5. Data nodes that match predicates or inferred return nodes are output as query results.

6. Experimental results show that XSeek generates search results with significantly improved precision and recall compared with *Subtree Return* and *Path Return* approaches with good scalability.

3. INFERRING KEYWORD SEARCH SEMANTICS

Identifying and Connecting Keyword Matches. XSeek adopts the approach proposed in [9] for defining VLCA nodes and connecting keyword matches through their VLCA nodes. [1] An XML node is named as a *VLCA* node if its subtree contains matches to every keyword in the query, and none of its descendants contains every keyword in its subtree. Keyword matches in a subtree rooted at a VLCA node are considered as closely related and are connected through the VLCA node; while the matches that are not descendants of any VLCA node are determined as irrelevant and discarded.

For example, consider Q_3 on XML data in Figure 1(a). There is one match to keyword `Mutombo`: 0.2.4.0.0.0, and two matches to `position`: 0.2.4.0.1 and 0.2.4.1.1. According to definition, `player` (0.2.4.0) is the only VLCA node, which connects closely related matches: `Mutombo` (0.2.4.0.0.0) and `position` (0.2.4.0.1). Note that though `players` (0.2.4) has matches to both keywords in its subtree, it is not a VLCA node, and `position` (0.2.4.0.1) is considered as irrelevant.

As illustrated in Section 1, outputting match nodes and their connection or the whole subtrees rooted at VLCA nodes are not desirable in many cases. Next we discuss how to infer meaningful return nodes for XML keyword search.

Analyzing XML Data Structure. XSeek analyzes the structure of XML data, differentiates nodes representing entities from nodes representing attributes, similar as the *Entity-Relationship* model in relational databases. We believe that by issuing a query a user would like to find out information about entities along with their relationships in a document. Therefore the entities related to the input keywords are considered in determining return nodes.

For example, for the XML data in Figure 1(a), conceptually we can recognize two types of entities: `team` and `player`. Entity `team` has `name`, `division`, `arena` and `founded` as attributes. `player` has attributes `name`, `position` and `nationality`. The relationships between entities are represented by the paths connecting them. For example, a `team`

Q_1	Rockets
Q_2	Mutombo, center
Q_3	Mutombo, position
Q_4	team, Rockets, center

Figure 2: Sample Keyword Searches

[1]Alternatively, other approaches to compute VLCA nodes such as MLCA [7], Interconnection [2] can be incorporated seamlessly into XSeek for identifying and connecting relevant keyword matches.

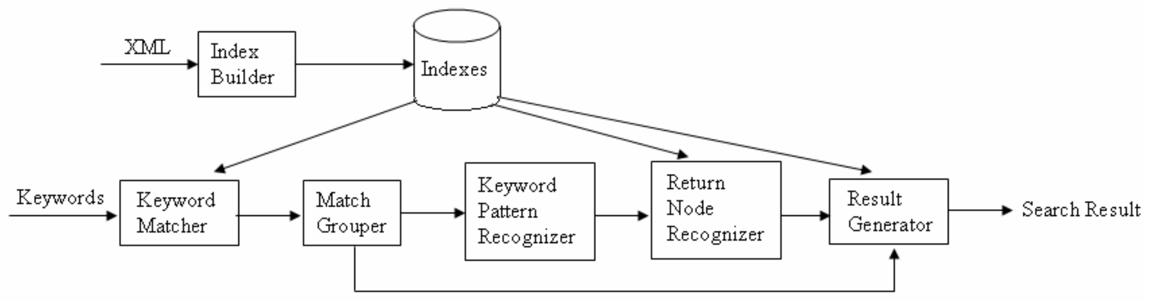

Figure 3: Architecture of XSeek

has one or more `players`. By issuing Q_1 `Rockets`, it is likely that the user would like to find out the information about the real world entity that `Rockets` corresponds to.

However, since XML data may be designed and generated by autonomous sources, the entity and attribute information may not be directly available. We use the following inference for data node categories.

1. If a node has siblings of the same name, then this indicates a many-to-one relationship with its parent node, and is considered to represent an *entity*.
2. If a node does not have siblings of the same name, and it has one child, which is a value, then it is considered to represent an *attribute*.
3. A node is a *connection* node if it represents neither an entity nor an attribute.

The node relationship and node categories can be detected according to the schema (if available) or the structural summary of the data.

For the XML data in Figure 1(a), since `team` has a many-to-one relationship with its parent node `league`, we infer that `team` represents an entity that has a relationship with `league`. Its children `name`, `division`, `arena`, and `founded` are considered as attributes of a `team` entity. On the other hand, `players` is considered as a connection node.

Analyzing Keyword Patterns. XSeek also analyzes keyword patterns to determine return information. It classifies input keywords into two categories: predicates and return nodes. Some keywords indicate *predicates* that restrict the search, corresponding to the *where* clause in XQuery. Others specify the desired output type, referred as *explicit return nodes*, corresponding to the *return* clause in XQuery.

Recall Q_2 and Q_3 in Figure 2, where the keyword matches are connected in the same way. However, different patterns of these two queries imply different user intensions. Q_2 searches information about `Mutombo` and `center`. Q_3 searches the `position` information of `Mutombo`. Intuitively, both `Mutombo` and `center` are considered to be predicates, while `position` in Q_3 indicates a return node.

The immediate question is how to infer predicates and return nodes in the input keywords. Recall that in a structured query language such as XQuery or SQL, typically a predicate consists of a pair of name and value, while a return clause only specifies names without value information (which is expected to be query results). For example, consider an SQL query: `select position from DB where name = "Mutombo"`. Based on this observation, we make the following inference for keyword categories.

1. If an input keyword k_1 matches a node name u, and there does not exist an input keyword k_2 matching a node value v, such that u is an ancestor of v, then we consider k_1 as an (explicit) *return node*.
2. A keyword that is not a return node is treated as a *predicate*.

For example, in Q_2, `center` is considered as a predicate since it matches a value (0.2.4.0.1.0). Similarly, `Mutombo` in both Q_2 and Q_3 are considered as a predicate. `team` in Q_4 is also inferred as a predicate since it matches a name node (0.2) which has a descendant value node (0.2.0.0) that matches another keyword `Rockets`. On the other hand, `position` in Q_3 is considered as a return node since it matches the name of two nodes (0.2.4.0.1, 0.2.4.1.1), neither of which has any descendant value node matching another keyword in Q_3.

Generating Search Results. Once we have identified inherent entities and attributes in the data and possible predicates and explicit return nodes in the input keywords, XSeek determines the return nodes and outputs data nodes that match search predicates and return nodes as query results.

As we have seen, return nodes can be *explicitly* inferred from the input keywords for some queries, such as `position` in Q_3. For queries where all the input keywords are considered as predicates, no return nodes can be inferred from the keywords themselves, such as Q_2. In this case, we believe that the user is interested in the general information about the entities related to the search. We define *relevant entities* as the entities in the data that are on the path from a VLCA node to each match node, as well as the lowest ancestor entity of a VLCA. Relevant entities are considered as *implicit* return nodes when the input keywords do not have return nodes specified, whose attribute information will be output.

In Q_2, since both `Mutombo` and `center` are inferred as predicates, there is no return nodes specified in the keywords. We first identify the `player` node (0.2.4.0) as the VLCA node, which is then determined to be the only relevant entity and the implicit return node. The name and attributes of `player` (0.2.4.0) are output as query results. Similarly, for Q_1, we infer `team` as an implicit return node since it is a relevant entity and no explicit return nodes can be inferred from the query. For Q_4, relevant entities `team` (0.2) and `player` (0.2.4.0) are considered to be implicit return nodes, and their names and attributes are returned as query result.

4. SYSTEM ARCHITECTURE AND IMPLEMENTATION

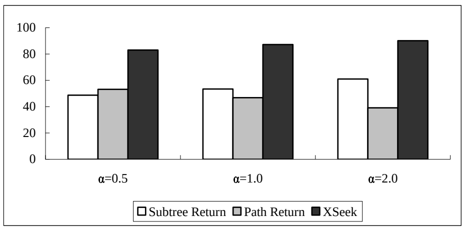

Figure 4: F-measure of Test Queries

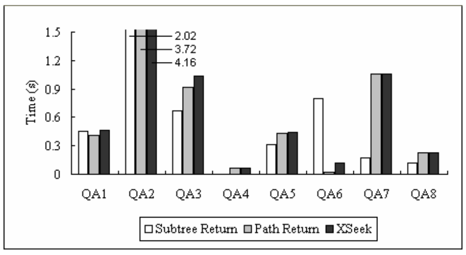

Figure 5: Processing Time on Benchmark Data

We have implemented XSeek in C++. It takes keywords as input, and returns the information in the XML documents that matches the predicates and inferred return nodes.

The system architecture of XSeek is presented in Figure 3. The *Index Builder* parses the input XML data, infers the inherent entities and attributes in the data, and builds indexes for retrieving the information about node category, parent, and children. Once a user issues a query, *Keyword Matcher* accesses the indexes and efficiently retrieves data matches to each keyword. *Match Grouper* connects closely related keyword matches together as a group according to their VLCA nodes [9]. Then, for every group of keyword matches, *Keyword Pattern Recognizer* analyzes the matches and categorizes input keywords as predicates or explicit return nodes. The *Return Node Constructor* generates return nodes for the query, which can be explicit return nodes inferred from input keywords, or implicit return nodes inferred from keyword matches and entities in the data indexes. Finally, the *Result Generator* outputs the search result by returning XML subtrees rooted at the lowest entity ancestors of the VLCA nodes, which contain the data nodes that match query predicates and return nodes.

We have empirically evaluated XSeek compared with two approaches *Subtree Return* and *Path Return*, introduced in Section 1. Figure 4 shows the average F-measure of three approaches across 24 keyword search queries over three different data sets, with different weights on precision and recall. Figure 5 shows the processing time of three approaches on XMark (http://monetdb.cwi.nl/xml/) data set of size 25MB. As we can see, XSeek significantly outperforms the *subtree return* and *path return* approaches in search quality, with reasonable processing time.

5. DEMONSTRATION

What is the goal of the demo? Through the demo, we present one challenge of XML keyword search that has been neglected in the literature: how should we identify the desired return nodes, analogous to inferring a "return" clause in XQuery from input keywords. The development and demonstration of XSeek show a promising step in identifying return nodes for XML keyword search, which effectively improves the search quality compared with existing approaches.

What will be shown in the demo? XSeek has a web-based user interface (http://XSeek.asu.edu/) which allows users to specify an XML document and keyword searches for retrieval. We also provide several sample XML documents, including geographical data (Mondial), course information(WSU), DBLP, Sigmod Record data, Shakespeare's plays, with some sample queries. Rather than outputting the whole subtrees or the paths that contain the keyword matches, XSeek intelligently infers desired return nodes by analyzing XML documents and input keyword patterns without eliciting user preference.

In the demonstration, we will also present the search results produced by *Subtree Return* and *Path Return*, as well as the original XML document fragments related to the search for comparison purpose. The user can provide feedback to the system by scoring the results returned by each approach and/or specify the desired search results. The user feedback will be collected and analyzed for improving the effectiveness of XSeek. Furthermore, statistics information such as the input document size, search result size, and the processing time will also be available.

Besides the demonstration of the implementation of XSeek, we will discuss and justify in more detail the structures and algorithms we employ in the system. In addition, performance evaluation compared with *Subtree Return* and *Path Return* approaches over a comprehensive test set will be exhibited.

6. REFERENCES

[1] G. Bhalotia, C. Nakhe, A. Hulgeri, S. Chakrabarti, and S. Sudarshan. Keyword Searching and Browsing in Databases using BANKS. In *ICDE*, 2002.

[2] S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv. XSEarch: A semantic Search Engine for XML, 2003.

[3] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRANK: Ranked Keyword Search over XML Documents. In *Proceedings of SIGMOD*, pages 16–27, 2003.

[4] V. Hristidis, N. Koudas, Y. Papakonstantinou, and D. Srivastava. Keyword Proximity Search in XML Trees. *IEEE Transactions on Knowledge and Data Engineering*, 18(4), 2006.

[5] V. Hristidis, Y. Papakonstantinou, and A. Balmin. Keyword Proximity Search on XML Graphs, 2003. In ICDE.

[6] G. Koutrika, A. Simitsis, and Y. E. Ioannidis. Précis: The essence of a query answer. In *ICDE*, page 69, 2006.

[7] Y. Li, C. Yu, and H. V. Jagadish. Schema-Free XQuery. In *VLDB*, 2004.

[8] Z. Liu and Y. Chen. Identifying Meaningful Return Information for XML Keyword Search. In *Proceedings of SIGMOD*, 2007.

[9] Y. Xu and Y. Papakonstantinou. Efficient Keyword Search for Smallest LCAs in XML Databases. In *Proceedings of SIGMOD*, pages 527–538, 2005.