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

ABSTRACT

We present XSeek, a keyword search engine that enables users to easily access XML data without the need of learning XPath or XQuery and studying possibly complex data schemas. XSeek addresses a challenge in XML keyword search that has been neglected in the literature: how to determine the desired return information, analogous to inferring a "return" clause in XQuery. To infer the search semantics, XSeek recognizes possible entities and attributes in the data, differentiates search predicates and return specifications in the keywords, and generates meaningful search results based on the analysis.

1. INTRODUCTION

Keyword search provides a user-friendly information discovery mechanism for web and scientific 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. 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. 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, referred as *VLCA* in this paper (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.

*This research is partially supported by NSF IIS-0612273.

VLDB '07, September 23-28, 2007, Vienna, Austria.

There are two baseline approaches for determining return nodes adopted in the existing works. 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 shown in the following examples.

Let us look at the sample queries listed in Figure 2 on XML data in Figure 1(a). For Q_1 (`Rockets`), 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.

Consider Q_2 and Q_3. By issuing Q_2 (`Mutombo, center`), 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 an desired output. In contrast, Q_3 (`Mutombo, position`) indicates that the user is interested in a particular piece of information: the `position` of `Mutombo`.

As we can see, input keywords can specify *predicates* for a search, or specify desired *return* nodes. However, existing approaches fail to differentiate these two types of keywords. In particular, the *Path Return* approach returns the paths from the VLCA `player` node (0.2.4.0) to `Mutombo` and `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, even though the user indicates that only `position` information is of interest in Q_3.

Now look at a more complex query Q_4 (`team, Rockets, center`), 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, *Path Return* 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. They may return too much information and therefore require users to manually exam large trees (low precision); or return

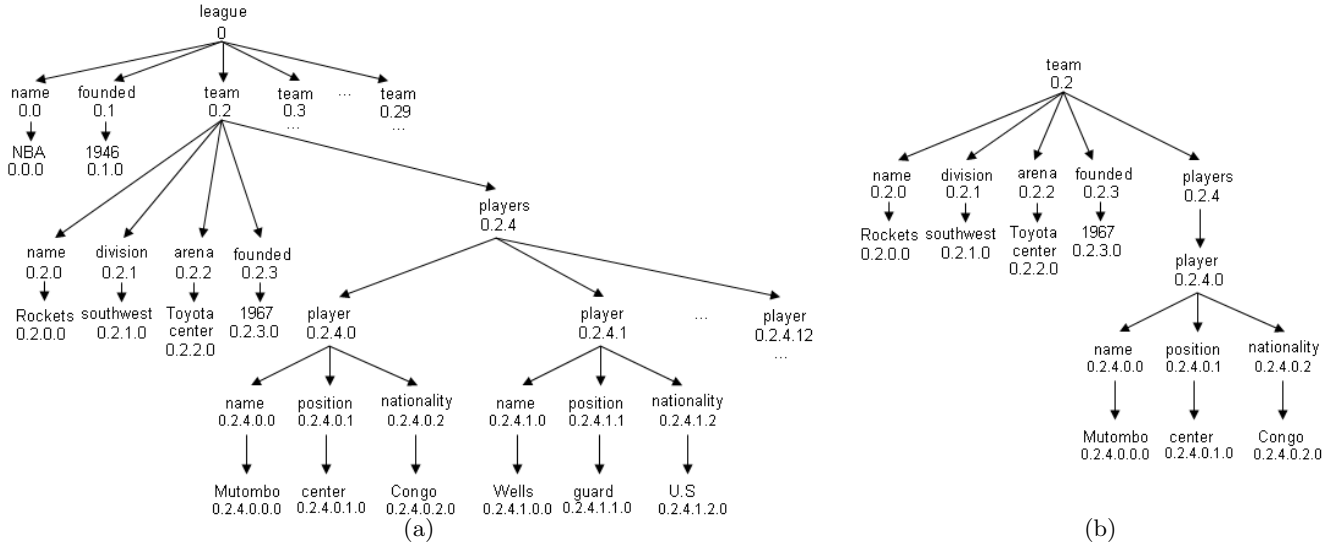

Figure 1: Sample XML Document(a) and Search Result for Q_4(b)

too little information to be informative (low recall).

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

In this demonstration, we present an XML keyword search engine, XSeek [8], that addresses the problem of identifying meaningful return nodes for XML keyword search without schema information or user preference solicitation. XSeek has several salient features compared with existing keyword search engines for XML data.

1. To the best of our knowledge, XSeek is the first XML keyword search engine that automatically infers desirable return nodes to form query results.

2. XSeek determines return nodes of two types: *explicit return nodes* that can possibly be inferred from input keywords; *implicit return nodes* that are not part of the keywords, but can be identified from XML data.

3. To infer explicit return nodes, XSeek analyzes keyword match 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 identify 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. Query results consist of data nodes that match predicates or inferred return nodes.

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

2. INFERRING 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 the definition, `player` (0.2.4.0) is the only VLCA node, which connects relevant 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` (0.2.4.1.1) is considered as irrelevant.

As illustrated in Section 1, simply outputting match nodes and their connection or outputting 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

[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.

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

Figure 2: Sample Keyword Searches

attributes. `player` has attributes `name`, `position` and `nationality`. The relationships between entities are represented by the paths connecting them. For example, a `team` has one or more `player`s. 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 such a node 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 Match Patterns. XSeek also analyzes keyword match patterns to determine return information. It classifies input keywords into two categories: predicates and return nodes. Some keywords specify *predicates* that restrict the search, corresponding to the *where* clause in XQuery. Others indicate 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, `center` in Q_2 is considered as a predicate since it matches a value (0.2.4.0.1.0). Similarly, `Mutombo` in Q_2 and Q_3 is 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 the inherent entities and attributes in the data and possible predicates and explicit return nodes in the input keywords, XSeek determines the return nodes, and then outputs the 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 explicitly 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 output as query result.

3. SYSTEM ARCHITECTURE AND IMPLEMENTATION

The system architecture of XSeek is presented in Figure 3. The *Data Analyzer* parses the input XML data, and infers the inherent entities and attributes in the data. The *Index Builder* constructs indexes for efficiently retrieving the information about node category, parent, and children. Once a user issues a query, *Keyword Matcher* accesses the indexes to retrieve 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, the *Keyword Analyzer* categorizes input keywords as predicates or explicit return nodes. The *Return Node Recognizer* 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 indexes. Finally, the *Result Generator* out-

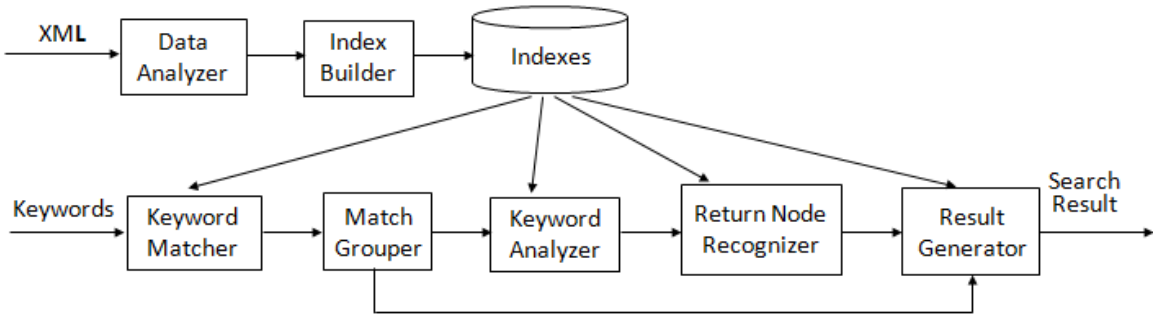

Figure 3: Architecture of XSeek

puts the search result by returning XML data nodes that match query predicates and return nodes.

We have implemented XSeek in C++. It takes keywords and XML documents as input, and returns XML nodes that match the inferred search predicates and return nodes. The empirical study of XSeek in comparison with two approaches *Subtree Return* and *Path Return* is presented in [8]. Experiments show that XSeek significantly outperforms *Subtree Return* and *Path Return* in search quality measured by precision, recall and F-measure, with efficiency.

4. DEMONSTRATION

XSeek has a web-based interface (http://xseek.asu.edu/) which allows users to specify an XML document and keyword searches for retrieval. Various sample XML documents, such as sports, DBLP, Shakespeare's plays, are provided. 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 keyword match patterns without eliciting user preference. A snapshot of XSeek is shown in Figure 4.

In the demonstration, we also present the search results produced by *Subtree Return* and *Path Return*, as well as the original XML document fragments for comparison purpose.

The user can provide feedback by scoring the results returned by each approach and/or specifying desired search results. The user feedback is collected and analyzed for improving the effectiveness of XSeek. Furthermore, statistics information, such as processing time, the number of queries so far, average score of each approach, is also available.

Besides demonstrating the functionalities of XSeek, we present the design, architecture and algorithms we employ in developing the system. In addition, performance evaluation compared with *Subtree Return* and *Path Return* over a comprehensive test set is exhibited.

Figure 4: A Snapshot of XSeek

5. 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. In *VLDB*, 2003.

[3] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRANK: Ranked Keyword Search over XML Documents. In *SIGMOD*, 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.

[6] G. Koutrika, A. Simitsis, and Y. E. Ioannidis. Précis: The essence of a query answer. In *ICDE*, 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 *SIGMOD*, 2007.

[9] Y. Xu and Y. Papakonstantinou. Efficient Keyword Search for Smallest LCAs in XML Databases. In *SIGMOD*, 2005.