ABSTRACT

The unprecedentedly large volume of semi-structured data has exacerbated the need for an easy-to-use query interface for semi-structured data sources. Natural language interfaces and keyword search techniques that take advantage of the data set structure make it very easy for ordinary users to access the data. In this paper, we introduce the important challenges that lie in the way of building an effective and efficient keyword and/or natural language query interface for semi-structured databases. We show that the current approaches to this problem rely heavily on heuristics that are intuitively appealing but ultimately ad hoc. Their assumptions are valid for some domains, database designs, and/or schema structures but they are not correct in general. Thus, they often retrieve false positive answers, overlook correct answers, and cannot rank answers appropriately. We overview our proposed approach called coherency ranking. It is based on the concept of data dependency and information theory and takes advantage of the deep structural information in the database. This makes the results of a keyword/natural language query invariant under schema reorganization. We developed efficient algorithms to discover the useful structural information and rank the answers to a keyword/natural language query. Our extensive user studies with real-world XML data sets showed that coherency ranking provides better precision, recall, and ranking quality than all previous approaches. Coherency ranking can also be used for keyword queries over relational data and information integration. We also explore our plan for future extensions of the coherency ranking framework.

1. INTRODUCTION

We are witnessing the emergence of huge volumes of semi-structured data in domains ranging from science [1] to business [2] to web [12], and text databases [16]. As the concepts such as schemas and query languages are alien to most users in these domains, they need an easy query interface. Visual interfaces are excellent for many situations [48], but must be customized individually for each domain, which is very expensive. Keyword search [15, 18, 19, 20, 24, 28, 41, 42, 47, 49, 50, 5] and natural language interfaces [30] have been proposed as less costly options; the challenge is how to find the data most closely related to the user’s query, since the query is not framed in terms of the data’s actual structure. Ideally, the query answer must include all portions of the data that are related to the query (high recall), and nothing unrelated (high precision).

As availability of structural information creates the opportunity to return more specific portions of data, compared to the traditional document-based retrieval [37], it imposes the following challenges.

CHALLENGE 1. The keyword query processing method must establish a framework that defines the degree of relatedness of the data elements to the input query, based on the available structural information.

We have analyzed the current keyword and natural language query processing proposals. As explained in the following section, the current XML keyword and natural language query answering approaches rely on heuristics that assume certain properties of the DB schema. Though these heuristics are intuitively reasonable, they are sufficiently ad hoc that they are frequently violated in practice, even in the highest-quality schemas. Thus, current approaches suffer from low precision, low recall, or both. As relatedness is a matter of degree, the system must rank the candidate answers based on their structural properties so that the most relevant answers appear at the beginning of the list. Current approaches either do not rank their query answers or use a very simple ranking heuristic (e.g., smallest answer first). This is undesirable because when queries do not precisely describe what the user wants, a good ranking of answers greatly improves the system’s usability [40]. In domains where information-retrieval-style statistics such as term frequency (TF) and inverse document frequency (IDF) [37] or PageRank [11] can be helpful in ranking query answers, the ranking schemes must be combined with such measures to improve the precision of query answers.

Moreover, keyword search techniques for data-centric and text-centric XML have been developed by two separate communities of researchers. However, there are many real-world semi-structured data sets that have both text-centric and data-centric data elements [12]. Researchers in the IR community use content-based ranking schemes to provide effective keyword search interface to text-centric XML collections [47]. These methods do not take advantage of the structural information at the schema level. The current key-
word search proposals for data-centric semi-structured data sources cannot boost the effectiveness of keyword search over text-centric semi-structure data sources, as their assumptions about the schemas are not valid for these data collections. The ideal framework must be able to apply the schema information for keyword search over text-centric semi-structured data sets.

**Challenge 2.** Huge volumes of semi-structured data are created from unstructured data sources using information extraction techniques [12, 13]. These data sets have heterogeneous schemas [15]. More importantly, their schema information is incomplete [12].

Since the current proposals use shallow structural information they cannot be effectively used to search over these data collections. For instance, some proposals [15, 28, 31] use the labels of data elements to measure the relatedness of the portions of the data to the input query. However, these data collections have many schema elements with anonymous labels [12]. The available methods also assume that the schema elements with different labels representing different entities and/or attributes. However, the same (or close) attributes and/or entities are often labeled differently. For instance, the `paper` and `article` entities are representing almost the same kind of entity in Fig. 1(a). These cases are very common in a heterogenous data collection.

**Challenge 3.** Many times, users need not only the related portions of the data but also the elements around those regions. The system must include enough extra data elements in the candidate answers.

For instance, if the user submits the query `Burt` to the database fragment in Fig. 1 (a), the system must return not only the node number 14 but also nodes number 7, 3, 8, and 10. As we show in the next section, the current methods do not address this problem in general.

**Contribution.** My research has been focused on addressing the above open problems. We have worked on finding effective and efficient methods that work for data sets with any schema structures and even with incomplete schema information. We expect the following results:

- A rigorous theoretical framework that defines the degree of relatedness of query terms to the data subgraphs based on available structural information. In order to solve the above challenges, the framework must not make assumptions that are not valid in general. It must work for data sets with heterogeneous schemas and incomplete schema information. The framework must also define the degree of the completeness of an answer. That is, it determines the size and the elements of each candidate answer. The framework must be able to combine the structural ranking with other sources of ranking information such as information-retrieval-style ranking.

- Efficient algorithms to extract the required structural information to rank the candidate answers. As the number of sub-structures is very large, naive approaches are prohibitively inefficient. Thus, we have to use optimizations and approximation methods to design algorithms that extract the structural information efficiently.

- Efficient indexing structures and algorithms that use the structural information extracted in the previous step to find and rank the candidate answers in query time. We also need algorithms that identify and return enough data elements for each candidate answer.

- A prototype which can be used to evaluate and prove the validity of our framework on real-world data sets.

## 2. RELATED WORK

The consensus in keyword search for relational, XML, and graph DBs is that the best answers are the most specific entities or data items containing the query terms [3, 8, 10, 15, 18, 19, 20, 24, 25, 28, 31, 32, 34, 35, 41, 49]. The specificity of a subtree or subgraph of data elements depends on the strength of the relationship between its nodes. For instance, if two nodes merely belong to the same bibliography, such as titles of two different papers, then the user will not gain any insight from seeing them together in an answer. If the nodes belong to the same paper, such as the paper’s title...
and author, the user will surely benefit from seeing them together. If the nodes represent titles of two different papers cited by one paper, the answer might be slightly helpful.

The baseline method for XML keyword search returns every candidate answer [41], with modest refinements in [18, 19, 50]). For instance, consider the DBLP fragment from http://dblp.uni-trier.de/xml shown in Fig. 1(a). The answer to query Integration Miller is (rooted at) node 2. This approach has high recall but very low precision. In $Q_3 = \text{Integration VLDB}$ for Fig. 1(a), candidate answers node 9 and 1 are unhelpful. The node 9 tree contains two otherwise-unrelated papers cited by the same paper, and the node 1 tree contains otherwise-unrelated papers in the same bibliography. A good query algorithm should either not return these answers, or rank them below the helpful answers.

One approach eliminates every candidate answer whose root is an ancestor of the root of another candidate answer [9, 34, 42, 49]; the LCAs of the remaining candidate answers are called the smallest LCAs (SLCAs). The SLCA approach relies on the intuitively appealing heuristic that far-apart nodes are not as tightly related as nodes that are closer together. For $Q_1$ in Fig. 1(a), the SLCA approach does not return node 1, however, it does still return node 9; as it does not rank its answers, the user will get a mix of unhelpful and desirable answers. SLCA’s recall is less than the baseline’s: for query $Q_3 = \text{XML Burt}$, nodes 3 and 15 are desirable; but since node 3 is an ancestor of node 15, node 3 will not be returned. Moreover, this heuristic cannot boost the effectiveness of the keyword search over text-centric data sets as they have a relatively flat structure [16].

[14, 15, 28] remove every candidate answer having two non-leaf nodes with the same label. The idea is that non-leaf nodes are instances of the same entity type if they have duplicate labels (DLs), and there is no interesting relationship between entities of the same type. We refer to this heuristic as DL. For instance, the subtree rooted at node 9 does not represent a meaningful relationship between nodes 19 and 20, because they have the same label and type. Therefore, node 9 should not be an answer to $Q_2$. DL is not an ideal way to detect nodes of similar type. For example, nodes article and paper in Fig. 1(b) have different names but represent similar objects. As a result, for the queries $Q_4 = \text{SIGMOD XPath}$, DL returns node 11, which is undesirable. This situation is prevalent in heterogeneous data collections. DL cannot detect uninteresting relationships between nodes of different types, either; it does not filter out node 1 for query $Q_3 = \text{UBC Green}$ in Fig. 1(b). Further, sometimes there are meaningful relationships between similar nodes, even in a DB with few entity types. For example, DL does not return any answer for Smith Burt in Fig. 1(b), as it filters out node 3. [15] ranks the answers by the number of nodes they contain and their TF/IDF, but this scheme does not help to avoid DL’s pitfalls. In [46, 44], we have shown that the same types of pitfalls exist in other approaches that rely on the element label names, such as [28, 30, 31]. These approaches cannot be used when the schema information is incomplete, i.e., the label names of a data element are missing.

XReal [9] uses some ad hoc heuristics to rank XML keyword search results. It gives higher rank to the entities that have more instances of a keyword query term. However, this is not true in general. For instance, in the DBLP database whose fragments are shown in Fig. 1(b), the booktitle attribute is duplicated for all papers published in the same venue. Therefore, this attribute has a higher chance of having more instances of a keyword query term such as XML compared to an important attribute such as title. Also, it does not consider the relationship between nodes. As an example consider the query ER diagram submitted to the DBLP database. Since there is a conference named ER, the frequency of the word ER is higher in the booktitle attribute than its frequency in the attribute title. Therefore, XReal ranks the papers published in the ER conferences and have the term diagram in their title higher than the papers that have these terms together in their titles. Finally, there are nodes that have the fewer number of query terms but are more relevant to the user query. For instance, users who are submitting the query Data Mining Han to the DBLP database are mostly looking for the text book written by Han about Data Mining rather than his papers on data mining. Nevertheless, the title attribute of the paper entity has more instances of the terms Data Mining than the title attribute of the book entity.

The shortcomings of all these methods are that they filter out answers instead of ranking them and/or they rely on very shallow structural properties to rank answers. Since these methods rely on ad hoc heuristics, they are ineffective for many queries.

A similar line of research has been followed for keyword queries in relational and graph DBs [3, 8, 10, 17, 22, 25, 27, 29, 32, 35, 43]. These approaches create a join tree whose tuples contain the query terms, and rank the tree based on its size. The ranking can also consider DBA-specified weights on the edges in the join tree [8, 25]; however, we need an automated approach. There are other works on ranking XML query results considering users have enough knowledge of of the schema of the database and the query languages [4, 6, 38].

There has been less research on the completeness of each candidate answer [9, 25, 26, 33, 39]. [33] relies on the heuristic that each repeating data element represents an entity, and the closest entity to the related subtree is the desirable answer. However, this entity is not usually the desired information. For instance, it returns only node 14 in Fig. 1(a) for the query Burt. Whereas, the user usually would like to know not only Burt’s affiliation but also his publications. [9] chooses the appropriate ancestor based on a parameter set by the database administrator. It then returns all descendants of this node. This approach leaves users with overwhelming number nodes; therefore, users have to filter the desired nodes manually. [25, 26] require domain experts to identify the desired entities. [39] searches through the query log and finds the attributes whose instances co-occur frequently. It groups those attributes as the pieces of information users like to see together. Nevertheless, reliable query logs are not always available. As the length of the keyword queries is very short, this approach [39] cannot find all desired attributes. Moreover, many keywords in the query log belong to more than one attribute. The works on text-centric data sources rely on information-retrieval-style heuristics [47].

3. OUR SOLUTION

3.1 Theoretical Framework

Current approaches implicitly assume that all relationships between nodes in a candidate answer have the same
strength. For example, they assume that the relationship between book/title and author is as important as the relationship between title and author in DBLP. However, a paper’s author is more closely related to its title than to its proceedings title, in which case users will prefer answers whose title and author children match the query terms, rather than answers whose book/title and author children match. Also, the relationship between the title of a paper and the title of the papers it cites is stronger than the relationship between the titles of the cited papers themselves. Similarly, the relationship between the title of an article and the title of a paper is weak if their only connection is that they are in the same database. These observations suggest that the higher the dependency between schema elements, the more meaningful their relationship is. The concept of data dependency have been used in relational data model to group highly related attributes in the same table [36, 7]. We have formalized this intuition as coherency ranking, which uses information theoretic concepts to capture the correlation between the schema elements in a subtree [46, 44]. Coherency ranking ranks a subtree higher if its normalized total correlation (NTC) is higher than that of other candidate answers. NTC is normalized according to the number of the schema elements in the candidate answer and their statistical properties. Our implementation of coherency ranking allows the user to set a minimum value for the NTC of the subtrees it returns. Using this minimum value, the user can set the precision/recall tradeoff. Other proposed heuristics could be analyzed within the framework.

The final ranking could be combined with other sources of ranking information such as content-based information-retrieval-style ranking [37] and PageRank [11]. We have performed an empirical study on real-world data sets and showed that coherency ranking has higher precision and recall and better ranking quality than previous approaches [46, 44]. We have also combined information-retrieval-style methods and coherency ranking. Since similar attributes tend to have similar statistical properties and similar statistical relationships with other attributes, the coherency information could also be used to perform data integration in heterogeneous data sets. This, in turn, boosts the effectiveness of the keyword search over such collections.

Since there is little data redundancy in text-centric semi-structured data collections, the concept of data dependency cannot be applied in these collections explicitly. For instance, while the abstracts of two papers could be very close to each other, they do not match exactly. We are working on how to extend our framework to define the coherency for these databases. We also plan to extend the framework so that it can pick the right granularity for candidate answers.

### 3.2 Structural Information Discovery

Our system needs the NTC of a candidate answer to rank it at query time. We have designed and implemented a preprocessing step that extracts the meaningful substructures from an XML DB and computes their NTC’s, before the query interface is deployed [46, 44]. Since naive methods are prohibitively inefficient for the preprocessing step, we presented and evaluated optimization and approximation techniques to reduce preprocessing costs. Our experiments showed that these optimizations improve preprocessing performance by orders of magnitude while introducing negligible approximation errors [46, 44]. Preprocessing needs to be repeated after structural changes in the DB that introduce new node types, so that subtrees containing those types of nodes can be correctly ranked. However, the results of the preprocessing phase are not affected by non-structural updates to the content of a populated DB, so the preprocessing phase rarely or never needs to be repeated as the DB content evolves. Moreover, as our framework discovers the structural information, it can be used for searching over heterogeneous data sets and data sets with incomplete schema information. Our preprocessing algorithm could also be used for purposes other than keyword search, such as finding highly correlated elements and patterns [23], or discovering approximate functional dependencies [21] in XML databases.

We are working to extend our discovery algorithms to work with graph DBs (e.g., RDF, OWL, XML with ID/IDREF). Also the current algorithms are designed for text-centric data sets. We plan to extend them for text-centric data sets.

### 3.3 Query Time Search

We have proposed an extension of the previous algorithms to efficiently find the data elements that match the input query terms and build and rank the sub-structures that are related to the query [46, 44]. Since we compute the rank of the sub-structures in the preprocessing step, we can use this information to refrain from building low rank sub-structures.

Our experimental results showed that a relatively large fraction of query processing time for very large data sets is spent on index lookup to find all the matching data elements in the data sets. We are looking for techniques to improve the lookup time. We are also working to speed up the efficiency of the sub-structure building algorithm.

### 3.4 Prototype

We have developed and presented a prototype that implements our proposed framework and designed preprocessing as well as query time search algorithms for XML data sources [45]. We have used the prototype to conduct our evaluations and user studies and with two real-world XML data sets. We are working on a new prototype that supports graph databases.

### 4. CONCLUSION

We have identified the challenges facing keyword and natural query processing for semi-structured data sources. The current approaches do not offer a general and effective framework to address these challenges. We have proposed coherency ranking, a new ranking method for keyword search over semi-structured data sources that ranks candidate answers based on statistical measures of their cohesiveness and avoids overreliance on shallow structural details. We plan to extend coherency ranking to address keyword search over text-centric data sets, find the best set of data elements per candidate answer, and experiment its use in schema integration. Coherency ranks are computed based on a one-time preprocessing phase that exploits the structure of the data set. As it is too expensive to extract the required structural information, we developed approximation and optimization methods to make preprocessing affordable. We plan to extend our preprocessing algorithms to work with graph-structured DBs. For query answering, we designed an algorithm to rank candidate answers based on the precomputed ranks. In our user studies using our prototype and
two real-world data sets, coherency ranking showed considerable improvements in precision, mean average precision and recall, compared to the previously proposed methods. We plan to decrease the index lookup time and the time of building substrictures at query time.

5. REFERENCES


