ABSTRACT
A system for keyword search on data graphs is demonstrated on two challenging datasets: the large DBLP and Mondial (which is highly cyclic and has a complex schema). The system supports search, exploration and question answering. The demonstration shows how the system copes with the main challenges in keywords search on data graphs. In particular, the system generates answers efficiently and completely (i.e., it does not miss answers). It has an effective ranking mechanism that also takes into account redundancies among answers. Finally, the system uses a novel technique for displaying multi-node subtrees in a compact graphical form that facilitates quick and easy understanding, which is essential for effective browsing of the answers.

Categories and Subject Descriptors
H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval—Search process; H.2.4 [Database Management]: Systems—Query processing

General Terms
Algorithms, Human Factors

Keywords
Keyword search on graphs, information retrieval on graphs, redundancy elimination

1. INTRODUCTION
Keyword search over databases is indispensable when we want to quickly pose a focused query (i.e., one that involves some specific entities) without first studying the schema, or when the database has chunks of text. If properly designed and implemented, it could also be an important tool for data exploration, namely, a mechanism for finding the semantically different ways in which the keywords are interconnected. Thirdly, we would like to use it for question answering (e.g., what is the capital of France?). Our demonstration shows how to cope with these three tasks, namely, keyword search, data exploration, and question answering. In particular, we have developed methods for dealing with the main challenges in building an effective, user friendly system. It is not merely an efficient implementation of an algorithm for enumerating answers. The demonstration also employs a novel approach for displaying answers so that they are easily understood. In addition, it illustrates how to deal with redundant answers.

The core of any search engine is the algorithm for enumerating answers. When the underlying domain is a database, there are two approaches. In one (e.g., [3, 7, 8]), candidate expressions are extracted from the schema and evaluated. In the second approach (e.g., [1, 6]), the algorithm operates directly on a graph representation of the data. The advantage of the first approach is the ability to use the query processor of the database. The disadvantages are that many expressions could have empty results (causing inefficiency), and the algorithm can rank only according to a function on the expressions, but not on individual answers. The second approach may potentially overcome these disadvantages.

In [2], we have developed an enumeration algorithm that follows the second approach and is based on the techniques of [4]. This algorithm can handle any database, and in addition, it provably has the following three important properties. First, it is efficient, namely, answers are enumerated with polynomial delay. Second, it generates answers according to an initial ranking (which is a 2-approximate order by increasing height). Third, it is complete, that is, it can generate all the answers.\footnote{The initial implementation was based on the algorithm of [2]. In later stages, the system has evolved by incorporating optimizations as well as new techniques in order to improve the actual running time.}

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

\*Work supported by The German-Israeli Foundation for Scientific Research & Development (Grant 973–150.6/2007).
lenging datasets. One is the large DBLP\(^3\) (i.e., all the publications with all their data items), which we further enhanced with (a small number of) abstracts of some articles. The second is Mondial,\(^4\) which is highly cyclic and has a complex schema.

Quite often different answers to the same query have a common part, whereas the user may be interested in seeing (at least initially) results that are as different from one another as possible. We call this phenomenon the redundancy problem. Our system implements the techniques of [2] for handling this problem, and also allows the user to mark manually some of the displayed answers, thereby excluding (from subsequent pages) similar results. These features are important for exploration.

Our system is highly configurable. Many parameters (such as the ones that tune the handling of the redundancy problem) can be set by the user. Controlling these parameters enables the user to see the effect on the search results and facilitates exploration, rather than just search of the data.

Lastly, we have developed a novel approach to displaying answers so that they are easily and quickly understood. We discuss it in length in Section 3.4.

The ExQueX system of [5] is intended for assisting users in the task of posing queries over XML documents. Its goal is to facilitate quick formulation of queries without any advance knowledge of the schema and without being hindered by the complex syntax of a query language (such as XQuery). Thus, ExQueX is suitable for answering queries; for example, “list all the countries and their capital cities.” The system described in this paper is intended for keyword search; for example, it can find how the keywords “France” and “Paris” are related. Although the two systems have some similarities, they are mostly based on different techniques and serve complementary rather than overlapping purposes.

2. PRELIMINARIES

In the formalism of [2], data is represented as a graph. A node of the graph is either structural or a keyword. In particular, for each word appearing in the underlying data, there is exactly one corresponding keyword node. Data graphs are directed. An edge from a structural node to another represents a semantic relationship (e.g., between an author and her article). If a keyword \(w\) appears in the data associated with a structural node \(s\), then there is an edge from \(s\) to \(w\). Keyword nodes do not have outgoing edges.

Consider a data graph \(G\) and a subset \(V\) of the nodes of \(G\). A subtree \(T\) of \(G\) is reduced with respect to \(V\) if \(T\) includes the nodes of \(V\) and there is no proper subtree of \(T\) that also has all of those nodes.

A query \(Q\) is a set of keywords; equivalently, \(Q\) can be viewed as a set of keyword nodes. In principle, a reduced subtree of \(G\) w.r.t. \(Q\) is an answer to \(Q\).

Given a data graph \(G\) and a query \(Q\), the algorithm of [2] enumerates all the answers (i.e., reduced subtrees of \(G\) w.r.t. \(Q\)) with polynomial delay in a 2-approximate order by increasing height.

Either a relational database or an XML document can be translated into a data graph. In this paper, we assume that the underlying data is in XML format. The data graph incorporates edges that correspond to ID references, and hence could be cyclic.

Translating an XML document to a data graph is not necessarily trivial. For example, suppose that two structural nodes represent a paper \(p\) and one of its authors \(a\). Should there be just one directed edge between \(a\) and \(p\), or two (one in each direction)? If we use the rule that there is an edge from a paper to each one of its authors (but not in the opposite direction), then it might be impossible to get answers that consist of an author and several of her papers (since such an answer cannot be represented as a directed reduced subtree). If, on the other hand, there are edges in both directions, then we might get distinct answers that are essentially the same, except for the direction of the edge between the author \(a\) and the paper \(p\). (One may suggest to view the data graph as undirected, but this also raises problems as discussed later.)

We have taken the approach that maximizes the potential answers. For that reason, the default rule is that the data graph represents an ID reference (of the original XML document) by directed edges in both directions. However, this approach necessitates an effective mechanism for dealing with redundant answers, as explained later.

3. OVERALL ARCHITECTURE

In this section, we discuss four important components of our system. We start with the storage of the data graph. Secondly, we describe how weights are assigned to the nodes and edges of the data graph. Thirdly, we give an overview of the search engine. Fourthly, we describe how answers are displayed to the user.

It is important to note that the algorithm for enumerating reduced subtrees is just the first part of the search engine. This algorithm ranks answers by increasing height (rather than weight), and it cannot deal with the redundancy problem. The search engine is a two-stage process. Firstly, it applies the enumeration algorithm. Secondly, it determines the final ranking of the answers according to their weight as well as their degree of redundancy relative to previous answers, as explained in Section 3.3. This two-stage process is an effective approach, because there is a good correlation between ranking by height and ranking by weight.

3.1 Storage

Our system combines a skeleton of the data graph \(G\) that resides in main memory and a DBMS that stores the rest of the information. Dividing the data between the skeleton and the DBMS reduces the amount of main memory consumed by the algorithm that enumerates reduced subtrees.

The skeleton of \(G\) consists of only the structural nodes and the edges between them. The DBMS stores (among other things) the keyword nodes and the edges that enter those nodes. When a user poses a query \(Q\), the system retrieves all the keywords \(w\) of \(Q\) and the edges that enter \(w\), and adds those nodes and edges to the skeleton. When a reduced subtree \(T\) is found, the DBMS is employed again in order to augment \(T\) with additional information (as explained later) so that the presentation of \(T\) to the user would be meaningful.

3.2 Weights

The weights that are assigned to the nodes and the edges of \(G\) determine the ranking of the results produced by the

\(^3\)http://dblp.uni-trier.de/xml/
\(^4\)http://www.dbis.informatik.uni-goettingen.de/Mondial/
search engine. The skeleton represents database entities and relationships between them. Therefore, in the skeleton, the weights of the nodes and the edges should reflect the semantic strength of those entities and relationships. In particular, these weights are typically independent of the keywords, and therefore it is sufficient to compute them just once. There are various ways of determining these weights (e.g., see [2]).

When attaching a keyword \( w \) to the skeleton, as described earlier, we also add the information about the weights of \( w \) and its edges. For these weights, it is appropriate to use information-retrieval (IR) measures, such as tf/idf. For example, suppose that a structural node \( s \) represents the abstract of an article, and the keyword \( w \) appears in that abstract. The weight of the edge \( e \) from \( s \) to \( w \) could be chosen based on the score\(^5\) of the tf/idf formula w.r.t. the keyword \( w \) and the text associated with \( s \).

Sometimes a user is interested only in answers in which some specified keywords appear together in the same fragment of text. If the user puts a group of keywords inside curly brackets (e.g., \{operating system\}), then a single node \( c \) is created for all of those keywords. An edge is introduced from a structural node \( s \) (e.g., abstract) to \( c \) only if the text associated with \( s \) includes all the keywords corresponding to \( c \).

Even when curly brackets are absent from the query, users typically prefer answers in which several keywords appear in the same fragment. If, after attaching the keywords of the query to the skeleton, a structural node \( s \) has \( n \) edges to keyword nodes, then the weight of each of those edges is divided by \( n \). This improves the ranking of answers with several keywords in the same fragment.

3.3 The Search Engine

The search engine is responsible for enumerating answers, ranking them, and handling the redundancy problem. We first discuss different types of redundancy.

The most basic form of redundancy is duplication. The enumeration algorithm generates each reduced subtree exactly once, but two distinct subtrees could be deemed identical if, for example, the only difference between them is the direction of the edge connecting the same pair of nodes. We can eliminate such a redundancy if between any pair of connected nodes, there is an edge in only one direction. However, as explained earlier, this may exclude answers of certain types. Alternatively, we may view the data graph as undirected, but our demonstration shows that in certain situations, this creates some answers that are semantically meaningless. It suffices to say that the data graph should be created judiciously in order not to exclude meaningful answers while avoiding (as much as possible) duplicate or utterly meaningless answers. There is also a need for a suitable definition of duplicate answers. A thorough discussion of these issues is beyond the scope of this paper. However, the demonstration gives some specific examples that illustrate how to deal with these issues.

Duplication is just one facet of redundancy. As enunciated in [2], keyword search on graphs sometimes inundate the user with many answers that are too similar (but not identical) to one another. One type of similarity is when two answers represent different instances of the same semantic connection between the keywords. For example, the top answers to the query \textit{milo abiteboul} are papers co-authored by the two. After seeing a few of those papers, a user might be interested in answers that represent different semantic connections.

Another measure of similarity is based on analyzing, in each answer, the connection between every pair of keywords. Let \( w_1 \) and \( w_2 \) be two keywords of the query. If two answers have identical paths of nodes between \( w_1 \) and \( w_2 \), then they represent exactly the same semantic connection between the two keywords. (Note that other pairs of keywords could be connected in semantically different ways in the two answers.)

If the two answers have the same sequence of labels (but not the same nodes) between \( w_1 \) and \( w_2 \), then the two connections have similar semantic “flavors.”

We now describe how the search engine ranks answers. In order to present the next \( n \) answers to the user, the system generates the next \( kn \) reduced subtrees by applying the enumeration algorithm mentioned earlier. Then, the system finds the top-\( n \) answers among those that have already been generated, but not yet shown to the user. These \( n \) answers are the next ones in the final ranking order that is presented to the user. This is the approach employed in [1]. However, as discussed in [2], we also take into account similarity to previous answers when determining the final ranking. In particular, the rank of an answer is the sum of two factors. The first is the weight of the reduced subtree. The second is a penalty that is calculated as follows. Let \( A \) be the set of the answers that have already been shown to the user, and let \( S \) be the rest of the reduced subtrees that have been generated thus far. For each subtree \( T \) of \( S \), we compare the degree of similarity between \( T \) and the answers of \( A \). Based on this comparison, we assign a penalty to \( T \). In particular, for each pair of keywords of the query, we check whether the connection between them in \( T \) is identical (i.e., same path) or similar (i.e., same sequence of labels) to that in some subtree of \( A \). Clearly, an identical connection incurs a higher penalty than a similar one. The penalty of \( T \) is cumulative over all pairs of keywords, and is added to the weight of \( T \). (See [2] for more details.) The next answer in the final ranking order is the subtree of \( S \) that has the smallest sum of the weight and the penalty. Note that the penalty has to be recomputed after each answer that is added to the final ranking order. The system displays to the user the next \( n \) answers in the final ranking order and then repeats the whole process (i.e., it enumerates the next \( kn \) reduced subtrees by increasing height and so on). It should be noted that when computing the penalty of a subtree \( T \) of \( S \), we also check whether \( T \) duplicates an answer that has already been presented to the user. If so, \( T \) is discarded.

3.4 Displaying Answers

Our first attempt at displaying a reduced subtree was to translate it into XML, and then to render it by means of an XSL stylesheet. The result, however, was hard to understand, especially when the subtree had quite a few nodes. In fact, it took a considerable amount of time just to grasp the rough essence of a single answer, and the whole process of browsing a few pages of answers took so long that it was ineffective. Another problem we encountered was that reduced subtrees often lack essential information (e.g., a country appears with its id, but without its name).

So, we developed a whole new approach. First, similarly to [6], we classify each node of a reduced subtree as either...
an object, a connector, or a property. This is done by applying rules that implement the chain-of-responsibility design pattern. These rules are derived from the underlying XML document and its DTD. In comparison, the goal of [6] is different, namely, a heuristic for understanding the meaning of a user query. Moreover, their rules are too simplistic for our purpose, because they assume a tree structure with a simple schema. We had to design a new framework for combining a large set of rules and, in particular, resolving conflicts therein.

Not only does the classification imply how a reduced subtree should be displayed, it also indicates which information from the original XML is missing (e.g., if an object is in the answer, then all its properties should be included as well). Using this classification, we translate a subtree into a new XML fragment, and display it using an XSL stylessheet and JavaScript code that we developed.

Figure 1 shows the rendering of an answer (where the query is **Andalusia Spain Sevilla**) vs. the corresponding fragment of the underlying XML. To avoid clutter, the number of nodes is small. To enhance clarity, different shapes and colors are used. Objects are gray rectangles. The basic information about an object is its title (e.g., Spain) and its label (e.g., country). The former is shown in blue and the latter in red inside parentheses. Additional information consists of properties (e.g., population) and their values. By default, a property is displayed only if its value contains occurrences of some of the query keywords. By clicking the link “Expand to view full result,” all the properties and their values are shown (in the rectangle).

A connector is a red line between two objects. If the meaning of the connector is not clear from the labels of the two objects, then the red line goes through a circle that has a label. Thus, the answer of Figure 1 states that the province **Andalusia** is in **Spain** and **Sevilla** is its capital. Note that occurrences of the query keywords are underlined in green.

In comparison, existing systems (e.g., [1,3,7]) display the answers either in a format that is quite similar to the raw XML or in a graphical interface that clutters the display with too many nodes. In either case, it is much harder to understand the meaning of an answer than in our approach.

### 4. THE DEMO SCENARIO

Those attending the demonstration can pose queries on DBLP and Mondial. The system has an extensive configuration file and users can change parameters and see the effect on the generated answers. For example, ranking can be done without assigning penalties.

When posing the query **tova milo**, the user will unexpectedly get articles by the author Milos, due to stemming. Enclosing the keywords in curly brackets (i.e., (**tova milo**)) will generate answers where “Tova” and “Milo” appear together, thereby getting the desired articles.

For the query **query languages**, the top answers are articles that have these keywords in their properties (e.g., title); that is, each of these answers consists of a single object. If the user is looking for answers that consist of more than one object (e.g., two articles written by the same author, such that one has the keyword **query** and the other has an occurrence of **languages**), then she should check the box “exclude similar answers.” As a result, subsequent answers will have different schemas from those already shown to the user; that is, the new answers will not be isomorphic to any of the previous ones.

The system also supports question answering by allowing to use labels as keywords. For example, for the query **tova milo author**, the top answers are authors that wrote papers with Tova Milo. Similarly, the query **CoXML author** returns as top answers authors of articles on CoXML.

Mondial is a challenge to any search system, because it typically returns many answers with a large variety of semantic flavors. The demonstration shows that question answering works also on Mondial. The query **russia capital** returns Moscow as the top answer. If we want to find the capitals of all provinces in Russia, we run the query **capital province russia**. However, for each province of Russia, we will get two answers. In one, the city is the capital of that province, whereas in the second, the city is the capital of Russia. If the user checks the box “exclude similar answers” next to one of the unwanted answers (i.e., where Moscow is shown as the capital of Russia), then subsequently she will get only capitals of provinces.

The full capabilities of the system to explore, rather than just search, become evident when posing the query **france germany**, which returns many answers with a multitude of different semantic flavors. For example, there are answers that provide information about organizations in which both countries are members. Other answers are about a third country that has borders with both France and Germany, and there are many more answers.

### 5. REFERENCES


