User Interaction in the BANKS System: A Demonstration

B. Aditya[1] **Soumen Chakrabarti** **Rushi Desai** **Arvind Hulgeri** **Hrishikesh Karambelkar**
Rupesh Nasre **Parag**[1] **S. Sudarshan**

Indian Institute of Technology, Bombay
Aditya_Bhashyam@mckinsey.com, {soumen,rushi,aru,hrishi,rupesh,sudarsha}@cse.iitb.ac.in, parag@cs.washington.edu

Abstract

The BANKS system supports keyword search on databases storing structured/semi-structured data. Answers to keyword queries are ranked, and as in IR systems, the top answers may not be exactly what a user is looking for. Further interaction with the system is required to narrow in on desired answers. We describe some of the new features that we have added to the BANKS system to improve user interaction. These include an extended query model, richer support for user feedback and better display of answers.

1. Introduction

Traditionally, searching text documents and searching structured databases have been supported by very different systems. The Information Retrieval (IR) community has developed systems for searching document collections using free-format keyword queries which do not require any notion of a schema, whereas the Relational Database (RDBMS) community has developed a precise, schema-cognizant query language (SQL) together with query-processing technology.

We see a great need for systems that will embody a convergence of these technologies. The Web, XML, and database application servers have blurred the distinction between scenarios where one or the other technology is suitable. Increasingly, relational, object-relational, and XML databases are being searched over a Web interface (typically, using HTML forms).

Users of such interfaces do not wish to know, or cannot make use of, detailed schema information, and therefore they cannot use precise query languages like SQL or XQuery to interact with such databases. On the other hand, although IR systems support keyword querying along with useful features such as ranking of answers, they deal with documents that are well-defined units of text, whereas information needed to answer a query on a database may be split across multiple items (tuples or XML elements).

To address the gap between the two extreme search paradigms mentioned above, we are building a system we call BANKS (an acronym for Browsing ANd Keyword Searching) [1, 2], to support keyword search on databases storing structured/semi-structured data. A BANKS query, in its simplest form, is just a set of keywords, as in an IR system. Unlike in an IR system, the answer to a keyword query may be a set of data items, each of which matches a keyword, along with the connections between the data items. Examples of connections include foreign key references in relational data sources, and element containment in XML.

As in IR systems, there is no guarantee that the top answers will satisfy the user's needs, but further interaction with the BANKS system can help users narrow in on the information that they require.

In this paper we describe some of the new features that we have recently added to the BANKS system to improve user interaction. These include an extended query model, richer support for user feedback and better display of answers. The demo will emphasize these features.

A demo of the BANKS system is accessible over the Web at `http://www.cse.iitb.ac.in/banks/`. The demo will use several sample databases, one containing bibliographic information from DBLP, another containing student information related to thesis submissions at IIT Bombay, and a third database derived from IMDB (the Internet Movie Database).

2. Basic Data and Query Model

BANKS uses a uniform directed graph model, which naturally captures relational data, XML data and HTML data with hyperlinks. The graph nodes correspond to tuples or XML elements. Graph edges correspond to connections between the nodes; for example, foreign-key references in a relational database, and element containment or IDREFs in an XML database. Graph edges and nodes are assigned weights as outlined in Section 3.3; conceptually, edge weights correspond inversely with the degree of closeness, while node weights are a measure of "prestige".

A keyword query seeks to find subgraphs whose nodes *collectively* satisfy the query. An example of a keyword query is "`web mendelzon`". A keyword may match text in a relational attribute (or XML element), or even *metadata* text, such as the name of a table, column, or XML element or attribute tag. Given a set of keywords, the system returns a set of *answer trees*; each answer tree is such that each

[1]Work performed while at I.I.T. Bombay.

user-specified keyword matches (at least) one of the nodes of the tree. Answer trees are ranked based on the weights of their edges and nodes. See [2] for details.

3. Extended Query Model

We describe several recently implemented extensions of the BANKS query model in this section.

3.1. Node Selections

BANKS allows selection conditions to be specified on nodes, along with keywords. For instance,

`optimization (year = 2000)`

selects nodes that match the keyword optimization, and have an attribute called *year* whose value is 2000.[2] Selections on ordered domains can be approximate; for example, the query

`optimization (year ∼ 2000)`

looks for papers on optimization, published in a year around 2000. The weight of a node matching the keyword "optimization" then becomes a function of how closely its *year* attribute matches 2000. The attribute name can be dropped, as in "`optimization (∼ 2000)`" in which case matching is done on all available attributes.

3.2. Proximity

BANKS allows the ranking function (**near** K), where K is a set of keywords, to be specified along with a keyword. For example, consider the query

`movies (`**near** `hitchcock, reagan)`

Answers to the query are nodes that match the keyword "movies", and are in proximity to nodes that match the keywords "hitchcock" or "reagan". Intuitively, the presence of multiple nodes that match the near set ("hitchcock" or "reagan") near a particular movie node increases the weight (relevance score) of that movie. As another example, the query "`author (`**near** `recovery)`" would intuitively give us authors who have published many papers on recovery.

When the **near** ranking function is specified along with a keyword, its effect is to modify the weights of nodes matching the keyword. The rest of the BANKS node and edge weight model remains unchanged.

The **near** ranking function aggregates the proximity to multiple nodes. For example, as in [4], the proximity of a node m with a near set Y can be defined as $\prod(1 - \frac{1}{d_i})$ where d_i is the distance from m to the ith node matching a keyword in the near set Y; the weight of m is then inversely related to the above proximity. Goldman et al. [4] introduced this model of search based on proximity; however, unlike BANKS, they do not consider general keyword queries. The **near** ranking function allows the Goldman proximity model to be cleanly integrated into BANKS.

[2]This can be extended to allow attribute names that are similar, or are in closely connected tuples, although this extension is not currently implemented.

3.3. Tree Weight Model

In the current BANKS model, the weight of a tree is computed as a combination (e.g. the sum) of the weights of its edges and nodes. The weight of a forward edge is based on its edge type, and can be specified by the system administrator (defaulting to 1) The weight of a backward edge from a node (i.e. a backward traversal of an edge pointing into the node) is proportional to the number of edges pointing to the node. Nodes weights are a measure of "prestige" can be based on the indegree of a node, or using the PageRank algorithm [3]. We are currently experimenting with extensions of the random walk model of PageRank, by biasing the walk, using attributes of tuples associated with nodes, or external statistical data.

Although the current ranking model has worked well on the data sets we have studied so far, a potential problem is that a small variation in the schema can significantly change the weight of paths connecting nodes of interest. In ongoing work we are exploring an alternative approach to "charging" an answer tree for (lack of) proximity between nodes of interest. Our approach is based on a random walk model, but restricted to the neighbourhood of the answer tree.

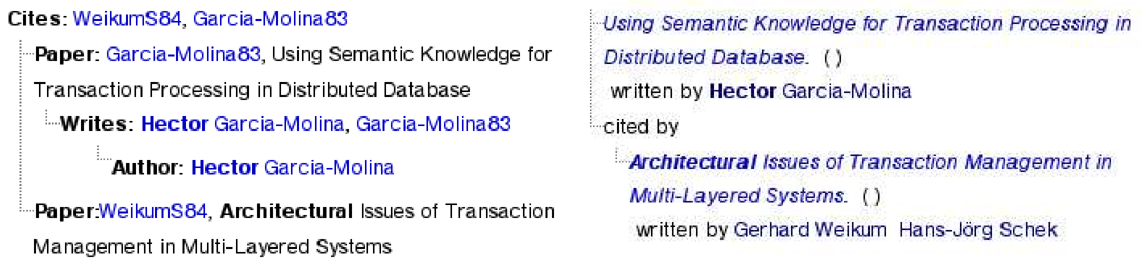
4. User Feedback

Keyword queries are inherently ambiguous, so a user may need to interact with the system to find required answers. BANKS provides several strategies for refining queries to get required results.

Disambiguation of Nodes: A given keyword, such as "sudarshan" may match several nodes, for example "S. Sudarshan" and "Sudarshan Chawathe". BANKS allows users to select which nodes are relevant and re-execute the query with those nodes.

Answer Patterns: Suppose a user wishes to find papers by Soumen that refer to papers by Sudarshan, and executes a query `sudarshan soumen`. This query would return papers written jointly by Sudarshan and Soumen, papers by Sudarshan that refer to papers by Soumen in addition to papers by Soumen that refer to papers by Sudarshan. The BANKS system system allows the users to select particular tree patterns as relevant and find only answers that match that pattern. The tree patterns are used to prune the search for answers.

Re-scoring: Node disambiguation and answer tree patterns result in a strict selection/rejection of answers. We are also adding a feature for the user to express a "softer" preference, by simply marking some answers as relevant (or more relevant than unmarked answers). The random walk model for answer tree scoring, described earlier, can use this information to prefer or avoid certain paths.



Cites: WeikumS84, Garcia-Molina83

Paper: Garcia-Molina83, Using Semantic Knowledge for Transaction Processing in Distributed Database

Writes: **Hector** Garcia-Molina, Garcia-Molina83

Author: **Hector** Garcia-Molina

Paper:WeikumS84, **Architectural** Issues of Transaction Management in Multi-Layered Systems

Using Semantic Knowledge for Transaction Processing in Distributed Database. ( )

written by **Hector** Garcia-Molina

cited by

***Architectural** Issues of Transaction Management in Multi-Layered Systems.* ( )

written by Gerhard Weikum Hans-Jörg Schek

Figure 1. Answer formatting: (a) Connection Tree Display (b) Improved Display

5. Answer Formatting

Consider the keyword query "`hector architectural`" on the *DBLP* database. One of the result trees have been shown in Figure 1(a). Two shortcomings can be easily identified. First, the *paper-id* attribute which forms the primary key for *paper* is meaningless to users, and displaying the title and year of a paper without its authors (or with a partial list of authors) gives incomplete information to the user. Second, the *writes* and *cites* tuples are "relationship" tuples, which serve only to link other tuples. Since the linked tuples are adjacent in the tree, the content of the tuples are superfluous except to indicate the directionality of the relationship.

The first problem is solved by allowing information from related nodes to be "folded in" when displaying a node. For instance, the system administrator can specify that whenever a *paper* tuple is displayed, the system should also display the *author* tuples linked through the *writes* relationship. This can be seen in Figure 1(b), where the author name Schek has been folded in and displayed with the second paper.

The second problem is solved by allowing the system administrator to specify "forward" and "backward" names to relations that model binary relationships (i.e. relations having two foreign keys). The foreign key information is then suppressed, but the directionality of the relationship indicated by displaysing the appropriate (forward or backward) name. For instance, the *cites* tuple displayed in Figure 1(a) has been replaced by the name "cited by" in Figure 1(b) (if the parent and child nodes were interchanged, the name "cites" would have been used instead).

In some cases, the information in a leaf node of the tree may have already been folded into the display of a parent node (for example, an author name into a paper). In such cases, the display of the leaf node (along with any parent binary relationship node) is suppressed whenever doing so would not result in loss of information.

The BANKS system supports templates for formatting the display of tuples; templates can contain HTML code, along with hyperlinks to attributes, and relationships to be folded in.

6. Implementation and Efficiency Issues

The BANKS system is built using Java Servlets to provide a Web interface, and JDBC to communicate with relational databases; BANKS can be run on any database supporting JDBC, without any programming. Extensions to support XML are being implemented.

An important concern when using a system such as BANKS is the efficiency of graph traversals required to find answers. If a disk access were potentially required for each edge traversal, the system would be unusably slow. We therefore store the database graph in memory. Note that we do *not* require the database to fit in memory. The in-memory structure is a graph which basically acts as an index on the database; the graph stores a single node (an integer identifier along with some pointers) for each edge and each node, which in our current Java implementation takes around 30 bytes. No other structure need be in memory, and no strings are stored in the in-memory structure. As a result, we can easily handle databases containing millions to even tens of millions of records using a modest amount of memory, sufficient for most organizational data.

7. Conclusion

In this paper we have outlined novel user interaction features of BANKS. Future work includes improved user feedback, and querying across multiple data sources using different data models, using the unifying graph model.

References

[1] B. Aditya, G. Bhalotia, S. Chakrabarti, A. Hulgeri, C. Nakhe, Parag, and S. Sudarshan. Banks: Browsing and keyword searching in relational databases. In *Proc. of the Int'l Conf. on VLDB*, Aug. 2002. Demo paper.

[2] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using BANKS. In *IEEE Int'l Conf. on Data Engineering*, Feb. 2002.

[3] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7), 1998.

[4] R. Goldman, N. Shivakumar, S. Venkatasubramanian, and H. Garcia-Molina. Proximity search in databases. In *Proc. of the Int'l Conf. on VLDB*, pages 26–37, 1998.