A System for Keyword Proximity Search on XML Databases

Andrey Balmin
UC, San Diego
abalmin@cs.ucsd.edu

Vagelis Hristidis
UC, San Diego
vagelis@cs.ucsd.edu

Nick Koudas
AT&T Labs–Research
koudas@research.att.com

Yannis Papakonstantinou
UC, San Diego
yannis@cs.ucsd.edu

Divesh Srivastava
AT&T Labs–Research
divesh@research.att.com

Tianqiu Wang
UC, San Diego
tiwang@ucsd.edu

1 Overview

Keyword proximity search is a user-friendly information discovery technique that has been extensively studied for text documents. In extending this technique to structured databases, recent works [6, 7, 4, 2] provide *keyword proximity search* on labeled graphs. A keyword proximity search does not require the user to know the structure of the graph, the role of the objects containing the keywords, or the type of the connections between the objects. The user simply submits a list of keywords and the system returns the sub-graphs that connect the objects containing the keywords.

XML and its labeled graph/tree abstractions are becoming the data model of choice for representing semistructured, self-describing data, and keyword proximity search is well-suited to XML documents as well. We describe a system that provides keyword proximity search on XML data that are modeled as labeled graphs or trees, the edges correspond to the element-subelement relationship and to `ID/IDREF` links (in the case of graphs). Our work differs from prior systems for proximity search on labeled graphs in that it can take advantage of knowledge of the schema, e.g., the XML Schema [12], to which the XML data conforms. The schema facilitates the presentation of the results and is also used in optimizing the performance of the system.

The results of keyword proximity search in our system are the minimum connecting trees of XML fragments (called *target objects*) that contain all the keywords and are ranked according to their size. Trees of smaller sizes denote higher association between the keywords, which is generally true for reasonable schema designs. For example, consider the keyword query "Yannis, Vasilis" on the graph of Figure 1. The first highlighted tree (thick edges) $Person[Yannis] \leftarrow Author \leftarrow Paper \rightarrow Author \rightarrow Person[Vasilis]$ on the source XML graph of Figure 1 is a result of size 4. The second highlighted tree (gray arrows) $Person[Yannis] \leftarrow Author \leftarrow Paper \leftarrow Cites \rightarrow Paper \rightarrow Author \rightarrow Person[Vasilis]$ is a result of size 6. The first result is considered to be a "better" one by our system (as well as by other keyword proximity search systems [2, 4, 6, 7] that use structure to rank results) since the smaller size corresponds to the closer connection between "Yannis" and "Vasilis" in the first solution, where they are co-authors of the same paper, as opposed to being authors of different papers one of which cites the other. Notice that we allow edges to be followed in either direction.

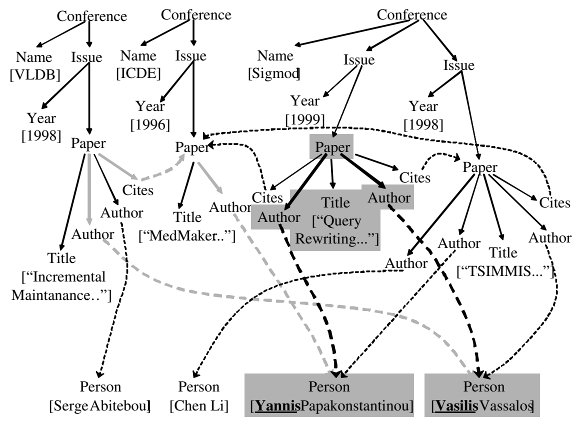


Figure 1: Sample XML document

The presentation of results faces two key challenges that have not been addressed by prior systems for proximity search on labeled graphs. First, the results need to be semantically meaningful to the user. Towards this direction, we group the nodes of the source XML graph into *target objects*, that are presented to the user. In the DBLP demo (Figure 4) we display target object fields such as the paper title and conference along with a paper. For example, we display the following *target object* :

$$Paper[title[Query\ Rewriting\ for\ Semistructured\ Data]]$$

Proceedings of the 29th VLDB Conference,
Berlin, Germany, 2003

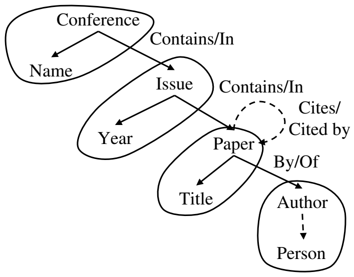


Figure 2: Target decomposition of a schema graph

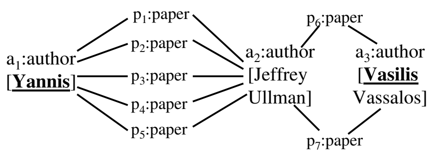


Figure 3: Multivalued dependencies in results

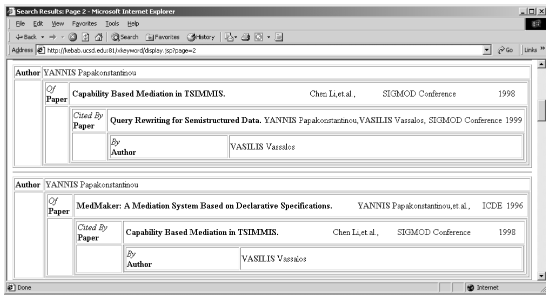
in the place of the intermediate $Paper$ node. The edges connecting the target objects in the presentation graph are annotated with their semantic description. For example the $Paper \rightarrow Author$ edge is named "*By*". Target objects are identified by the DBA who splits the schema graph in minimal self-contained information pieces (Figure 2), which we call *Target Schema Segments (TSS)* and correspond to the target objects presented to the user.

The second challenge is to avoid overwhelming the user with a huge number of often trivial results, as is the case with DISCOVER [7] and DBXplorer [2].[1] Both systems present *all* trees that connect the keywords. In doing so they produce a large number of trees that contain the same pieces of information many times. For example, consider the keyword query "Yannis, Vasilis" and the XML subgraph shown in Figure 3. Since "Yannis" and "Ullman" co-authored five papers, and "Vasilis" and "Ullman" co-authored two, this XML fragment contains 10 results, of which four are shown below:

$N_1 : a_1 \leftarrow p_1 \rightarrow a_2 \leftarrow p_6 \rightarrow a_3,$
$N_2 : a_1 \leftarrow p_2 \rightarrow a_2 \leftarrow p_6 \rightarrow a_3,$
$N_3 : a_1 \leftarrow p_1 \rightarrow a_2 \leftarrow p_7 \rightarrow a_3,$
$N_4 : a_1 \leftarrow p_2 \rightarrow a_2 \leftarrow p_7 \rightarrow a_3$

The above results contain a form of redundancy similar to multi-valued dependencies [11]: we can infer N_3 and N_4 from N_1 and N_2. In that sense, N_3 and N_4 are trivial, once N_1 and N_2 are given. Such trivial results penalize performance and overwhelm the user. Our execution algorithm and presentation interface avoids producing and presenting such "duplicate" results.

[1]Both systems work on relational databases, but the presentation challenges are similar.


Search Results: Page 2 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://kebab.ucsd.edu:81/xkeyword/display.jsp?page=2
Go Links
Done
Internet

Author	YANNIS Papakonstantinou		
	Of **Paper**	**Capability Based Mediation in TSIMMIS.** Chen Li,et.al., SIGMOD Conference 1998	
		Cited By **Paper**	**Query Rewriting for Semistructured Data.** YANNIS Papakonstantinou,VASILIS Vassalos, SIGMOD Conference 1999
			By **Author** VASILIS Vassalos

Author	YANNIS Papakonstantinou		
	Of **Paper**	**MedMaker: A Mediation System Based on Declarative Specifications.** YANNIS Papakonstantinou,et.al., ICDE 1996	
		Cited By **Paper**	**Capability Based Mediation in TSIMMIS.** Chen Li,et.al., SIGMOD Conference 1998
			By **Author** VASILIS Vassalos

Figure 4: On-line demo output

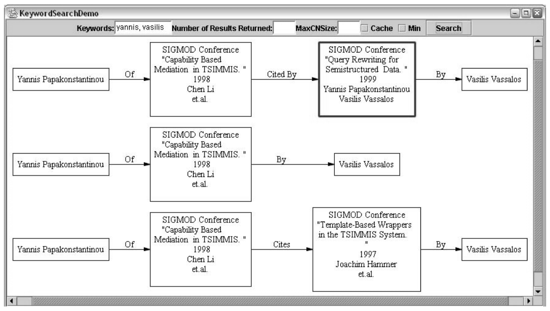

KeywordSearchDemo
Keywords: yannis, vasilis
Number of Results Returned:
MaxCNSize:
Cache
Min
Search
Yannis Papakonstantinou
Of
SIGMOD Conference "Capability Based Mediation in TSIMMIS." 1998 Chen Li et.al.
Cited By
SIGMOD Conference "Query Rewriting for Semistructured Data." 1999 Yannis Papakonstantinou Vasilis Vassalos
By
Vasilis Vassalos
Yannis Papakonstantinou
Of
SIGMOD Conference "Capability Based Mediation in TSIMMIS." 1998 Chen Li et.al.
By
Vasilis Vassalos
Yannis Papakonstantinou
Of
SIGMOD Conference "Capability Based Mediation in TSIMMIS." 1998 Chen Li et.al.
Cites
SIGMOD Conference "Template-Based Wrappers in the TSIMMIS System." 1997 Joachim Hammer et.al.
By
Vasilis Vassalos

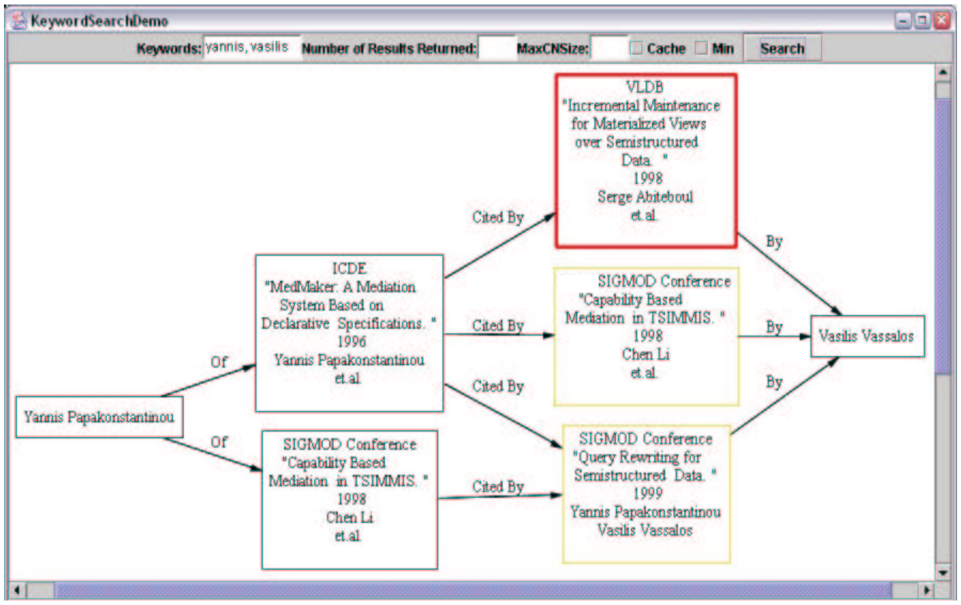
Figure 5: Initial presentation graph

We handle these challenges using *presentation graphs* [8], which encapsulate all the results of the same schema. A presentation graph is displayed for each result schema (see Figure 5). At any point, only a subset of the graph is shown (see Figure 6), as determined by various navigation actions of the user. Initially, the user sees one result tree r_0. By clicking on a node of interest the graph is expanded to display more nodes of the same type that belong to result trees that contain as many as possible of the other nodes of r_0. Towards this purpose we define a minimal expansion concept. For example, clicking on the highlighted $paper$ node of Figure 5 displays all $paper$ nodes connected to "Vasilis", along with a minimal number of nodes that they connect to, as shown in Figure 6. In order to provide fast response times, we employ indexing techniques that allow us to quickly navigate in the XML graph/tree and find connections between the nodes that contain the keywords.

2 Related Work

There are a number of proposals for less structured ways to query an XML database by incorporating keyword search [5, 1] or by relaxing the semantics of the query language [9, 3]. However none of these works incorporates proximity search. Florescu et al. [5] propose an extension to XML query languages that enables keyword search at the granularity of XML elements, which helps novice users formulate queries.

In [6] and [4], a database is viewed as a graph with objects/tuples as nodes and relationships as edges. Relationships are defined based on the properties of each application. For example, an edge may denote a primary to foreign



KeywordSearchDemo

Keywords: yannis, vasilis Number of Results Returned: MaxCNSize: Cache Min Search

Figure 6: Expanded Presentation graph

key relationship. In [6], the user query specifies two sets of objects, the $Find$ and the $Near$ objects. These objects may be generated from two corresponding sets of keywords. The system ranks the objects in $Find$ according to their distance from the objects in $Near$. An algorithm is presented that efficiently calculates these distances by building hub indices. In [4], answers to keyword queries are provided by searching for Steiner trees [10] that contain all keywords. Heuristics are used to approximate the Steiner tree problem. Two drawbacks of these approaches are that (a) they work on the graph of the data, which is huge, and (b) the information provided by the database schema if available is ignored. In contrast, we use indexing techniques that allow quick navigation of the XML graph/tree.

DISCOVER [7] and DBXplorer [2] work on top of a DBMS to facilitate keyword search in relational databases. They are middleware systems in the sense that they can operate as an additional layer on top of existing DBMSs. In contrast, the system we present is dedicated to providing efficient keyword proximity querying of XML databases, by using sophisticated execution and indexing techniques. Furthermore, we adopt an elaborate presentation method using interactive graphs of results. In contrast, DISCOVER and DBXplorer output a list of results, including trivial ones. Finally, we handle the inherent differences of XML from relational data by introducing the notion of target objects.

3 Architecture

The architecture of the system is shown in Figure 7. During the preprocessing stage, the *master index* is created, which is an inverted index that stores for each keyword k a list of elements that contain k. We store a set of connection relations [8] that allow us to quickly navigate in the XML graph.

In the query processing stage, we retrieve from the master index the elements that contain the keywords and generate, in a pipelined way, trees of target objects that contain all the keywords. In the case where the database has IDREFs we exploit the information stored in the connection relations to efficiently discover the connections between the keywords. On the other hand, if the database is an XML tree, we employ efficient algorithms that execute in time linear in the size of the keyword lists.

Finally, the results are presented to the user. The system offers two presentation methods: displaying a presentation graph for each different result schema (Figure 5), or displaying a full list of results (Figure 4), where each result is a tree that contains every keyword exactly once. The former method offers a more compact and non-redundant representation, while the latter favors faster response times.

4 Presentation Graph

In its simplest result presentation method (Figure 4) the system outputs results page by page, as in web search engine interfaces. The smaller results, which are intuitively more important to the user, are output first. This naive presentation method provides fast response times, but may flood the user with results, many of which are trivial. In particular, as we explained earlier, a redundancy similar to the one observed in multi-valued dependencies emerges often. Displaying to the user results involving multi-valued dependencies is overwhelming and counter-intuitive. We address the problem by providing an interactive interface that allows navigation and hides trivial and duplicate results, as discussed below.

The system's interactive interface presents the results grouped by the schema to which they conform. Intuitively,

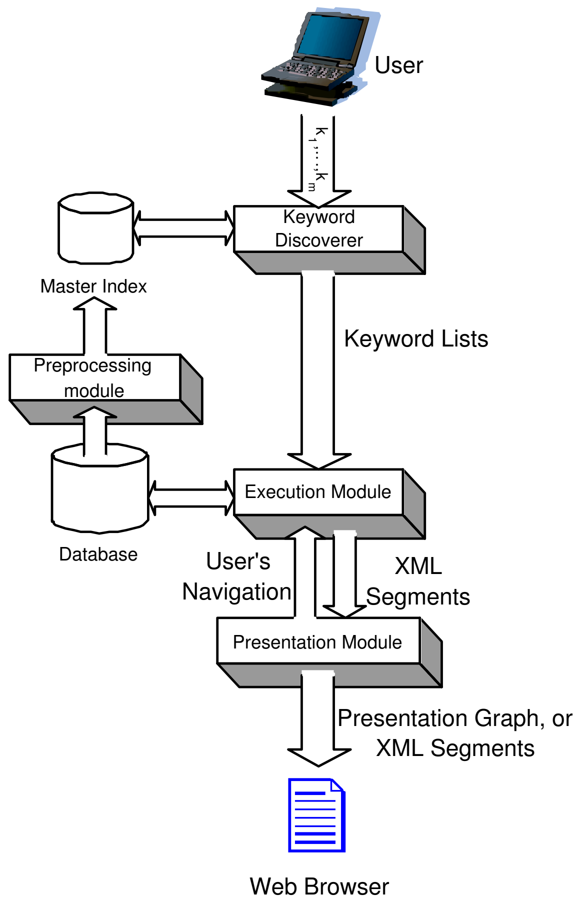


Figure 7: Architecture

results of the same schema have the same types of target objects and the same type of connections between them. The results are grouped for each schema to summarize the different connection types between the keywords and to simplify the visualization of the result.

The system compacts the results' representation and offers a drill-down navigational interface to the user. In particular, a *presentation graph* $PG(S)$ is created for each result schema S. The presentation graph contains all nodes that participate in a result of type S. A sequence of subgraphs $PG_0(S), \ldots, PG_n(S)$ of $PG(S)$ is *active* and is displayed at each point, as a result of the user's actions. The initial subgraph, $PG_0(S)$, is a single, arbitrarily chosen result m of type S, as shown in Figure 5.

An expansion $PG_{i+1}(S)$ of $PG_i(S)$ on a node n of type N is defined as follows. All distinct nodes n', of type N, of every result m' of type S are displayed and marked as *expanded* (Figure 6). In addition, a minimal number of nodes of other types are displayed, so that the expanded nodes appear as part of the results. In the demo, an expansion on a node n occurs when the user clicks on n. Notice also that if the expanded nodes are too many to fit in the screen then only the first 10 are displayed.

On the other hand, a contraction $PG_{i+1}(S)$ of $PG_i(S)$ on an expanded node n of type N is defined as follows. All nodes of type N, except for n, are hidden. In addition, a minimum number of nodes of types other than N (those that are not connected to n by some result) are hidden.

The presentation of the results of a keyword query by the interactive presentation graphs evokes the following requirements for the execution unit: First the top result of each result schema, which is the initial presentation graph, must be computed very quickly to provide a quick initial response time to the user. Second the expansion of the presentation graph must be performed on demand. This cannot be done simply by moving the cursor of some query we submit to the underlying database. Instead, when a user clicks on a node, a new minimal set of focused queries is sent to the database.

5 Status of the Demo

The demo of the system on the DBLP database, with HTML interface shown in Figure 4 is available at http://www.db.ucsd.edu/XKeyword. The presentation graphs interface is implemented as a standalone application (Figures 5 and 6). This application facilitates keyword queries on the DBLP dataset with subsequent interactive exploration of the results.

References

[1] http://www.xyzfind.com.

[2] S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer: A system for keyword-based search over relational databases. *ICDE*, 2002.

[3] S. Amer-Yahia, S. Cho, and D. Srivastava. Tree pattern relaxation. *EDBT*, 2002.

[4] G. Bhalotia, C. Nakhey, A. Hulgeri, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using BANKS. *ICDE*, 2002.

[5] D. Florescu, D. Kossmann, and I. Manolescu. Integrating keyword search into XML query processing. *WWW9*, 1999.

[6] R. Goldman, N. Shivakumar, S. Venkatasubramanian, and H. Garcia-Molina. Proximity search in databases. *VLDB*, 1998.

[7] V. Hristidis and Y. Papakonstantinou. DISCOVER: Keyword search in relational databases. *VLDB*, 2002.

[8] V. Hristidis, Y. Papakonstantinou, and A. Balmin. Keyword proximity search on XML graphs. *ICDE*, 2003.

[9] Y. Kanza and Y. Sagiv. Flexible queries over semistructured data. *PODS*, 2001.

[10] J. Plesník. A bound for the Steiner tree problem in graphs. *Math. Slovaca 31*, pages 155–163, 1981.

[11] J. D. Ullman, J. Widom, and H. Garcia-Molina. Database Systems: The Complete Book. *Prentice Hall*, 2001.

[12] W3C. XML schema definition, 2001. W3C Recommendation available at http://www.w3c.org/XML/Schema.