EXTRUCT: Using Deep Structural Information in XML Keyword Search

Arash Termehchy Marianne Winslett
Department of Computer Science, University of Illinois, Urbana, IL 61801
{termehch,winslett}@cs.uiuc.edu

ABSTRACT

Users who are unfamiliar with database query languages can search XML data sets using keyword queries. Previous work has shown that current XML keyword search methods, although intuitive, do not effectively use the data's structural information and provide poor precision, recall, and ranking for most queries. Based on an extension of the concept of information theory, we have developed principled frameworks called *normalized total correlation* (NTC) and *normalized term presence correlation* (NTPC) to measure the relevance of candidate answers to keyword queries. We demonstrate EXTRUCT, an XML keyword search interface that uses NTC and NTPC. An extensive empirical evaluation over two real-world XML DBs has shown that EXTRUCT has better precision and recall and provides better ranking than all previous approaches. We demonstrate EXTRUCT, along with seven other keyword search systems for four real-world XML data sets, using prepared queries as well as queries from the audience. The demonstration shows that using deep structural information increases the effectiveness of XML keyword search systems considerably.

Categories and Subject Descriptors: H.2.4 [Database Management]: Query Processing

General Terms: Algorithms, Designs, Performance

1. INTRODUCTION

Many users of XML databases are not familiar with concepts such as schemas and query languages. Keyword search [1, 2, 3, 5, 9] has been proposed as an appropriate interface for such users; each subtree that contains the query terms is a candidate answer for the input query. Since there are usually many such subtrees, the challenge is to identify the subtrees most closely related to the user's query, since the query is not framed in terms of the data's actual structure. Current systems filter subtrees they consider irrelevant to the query [9, 3, 2, 5, 1]. Some filtering methods extend IR techniques and do not take advantage of the structural information of the data. Others use intuitively appealing heuristics based on shallow structural details, which desirable answers often violate. Hence, current methods do not filter many unrelated subtrees and/or filter many relevant answers. After filtering, the user is shown a huge mix of relevant and irrelevant subtrees where she has to manually find the desirable answers. To help address this problem, some systems rank the filtered subtrees [3, 2, 1], using a modified version of the ranking heuristics used in IR for XML text-oriented data. Hence, they do not effectively use fine-grained structural information available in non-text-oriented XML and are ineffective for many queries, as our experimental results illustrate. In this demonstration, we introduce an XML keyword search system called EXTRUCT (**E**ffective **X**ML Ranking Using Deep **Struct**ure) that does not filter out any candidate answer, and that exploits XML structure to rank its results while avoiding overreliance on shallow structural details [7, 8]. Our contributions in the demo:

- We explain our principled frameworks that define the degree of relatedness of query terms to XML subtrees, based on NTC and NTPC, which are extended versions of the concepts of data dependencies and mutual information.
- Through examples, we show how previous approaches rely on intuitively appealing but ad hoc heuristics, causing low precision, recall, and ranking quality. We show how NTC and NTPC avoid these pitfalls.
- In domains where IR-style statistics or PageRank can be helpful in ranking query answers, NTC and NTPC can be combined with such measures to improve the precision of query answers. We explain how to combine NTC and NTPC with traditional IR ranking methods, so that query answer rankings consider both content and structure, and show its effect on example queries.
- We demonstrate how to deploy NTC and NTPC in EXTRUCT, using a two-phase approach. The first phase is a precomputation step that extracts the meaningful substructures from an XML DB, before the query interface is deployed. During normal query processing, we use the results of the precomputation phase to rank subtrees containing the query terms.
- Since naive methods are prohibitively inefficient for the precomputation step, we explain how to use novel optimization and approximation techniques to reduce precomputation costs, and show how these techniques reduce precomputation time by orders of magnitude, without affecting ranking quality.
- Throughout the demonstration, we use example queries

Proceedings of the VLDB Endowment, Vol. 3, No. 2

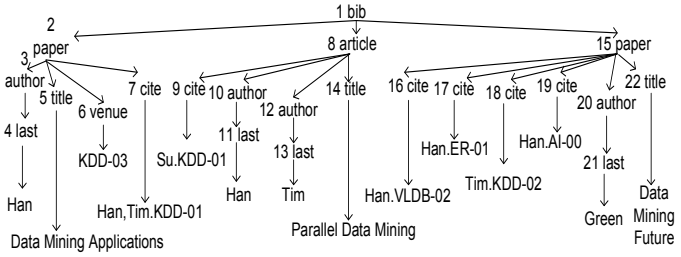

Figure 1: DBLP database fragment

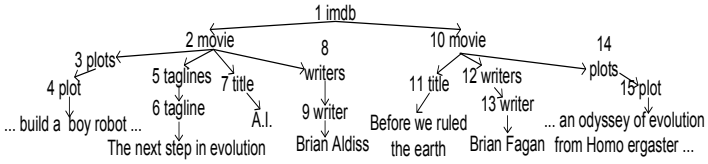

Figure 2: IMDB database fragment

over IMDB and DBLP provided by previous users and current audience members to (1) show the surprisingly bad answers produced by intuitively reasonable heuristics, and (2) show how EXTRUCT improves these answers.

2. BACKGROUND & MOTIVATION

We model an XML DB as a tree $T = (r, V, E, L, C, D)$, where V is the set of nodes in the tree, $r \in V$ is the root, E is the set of parent-child edges between members of V, $C \subset V$ is a subset of the leaf nodes of the tree called *content nodes*, L assigns a label to each member of $V - C$, and D assigns a data value (e.g., a string) to each content node. A *keyword query* is a sequence $Q = t_1 \cdots t_q$ of terms. A subtree S is a *candidate answer* to Q iff its content nodes contain at least one instance of each term in Q, and each of its content nodes contains such an instance. The root of a candidate answer is the *lowest common ancestor* (LCA) of its content nodes. When no confusion is possible, we identify a candidate answer by its root's node number. Trees T_1 and T_2 are **label isomorphic** if the nodes of T_1 can be mapped to the nodes of T_2 in such a way that node labels are preserved and the edges of T_1 are mapped to the edges of T_2. A **pattern** concisely represents a maximal set of isomorphic trees (its **instances**) [7]. For instance, pattern *bib/paper/title* corresponds to trees *1/8/14* and *1/15/22* in Fig. 1. The **value** of a subtree (if it exists) is the content associated with its leaves. For example, the value of *1/2/3/4* in Fig. 1 is ("Han"). The values of a pattern are all the values of its instances. A pattern is a **path** if it has only one leaf. The **size of a pattern** is the number of paths it contains. A **root-pattern** is a pattern whose root is the root of the DB. Except where otherwise noted, we consider only root-patterns in this paper.

First we show that the current pruning techniques deliver low precision and recall. The *baseline method* for XML keyword search returns *every* candidate answer, (with modest refinements in XRANK [3]). Consider query Q_1 : *Han KDD* in Fig. 1, which shows fragments of DBLP (*dblp.uni-trier.de*). The baseline method returns a relatively large set of subtrees rooted at nodes 1, 2, 7, 8, and 15 as answers for Q_1. However, subtrees rooted at node 1 are not helpful, as they merely show that these two terms both occur in the DB. Candidate answer *15/16,15/18* shows that a paper by *Han* and another paper published in KDD are cited by the same paper. This relationship is not as strong as that of subtree *2/3,2/6*, which represents a KDD paper by Han; hence it is far less interesting.

Methods such as SLCA and MaxMatch rely on the intuitively appealing heuristic that far-apart nodes are not as tightly related as nodes that are closer together [9, 5]. Thus, they eliminate every candidate answer whose root is an ancestor of the root of another candidate answer. This heuristic filters many relevant answers and returns many irrelevant subtrees. SLCA and MaxMatch do not return subtree 2 as an answer to Q_1 because its root is the parent of another candidate answer, subtree 7. They also return subtree 15 which is not a desired answer to Q_1.

XSearch and CVLCA remove every candidate answer having two non-leaf nodes with the same label [2, 4]. 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*. However, sometimes there *are* meaningful relationships between similar nodes, even in a DB with few entity types. Suppose a user submits query Q_2 : *Han Tim* to find the publications written by Han and Tim in the DB fragment in Fig. 1. DL does not return subtree 8, which is the desired answer to Q_2. Also, DL is not an ideal way to detect uninteresting relationships. It returns uninteresting candidate answers 1 and 15 for Q_2. XReal [1] uses IR statistics and filters out entity types that do not contain many of the query terms. For instance, DBLP has few books about *Data Mining*, so XReal filters out all *book* entities when answering query Q_3 : *Data Mining Han* – even Han's textbook. Also, the DBA has to specify the depth of desired entity types.

After pruning, some approaches rank the candidate answers. XRANK [3] uses a PageRank-based approach to rank subtrees. PageRank is effective only for certain domains and relationships, and is not intended for ranking subtrees [8]. For instance, node 15 has more links than nodes 8 and 2 in Fig. 1, but it is not more important than them. XSearch and XReal consider each subtree as a small document, ranking subtrees higher if they have more of the query terms. This heuristic still does not use structural information effectively. For instance, subtree 15 contains more terms of Q_3 than subtrees 8 and 2 in Fig. 1. However, the user submitting Q_3 is more likely to want the publications by Han about data mining than the data mining publications that cite papers by Han. Hence, subtrees 2 and 8 should rank higher than 15.

As do IR techniques, these methods penalize longer content nodes. Sometimes shorter content nodes, such as the children of *last* nodes in Fig. 1, are more important than longer ones, such as children of *cite* nodes. However, this is not always true. For instance, consider the IMDB fragment from *www.imdb.com* in Fig. 2. Because tag lines (the famous sentences in movie trailers) are less indicative of a movie's content than plot lines, the best answer to query Q_4 : *Evolution Brian* is the subtree rooted at node 10. As the *plot* field is longer than the *tagline* field, penalizing long fields will be misleading. The original IMDB DB contains

many other fields that are shorter and less informative than *plot* and/or *title*, such as *goofs* (mistakes in the movies) and *trivia*.

The *distance* between nodes n and m is the number of nodes in the path between n and m. XSearch and XReal rank higher the subtrees where the nodes containing query terms have smaller distances. This heuristic is sometimes helpful, but often misleads. For instance, nodes 17 and 22 are closer than nodes 4 and 5 in Fig. 1. However, subtree 2 is more relevant than subtree 16 for Q_3. Also, our empirical study shows that most candidate answers have the same distance.

The first key shortcoming of all these methods are that they *filter out answers instead of ranking them*. Second, they *rely on shallow structural properties* and/or *extensions of IR-style methods* to rank answers. Since these methods do not effectively use structural information in the data, they are ineffective for many queries, as our experimental results illustrate.

3. USING STRUCTURAL INFORMATION

The examples of Section 2 illustrate that there are two basic challenges in ranking candidate answers. First, a keyword search system must determine whether a candidate answer represents a strongly and meaningfully related portion of the data, and provide a metric to measure this property. Second, it must determine the similarity between candidate answers and the input query. For instance, in Fig. 1 subtree *15/20,15/22* represents a meaningful entity, while subtrees rooted at 1 contain a set of loosely related nodes. Thus, the former is more interesting for users than the latter and must rank higher. However, not every subtree rooted at a *paper* node represents an interesting and meaningful substructure. As mentioned in Section 2, subtree *15/16,15/18* represents a less interesting data fragment than subtree *2/3,2/6*.

The patterns of candidate answers represent their structural information. Each pattern provides a relationship between its paths, where every value of the pattern relates the values of its paths. We claim that the more correlated the values of the paths of a pattern are, the more it represents a meaningful and interesting relationship between its paths, i.e., a strongly related portion of the data. Consider patterns q_1 : *paper/title,paper/cite* and q_2 : *paper/title,paper/author* in Fig. 1. Instances of these patterns are candidate answers to Q_3. Each *title* node value is associated with more *cite* node values than *author* node values, on average. The same is true in the original DBLP DB, where each *title* is associated with 2.3 *author*s and 9.4 *cite*s on average. Also, the average author publishes less than 2 papers, while a cited paper is cited more than 2 times on average. Thus, q_2 represents an entity more strongly than q_1 and its instances should appear before instances of q_1 in the ranked list of answers. This complies with the observations mentioned in Section 2.

To capture the correlation among the paths of a pattern, we use **normalized total correlation** (NTC) [7]. NTC measures the correlation of a pattern; its value for a pattern p with paths $p_1, \ldots, p_n$, $n > 1$ is:

$$NTC(p) = g(n) \times \frac{\sum_{1 \le i \le n} H(p_i) - H(p)}{H(p_1, \ldots, p_n)}. \quad (1)$$

where $H(p)$ and $H(p_i), 1 \le i \le n$ are the entropies of pattern p and its paths, respectively [7]. Since users prefer smaller patterns, we penalize larger patterns using function $g(n)$; $g(n) = n^2/(n-1)^2, n > 1$ performs well in practice [7]. For patterns of size 1, we rank the ones with more entropy higher. Considering all instances in the original DBLP, the NTC of q_1 and q_2 is 1.09 and 1.74, respectively, which confirms our analysis.

However, NTC does not measure the correlation for patterns with long text fields well. Different movies have different plot lines and tag lines. In the original IMDB, on average each movie has more *plot* nodes than *tagline* nodes, so the NTC of *movie/taglines/tagline,movie/writers/writer* is 1.48 and the NTC of *movie/plots/plot,movie/writers/writer* is only 1.37. As the *plot* field is longer than the *tagline* field, penalizing long fields will not solve the problem. The problem can occur for short text fields as well [8]. Thus, intuitively, we should consider the individual components (words) of each value when computing correlations, both for long and short fields. For instance, the words in movie tag lines are not as representative of the movie's subject as the words in its plot lines. Thus in IMDB DB, the terms in the values of the field *writer* are more correlated with those of *plot* than *tagline*. We call $W(r_1 : w_1, \ldots, r_n : w_n)$ a **term** of a pattern instance r containing path instances $r_i, 1 \le i \le n$, with **value** $(r_1 : v_1, \ldots, r_n : v_n)$, if w_is are non-stop words that occur in values v_is, respectively. The **terms** of pattern p containing paths $(p_1, \ldots, p_n)$ are the union of the sets of terms of its instances. For instance, p_1 : *Han,* p_2 : *Data* is a term of pattern p :*bib/paper/cite,bib/paper/title* in Fig. 1. Each term $W(p_1 : w_1, \ldots, p_n : w_n)$ is associated with 2^n possible **events**. Each event takes the form $E(p_1 : f(w_1), \ldots, p_n : f(w_n))$, where each $f(w_i)$ is either w_i or $\bar{w_i}$, depending on whether w_i does or does not occur in $p_i, 1 \le i \le n$. Similar to NTC, we define **normalized total presence correlation** (NTPC) of term $W(p_1 : w_1, \ldots, p_n : w_n)$ of pattern p as:

$$NTPC(W) = g(n) \times \frac{\sum_{1 \le i \le n} H_p(w_i) - H_p(W)}{H_p(W)}. \quad (2)$$

where $H_p(W)$ and $H_p(W_i), 1 \le i \le n$ are the entropies of pattern W and w_i, respectively [8]. As explained in [8], we measure the correlation of a pattern by averaging over the top-k correlated terms, where k is reasonably large.

NTPC-based ranking successfully handles all the examples described earlier [8]. For instance, in the full IMDB the NTPC of *movie/taglines/tagline,movie/writers/writer* is 1.25, while the NTPC of *movie/plots/plot,movie/writers/writer* is 1.49. To capture the content similarity between candidate answers and the input query, we combined NTPC (and NTC) with pivoted normalization (PN) [6], an IR-style content ranking formula that we customized for XML. We control the relative weight of NTPC (and NTC) and PN as follows:

$$r(t) = \alpha NTPC(t) + (1 - \alpha) ir(t), \quad (3)$$

where $ir(t)$ is the content score of the candidate answer, computed based on the classical PN formula, and α is a constant that controls the relative weight of structural and contextual information in ranking. Based on our empirical evaluation, we set the value of α to 0.8 when combining PN with NTC and NTPC.

4. SYSTEM ARCHITECTURE

	NTPC	NTC	XReal	SLCA	MaxMatch	CVLCA	XSearch	XRank
Precision	0.611	0.599	0.566	0.566	0.545	0.048	0.046	0.050
Recall	0.985	0.965	0.918	0.798	0.798	0.975	0.976	0.975

Table 1: Average precision and recall for 40 user supplied queries over IMDB

	NTPC	NTC	XSearch	XReal	PN	XRank
IMDB	0.701	0.510	0.612	0.587	0.478	0.431
DBLP	0.834	0.834	0.794	0.790	0.621	0.591

Table 2: MAP for DBLP and IMDB queries

NTC and NTPC Computation: EXTRUCT computes the values of NTC and NTPC for all patterns in the DB in a separate phase before the first queries are submitted to the system. If the DB does not undergo drastic structural changes that introduce new node types and patterns, this computation need never be repeated [7]. The naive method to compute NTCs and NTPCs is so inefficient as to be impractical [7, 8]. As explained in [7], we expect the size of user's ideal answer to be quite low. Thus, the size of the patterns we seek does have a domain-dependent upper bound $MCAS$ (maximum candidate answer size). For instance, empirical studies suggest that 4 is a reasonable $MCAS$ value for bibliographic DBs [7]. we also approximate the NTC of larger patterns using the NTC of the smaller patterns in the DB [7]. Our empirical studies show that we can get the same ranking by approximating the NTCs of the patterns of size 4 and 5 using the exact NTC values of patterns up to size 3. From the properties of total correlation, it follows that infrequent and highly frequent terms have relatively low NTPC. Thus, we remove all terms in path p whose frequencies are less than $\epsilon|p|$ or more than $(1-\epsilon)|p|$, where $0 < \epsilon < 1$. The algorithms to compute NTC and NTPC generate new patterns of size n using the information of the patterns of size $n-1$ and compute their NTC and NTPC values. The details of the algorithms and their performances are in [7, 8].

Query Processing: Our query processing algorithm, *SA3* [8], finds each candidate answer using a stack-based method, looks up the NTC and NTPC values of the pattern of the candidate answer, and ranks the answer based on its $r(.)$ value. At startup, SA3 stores XML node information in a NODES table in BerkeleyDB (*www.oracle.com/berkeleydb*). SA3 builds an inverted index for the text information in the NODES table. It also creates an additional index on the parental information of the DB nodes to present the the full subtree to the user. We have performed an extensive user study to measure the effectiveness of NTC and NTPC methods[7, 8]. Table 1 summarizes the recall and precision of all methods discussed in Section 2, NTC, and NTPC, for IMDB by averaging over all queries in the workload. NTPC and NTC have higher precision and recall on IMDB queries than other methods. NTPC performs better than NTC as IMDB contains many long text fields. Table 2 shows the Mean Average Precisions (MAPs) of the ranking methods discussed in Section 2, NTC, NTPC, and pivoted normalization (PN) method without any structural information. Generally, NTPC and NTC provide better ranking than other methods. They have the same MAP for DBLP DB as there is not any long text field in DBLP DB. NTPC delivers larger MAP for IMDB DB. More results on effectiveness of EXTRUCT are in [7, 8].

5. DEMONSTRATION

In this demo, we present the challenges in XML keyword search. We show how current approaches deliver low recall, precision, and ranking quality because they use pruning instead of ranking and do not use deep structural information of the data. We discuss the NTC and NTPC based ranking methods, justify their approaches, and show how they overcome the effectiveness deficiencies of other XML keyword search methods. We have implemented all methods discusses in Section 2. We have created a GUI interface for these methods which allows users to select the XML DB, submit queries, and observe the search results. Throughout the demo we show how NTC and NTPC provide better ranking than other approaches using sample queries we collected in our user studies. The audience can also submit queries and observe the importance of using deep structural information in XML keyword search. Furthermore, the results from our user study will be available.

6. ACKNOWLEDGMENTS

We thank ChengXiang Zhai for helpful discussions and feedbacks. This work is supported by NSF grant number 0938071.

7. REFERENCES

[1] Z. Bao, T. W. Ling, B. Chen, and J. Lu. Effective XML Keyword Search with Relevance Oriented Ranking. In *ICDE*, 2009.

[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] G. Li, J. Feng, J. Wang, and L. Zhou. Effective Keyword Search for Valuable LCAs over XML Documents. In *CIKM*, 2007.

[5] Z. Liu and Y. Chen. Reasoning and Identifying Relevant Matches for XML Keyword Search. In *VLDB*, 2008.

[6] C. Manning, P. Raghavan, and H. Schtze. *An Introduction to Information Retrieval*. Cambridge University Press, 2008.

[7] A. Termehchy and M. Winslett. Effective, Design-Independent XML Keyword Search. In *CIKM*, 2009.

[8] A. Termehchy and M. Winslett. Keyword Search for Data-Centric XML Collections with Long Text Fields. In *EDBT*, 2010.

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