Effective, Design-Independent XML Keyword Search

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

ABSTRACT

Keyword search techniques that take advantage of XML structure make it very easy for ordinary users to query XML databases, but current approaches to processing these queries rely on intuitively appealing heuristics that are ultimately ad hoc. These approaches often retrieve irrelevant answers, overlook relevant answers, and cannot rank answers appropriately. To address these problems for data-centric XML, we propose *coherency ranking* (CR), a domain- and database design-independent ranking method for XML keyword queries that is based on an extension of the concept of mutual information. With CR, the results of a keyword query are invariant under schema reorganization. We analyze how previous approaches to XML keyword search approximate CR, and present efficient algorithms to perform CR. Our empirical evaluation with 65 user-supplied queries over two real-world XML data sets shows that CR has better precision and recall and provides better ranking than all previous approaches.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems

General Terms

Algorithms, Performance

1. INTRODUCTION

Many users of XML databases are not familiar with concepts such as schemas and query languages. Keyword search [4, 6, 7, 10, 13, 18] has been proposed as an appropriate interface for them; 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). 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 XML schemas. Thus current approaches suffer from low precision, low recall, or both. They 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. In this paper, we propose a ranking approach for keyword queries that exploits XML structure while avoiding overreliance on shallow structural details, and has higher precision and recall and better ranking quality than previous approaches. Our contributions:

- We develop a theoretical framework that defines the degree of relatedness of query terms to XML subtrees, based on *coherency ranking* (CR), an extended version of the concepts of data dependencies and mutual information.
- We show how CR avoids pitfalls that lower the precision, recall, and ranking quality of previous approaches, which rely on intuitively appealing but ad hoc heuristics that we analyze within the framework of CR. For a given set of content, the ranking produced by CR is invariant under equivalence-preserving reorganizations of the schema, thus avoiding reliance on shallow structural details.
- CR finds the most probable intention(s) for queries containing terms with multiple meanings. For instance, in the query *Maude References*, the term *Maude* can refer to a programming language or an author's last name.
- We show how to deploy CR in XML query processing, using a two-phase approach. The first phase is a preprocessing 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 preprocessing phase to rank the substructures in the DB that contain the query keywords. 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.
- Since naive methods are prohibitively inefficient for the preprocessing step, we present and evaluate optimization and approximation techniques to reduce preprocessing costs. Our experiments show that these optimizations

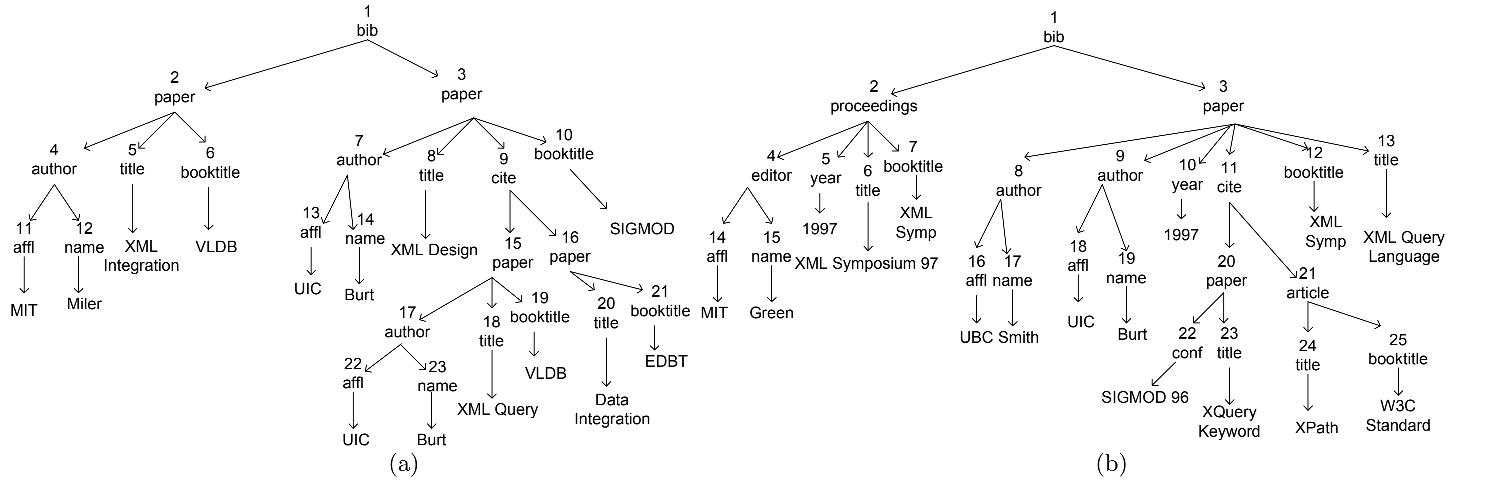

Figure 1: DBLP database fragments, with DBLP's native XML schema

- improve preprocessing performance by orders of magnitude, and the approximation errors do not affect answer ranking quality.
- Our extensive user study with two real-world XML data sets shows that CR is efficient at query time, and has higher precision and recall and provides better ranking than previous approaches.
- In domains where information-retrieval-style statistics such as term frequency (TF) and inverse document frequency (IDF) [13] or PageRank can be helpful in ranking query answers, CR can be combined with such measures to improve the precision of query answers. Our empirical study shows that combining the two provides a modest benefit for CR.

Although our focus in this paper is on XML keyword queries, CR is also appropriate for relational and graph DBs (e.g., RDF, OWL, XML with ID/IDREF), and for natural language queries. Our preprocessing algorithm could also be used for purposes other than keyword search, such as finding highly correlated elements and patterns [8] in XML databases.

In the reminder of the paper, Section 2 discusses problems with current XML keyword query processing. Section 3 introduces CR. Section 4 presents optimization and approximation techniques for DB preprocessing. Section 5 describes our implementation, Section 6 presents empirical results, and Section 7 concludes the paper.

2. MOTIVATION AND RELATED WORK

2.1 Basics and Current Approaches

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. We assume no node has both leaf and non-leaf children, and each node has at most one leaf child; other settings can easily be transformed to this one. Each node can be identified by its *path* from the root; e.g., node 5 in Figure 1(a) has path */bib/paper/title*. Each *subtree* $S = (r_S, V_S, E_S, L_S, C_S, D_S)$

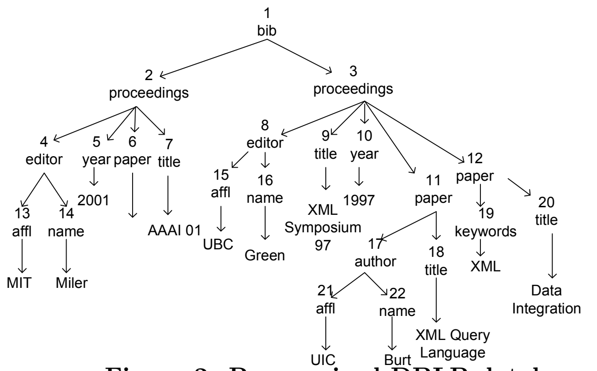

Figure 2: Reorganized DBLP database

of T is a tree such that $V_S \subseteq V$, $E_S \subseteq E$, $L_S \subseteq L$, and $C_S \subseteq C$. 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. (The containment test can rely on stemming, stop words, synonym tests, and other refinements; our experiments use only stop words.) 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. The IR community has been developing retrieval techniques for *text-centric* XML [13], where structures are simple and structural information plays almost no role in retrieval. The metadata of such content (e.g., title, author, conference) is the primary target of keyword search in data-centric XML, and the techniques we propose in this paper are not intended for use with very long text fields.

The consensus in keyword search for relational and XML DBs is that the best answers are the most specific entities or data items containing the query terms [2, 4, 6, 7, 10, 11, 18]. Thus in XML DBs, answers should include the most specific subtrees containing the query terms. The specificity of a subtree 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, (with modest refinements in [6, 7]). For instance, consider the DBLP fragment from www.informatik. uni-trier.de/~ley/db 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. For example, for query Q_1 = *Integration EDBT*, the baseline approach returns the desired answer of node 16, but also the unhelpful root node. In Q_2 = *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 approach should not return these answers, or else rank them below helpful answers.

Researchers have worked to boost the precision of the baseline method by filtering out unhelpful candidate answers. One approach eliminates every candidate answer whose root is an ancestor of the root of another candidate answer [18]; 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 for Q_2; 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 = *XML Burt*, nodes 3 and 15 are desirable; but since node 3 is an ancestor of node 15, node 3 will not be returned. MaxMatch [12] improves the precision of SLCA, but still has SLCA's effectiveness problems. It does not rank its answers either. XSearch [4] removes 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. XSearch ranks the remaining answers by the number of nodes they contain and their TF/IDF. 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 query Q_4 = *SIGMOD XPath*, DL returns node 11, which is undesirable. DL cannot detect uninteresting relationships between nodes of different types, either; it does not filter out node 1 for query Q_5 = *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. XSearch's size-based ranking scheme does not help to avoid DL's pitfalls. The MLCA [11] and VLCA [10] approaches have almost the same problems, as described in the long version of this paper [14].

XRank [6] finds the LCAs using a modest modification of the baseline approach and uses a PageRank-based approach to rank subtrees. It has the same precision problems as the baseline method. Also, PageRank is effective only in certain domains and relationships and is not intended for ranking subtrees [14]. For instance, all the *Proceedings* tags

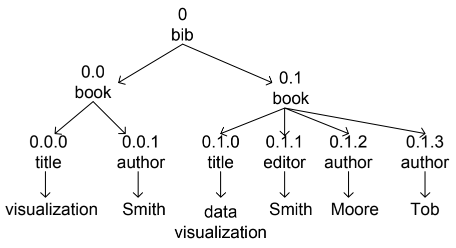

Figure 3: A bibliographic database

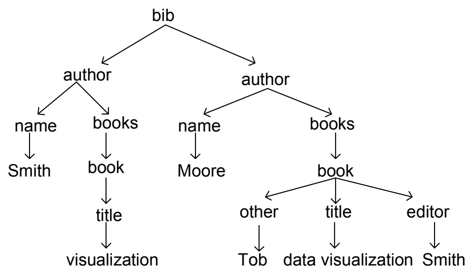

Figure 4: Reorganized bibliographic database

in Fig. 1(a) have the same PageRank. XReal [1] filters out entity types that do not contain many of the query terms. Then it ranks subtrees higher if they and their entities' types have more of the query terms. For instance, DBLP has few books about *Data Mining*, so XReal filters out all *book* entities when answering the query *Data Mining Han* – even Han's textbook. XReal does not consider the relationship *between* nodes of a subtree when it ranks its answers, so irrelevant subtrees can be ranked very high. For query *SIGMOD 1997*, XReal ranks the subtree rooted at node 3 in Fig. 1(b) higher than the 1997 papers with *booktitle* SIGMOD. This is because *SIGMOD* occurs very frequently in subtrees rooted at *cite*. Even if we ignore the importance of the *cite* entity type in XReal ranking, XReal still ranks papers published in 1997 below papers that cite multiple articles and papers published in 1997. Generally, IR ranking techniques do not take advantage of XML's structural properties effectively.

The first key shortcoming of all these methods is that they *filter out answers instead of ranking them* and/or they *rely on shallow structural properties to rank answers*. Since these methods rely on ad hoc heuristics, they are ineffective for many queries, as our experimental results illustrate. Second, these methods are dependent on the current schema of the database. If the schema is reorganized in an equivalence-preserving manner, their query answers may change.

2.2 Toward More Meaningful Ranking

Current approaches implicitly assume that all relationships between nodes in a candidate answer have the same strength. For example, consider the original XML version of DBLP in Fig. 1(b). Current approaches assume that the relationship between *booktitle* and *author* is as important as the relationship between *title* and *author*. However, the reasonable redesign of DBLP in Fig. 2 suggests that a paper's title is more closely related to its author 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 *booktitle* and *author* children match. This distinction is particularly important because flat XML DBs (like the original version of DBLP) are proliferating, due to the use of semantic tagging of text corpora using information extraction codes [3]. Thus if a paper's title really is more closely related to its author than to its proceedings title, we need to be able to reflect this in our ranking of query answers, *no matter what schema is used.* Even in well-organized XML schemas, relationships between attributes at the same level of a subtree can have different strengths. For query *visualization Smith* in Fig. 3, the subtree on the left is more important: the user probably wants books Smith wrote, not books Smith edited. In other words, the relationship between *title* and *author* is more important than the relationship between *title* and *editor*.

One option is to rank smaller subtrees higher, where the size of the subtree is defined as the length of the longest path to its root. This heuristic is too ad hoc to work well in general: consider the subtrees rooted at *cite* and *bib* in Fig. 1(a), which contain paper titles. They have the same size, but the relationship between the *title*s under *cite* is more meaningful than between the *title*s under *bib*. Even inside the subtree rooted at *cite*, the *title*s are more related to each other than the *booktitle*s. Also, we need to be able to rank subtrees with the same size. Heuristics based on physical distance depend on schema design details, so small changes in the schema can cause big changes in query answers. For example, consider the redesign of Fig. 3, shown in Fig. 4. The new design categorizes books based on their first authors, which makes the title closer to the name of the editor than it is to the book's author-even though the new design is still normalized. A similar line of research has been followed for keyword queries in relational DBs [2]. These approaches create a join tree whose tuples contain the query terms, and rank the tree based on its size.

In Fig. 2, node 17 has a closer relationship to node 18 than to node 9. This is because there is just one author associated with the paper title, whereas all the authors who have a paper in the proceedings are associated with the title of the proceedings. This suggests that the fewer instances of type B associated with each instance of type A, the stronger their relationship is. Researchers have combined this idea with distance based heuristics for keyword search in relational and graph DB systems [2, 19].

One approach ranks a tuple higher if many tuples link to it, but it links to just a few [2]. Another employs the notion of join selectivity to measure the degree of closure among a set of tables [19]. However, these heuristics are misleading. First, if we compare the relationship of *title* with *editor* and *author* in Fig. 3, we see that there are more authors associated with a title than there are editors for the title. However, intuitively *title* is more closely related to *author* than to *editor*. As another example, consider *title*, *author*, and *booktitle* in Fig. 1. This heuristic says *author* is as strongly related to *title* as to *booktitle*. But intuitively, the former relationship is much stronger than the latter. Second, these measurements are not normalized by the number of distinct values for each type. For example, suppose that type B has just three values. Having a distinct value of type A associated with just two values of type B does not mean that the relationship is very meaningful. Yet having the same association when B has over 100 distinct values could mean that the relationship between the two is very close. Third, the join-selectivity method does not address the case where the join path does not cover all the tuples in the relations. For instance, consider a university DB where each professor advises a few students and a few professors offer on-line courses for hundreds of students. Since the on-line course join path does not cover all tuples in the faculty relation, its join selectivity can be as small as or even smaller than the advisor relationship's. Finally, these approaches do not rank different candidate tuples that come from the same table. These problems, plus the lack of certain other forms of normalization discussed later, mean that these heuristics will not rank answers well in general.

The right intuition is that type A is more related to type B if each instance of type A is associated with relatively few instances of type B **and** each instance of type B is associated with relatively few instances of type A. In other words, from the value of one type we can predict the value of the other. The closer this association is between the types in a subtree, the more the subtree represents a meaningful and coherent object. This intuition can be generalized to more than two types, and can be normalized based on the number of values of each type. *Mutual information* can help capture this relationship. Mutual information and other statistical measurements are used in the data mining community to find correlated data items [8]. However, as explained in detail in the long version of this paper [14], we cannot directly use these approaches for our problem. In general, these approaches are not suitable for use with non-binary variables such as XML paths; do not provide exact correlations; rely on upward closure, which does not hold for our applications; and do not consider all the relationships between nodes that are important for ranking query answers. As shown in our experiments, in application domains where PageRank or IR ranking techniques are applicable, they can be combined with CR by weighting and adding the ranks suggested by each approach.

3. COHERENCY-BASED RANKING

In this section we ignore all non-content leaf nodes, as they do not affect rankings. Consider the depth-first traversal of a tree, where we visit the children of a node in the alphabetic order of their labels. Each time we visit a node, we output its number (or content); each time we move up one level in the tree, we output *-1*. The result is the unique *prefix string* for that tree [20]. For instance, the prefix string for the subtree rooted at node 4 in Fig. 1(a) is *4 11 MIT -1 -1 12 Miller -1 -1*. 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**). The pattern can be obtained from any member of the set, by replacing each node number in its prefix string by the corresponding label. For instance, pattern *bib paper title -1 -1* corresponds to trees *1 2 5 -1 -1* and *1 3 8 -1 -1* in Fig. 1(a). P_1 is a **subpattern** of P_2 if P_1's trees are all subtrees of P_2's trees.

Definition 1. A tree $S = (r, V_s, E_s, L_s, \emptyset, \emptyset)$ is a **root-subtree** of tree $T = (r, V, E, L, C, D)$ if S is a subtree of T and each leaf node of S is the parent of a leaf node of T.

For instance, *1 2 5 -1 6 -1 -1* is a root-subtree in Fig. 1(a). If the root-subtree is a path, we call it a **root-path**. Each path from root to leaf in a root-subtree is a root-path, so each root-subtree contains at least one root-path. Intuitively, the *value* of a root-subtree is the content that was pruned from its leaves. For example, the value of *1 2 5 -1 6 -1 -1* is ("XML Integration", "VLDB").

Definition 2. Let S' be a subtree of T, and let S be a subtree of S', such that S is a root-subtree of T and every leaf of S' is also a leaf of T. Let $c_1, \ldots, c_n$ be the list of content nodes encountered in a depth-first traversal of S'. Then $(c_1, \ldots, c_n)$ is the **value** of S.

Every maximal set of isomorphic root-subtrees in a tree T corresponds to a pattern. Each root-subtree pattern includes one or more root-path patterns, corresponding to the root-paths of its root-subtrees. The **size** of a root-subtree pattern is the number of root-path patterns it contains. The **values** of a pattern are all the values of its instances. The only patterns we consider hereafter are those of root-subtrees. Information theory allows us to measure the association between the values of random variables in tables [5]. We now translate information theory metrics to apply to XML. Each root-path pattern p represents a discrete random variable that takes value a with probability $P(a) = \frac{1}{n} count(a)$, where $count(a)$ is the number of instances of p with value a and n is the total number of instances of p in the DB. In Fig. 1(a), if $p =$ *bib paper title -1 -1*, $P(p =$"XML Design"$) = \frac{1}{2}$. Since a root-subtree pattern defines the association between the patterns of its root-paths, the root-subtree pattern represents the joint distribution of random variables. For instance, the root-subtree pattern $t_1 =$ *bib paper title -1 booktitle -1 -1* represents an association between root-path patterns $p_1 =$ *bib paper title -1 -1* and $p_2 =$ *bib paper booktitle -1 -1*. The probability of each value of a root-subtree pattern can be defined in the same manner as the probability of each value of a root-path pattern. For example, the probability of $P(p_1 =$"XML Design", $p_2 =$"SIGMOD"$) = \frac{1}{2}$ in Fig. 1(a). Generally, one value of a root-path can be associated with one or more values of other root-path(s).

Intuitively, the entropy of a random variable indicates how predictable it is. The long version of this paper [14] reviews the definitions of entropy, joint entropy, and mutual information. Here we extend those concepts to apply to XML.

Definition 3. Given a pattern p that takes values $a_1, \ldots, a_n$ with probabilities $P(a_1), \ldots, P(a_n)$ respectively, the **entropy** of p is $H(p) = \sum_{1 \le j \le n} P(a_j) \lg(1/P(a_j))$.

When the pattern is not a root-path pattern, we refer to its entropy as *joint* entropy, because it explains the behavior of a joint random variable. We refer to the *joint* entropy of pattern t as $H(t)$ or $H(p_1, \ldots, p_n)$, where $p_1, \ldots, p_n$ are the root-path patterns of t. *Total correlation* [15] is closely related to mutual information; it measures the correlation between random variables.

Definition 4. The **total correlation** of the random variables $A_1, \ldots, A_n$ is:

$$I(t) = \sum_{1 \le i \le n} H(A_i) - H(A_1, \ldots, A_n). \tag{1}$$

We extend it to apply to XML DBs:

Definition 5. Let $p_1, \ldots, p_n$ be the root-path patterns of pattern t. The **total correlation** of t is:

$$I(t) = \sum_{1 \le i \le n} H(p_i) - H(p_1, \ldots, p_n). \tag{2}$$

To compute the total correlation of t, we consider only the instances of the root-path p_i that are subtrees of the instances of the root-pattern t. Total correlation is minimal when each value of each root-path pattern is associated with every value of the other root-path patterns; it is maximal when each value of one root-path pattern is associated with just one value of every other variable, and vice versa. Total correlation does not consider the possibility that a pattern may have higher entropy just because it has more values. Total correlation also goes up when one root-path pattern has more diverse values, even if it is not correlated with the other root-path patterns. For example, in Fig. 2, consider patterns $t =$ *bib proceedings editor name -1 -1 title -1 -1 -1* and $t' =$ *bib proceedings paper author name -1 -1 title -1 -1 -1*. Intuitively, the relationship between a paper and its author is as close as between a proceedings and its editor. But the root-paths ending with *paper title* and *paper author* have more values than those ending with *proceedings title* and *proceedings editor*, so $I(t) > I(t')$. We address this in a manner similar to that used to normalize mutual information [17]. We define the **normalized total correlation (NTC)** of t as:

$$\hat{I}(t) = f(n) \times (\frac{I(t)}{H(p_1, \ldots, p_n)}). \tag{3}$$

It can be tricky to compare patterns of very different sizes. As discussed earlier, adding new nodes to a pattern that already contains all the query terms should not improve its rank. NTC solves this problem by dividing the total correlation by the sum of the entropies of all root-path patterns, thereby penalizing sets with more root-path patterns. Nonetheless, the range of total correlation values for $n > 1$ root-path patterns is $[0, \frac{n-1}{n}]$, as the maximum total correlation for n root-path patterns is reached when they all have the same entropy as their root-subtree pattern. Thus as n grows, the total correlation may also increase, which may penalize small candidate answers during ranking. NTC removes this bias by including a factor $f(n)$ that is a decreasing function of the answer size (number of root-path patterns) n, for $n > 1$. Based on the observations above, we can stipulate that as n grows, $f(n)$ must decrease at least as fast as $\frac{n}{n-1}$. Through empirical observation we found that $f(n) = n^2/(n-1)^2$ performs well in practice. Exploring options for $f(n)$ is an interesting area for future work.

Coherency ranking (CR) uses NTC to rank subtrees and filter answers for XML keyword queries, as follows. First we extend each candidate answer to be a root-subtree, by adding the path from the root of that answer to the root of the DB. Then we rank the candidate answers in the order given by the NTC of their root-subtrees' patterns. (We rank patterns with only one root-path highest, in descending order of entropy, as $f(n)$ is undefined for them.) If desired, one can set a low threshold $NTC_{min} \ge 0$ appropriate for the domain, and include only candidate answers whose patterns have NTC $> NTC_{min}$. With a good ranking function such as NTC, one does not really need this cutoff, as the worst answers appear last; but its use can improve the user experience by removing particularly unhelpful answers. For example, consider the query *XML SIGMOD* in Fig. 1(a). When computed over all of DBLP, the NTC of the *title* of a paper and its *booktitle* is 1.47, so node 3 will be returned as an answer. The subtree that connects nodes 18 and 10 will be returned as the second answer (NTC = 0.42), and the root node will be ranked last (NTC = 0). For $Q_2 =$ *Integration VLDB* in Fig. 1(a), pattern *bib paper cite paper title -1 -1 -1 -1 paper booktitle -1 -1* has NTC = 0; its candidate answers will be ranked last. Similarly, *bib paper title -1 booktitle -1 -1* (NTC = 1.47) outranks *bib paper cite paper title -1 -1 paper booktitle -1 -1 -1 -1* (NTC = 0.84). In this paper, we set $NTC_{min} = 0$ unless otherwise noted, which is an extremely conservative setting.

As CR does not rely on duplicate labels, it successfully ranks the answers for $Q_4 =$ *SIGMOD XPath* in Fig. 1(b), giving node 11 a lower rank than a paper published in *SIGMOD* whose title contains *XPath*. It also determines that there is no meaningful answer in Fig. 1(b) for $Q_5 =$ *UBC Green*. CR also correctly differentiates between the relationship of nodes 15 and 16 and nodes 2 and 3 in Fig. 1(a), and ranks the former higher because the NTC of two papers cited in the same publication is greater than the NTC of two papers listed in the same bibliography. As for recall, unlike previous methods, CR correctly returns both nodes 3 and 15 for $Q_3 =$ *XML Burt* in Fig. 1(a), and returns node 3 as an answer to query *Smith Burt* in Fig. 1(b). In fact, CR gives the intuitively desirable ranking for all the queries considered in Section 2. Moreover, its recall is equal to that of the baseline approach and higher than that of all other approaches discussed earlier, as long as the threshold is reasonably low. CR also performs well on different designs of the same DB, by avoiding *overreliance* on schema details and discovering the internal relationships itself.

NTC sheds light on the behavior of previously-proposed heuristics. In SLCA, the intuition is that closer nodes are more strongly correlated. DL assumes that repeated root-path patterns will have lower correlation than non-repeated root-path patterns in a subtree. MLCA assumes that two root-path patterns that are closer together will be more correlated than two that are far apart.

4. PRECOMPUTING COHERENCY RANKS

NTCs can be computed offline in advance with a populated version of the DB, and then used at query time. A naive approach to finding NTCs for all DB patterns is to generate all patterns, find all their instances, then compute the instances' NTCs. We can generate all patterns using existing data mining algorithms to generate candidate subtrees in a forest [20]. These works find trees in the forest that have at least one subtree isomorphic to the candidate, but we must find *all* instances of a pattern.

Overall, as explained in detail in the long version of this paper [14], the naive method is so inefficient as to be impractical. In a nutshell, we only want to find root-path subtrees, not all subtrees; we cannot afford to store all the values of a pattern to compute its entropy; and we cannot use simplifying properties such as upward closure, cutoff thresholds, sampling, or query selectivity estimates to speed up the process of computing NTCs.

We use three approaches to speed up preprocessing: keyword search characteristics, pruning of zero NTCs, and approximation.

Keyword Search Parameters. Previous studies of internet document retrieval have shown that the average query has roughly 2.5 keywords [16]. As users want specific answers, we expect the number of nodes in the user's ideal answer to be quite low. That is, if the query has many keywords (e.g., a long paper title), probably many of those words reside in just a few nodes of the top-ranked answer. Hence we introduce a domain-dependent upper bound MQL (maximum query length) on the value of n (pattern size) considered when precomputing NTCs. Setting MQL to 4 or 5 seems reasonable for bibliographic DBs.

Zero NTCs. We refer to the root of a candidate answer as the LCA of its pattern. If a pattern's LCA has only one instance in the DB (e.g., the DB root), its NTC is zero [14]. During preprocessing, we ignore these patterns, because knowing the value of one root-path pattern does not help to narrow down the value of another, when their only relationship is that they belong to the same DB. This property boosts preprocessing performance for deep XML data sets such as Xmark, as shown in Section 6.

Approximation. In the long version of this paper [14], we extended the definition of the concept of *interaction information* [5] to work with XML. Using this concept, we presented and analyzed multiple potential approaches to approximating NTCs [14]. Our final choice exploits the following formula:

$$H(t) = \sum_{1 \le i \le n} H(p_i) + \min\left(0, \sum_{T \subseteq \{p_1, \ldots, p_n\}} (-1)^{|T|} Intr(T)\right) \tag{4}$$

where $Intr(T)$ is the interaction information of pattern T. Using the above approach, we compute the exact NTC of patterns up to a certain size $EV \le MQL$. Then we approximate the NTC of larger patterns using formula (4). One should set EV based on the time available for preprocessing (e.g., half a day). Our experiments in Section 6 show that even low values of EV are very effective.

5. SEARCH & RANKING ALGORITHMS

The algorithm in Fig. 5 shows our method of computing NTCs. Our first challenge is to generate the patterns efficiently (lines 1-20), by generating only root-subtree patterns and generating every pattern only once. To help ensure that we generate patterns only once, we impose a somewhat counterintuitive transitive < relationship on patterns, where -1 is greater than all other labels:

Definition 6. For patterns t and t', we have $t < t'$ iff the prefix string of t' is a prefix of the prefix string of t, or the prefix string of t is lexicographically greater than the prefix string of t'. We have $t = t'$ if both patterns have the same prefix string.

Two subtrees of t are *siblings* if their roots are siblings in t.

Definition 7. A pattern t is **sorted** when for every pair of sibling proper subtrees u and u', $u \le u'$ iff the root of u appears before the root of u' in the prefix string for t.

For instance, pattern *bib paper author -1 title -1 -1 paper author -1 -1* is sorted, while pattern *bib paper author -1 -1 paper author -1 title -1 -1* is not. In the long version of this paper [14], we prove that every prefix of a sorted pattern is sorted. We generate only sorted patterns. The **last path** of tree t is the path that starts at the last node with only one child in the depth first traversal of t, and ends at the last node in the depth first traversal of t. For instance, *10 SIGMOD* is the last path in Fig. 1(a). The *last rightmost node* of a subtree is the last node in its last path. The **level** of a last path is the level of the parent of its root in t. Everything that is not on the last path of t is t's **prefix-tree**. All patterns with the same prefix-tree pattern form a **prefix class**. For example, all root-path patterns belong to the same prefix class, as their prefix-tree is empty. We can

```
Input: XML data file $data$
Input: Maximum size $EV$ of patterns to compute
       exact NTC for
Input: Maximum size $MQL$ of patterns to compute
       exact or estimated NTC for
Input: Minimum frequency thresholds $MIN\_FREQ$
Output: Table $CT$ of NTCs for $data$

   /* Create path indices for all frequent
      root-path patterns                              */
1  $indxs$ = Depth_First_Scan($data$, $MIN\_FREQ$);
   // Compute the entropy for root-path patterns
2  forall $p \in indxs$ do
3      $p$.entropy();
   /* Initialize the set of prefix classes           */
4  $pfxSet \leftarrow \emptyset$;
   /* Add all root-path patterns as one prefix
      class                                           */
5  $pfxSet$.add($indxs$);
6  for $k = 2$ to $MQL$ do
7      $nextPfxSet \leftarrow \emptyset$;
8      last = ();
9      forall $pfx \in pfxSet$ do
10         forall $p \in pfx$ do
               /* Compute all prefix classes whose
                  prefixes are $p$                    */
11             $nextPfx \leftarrow \emptyset$;
12             forall $q \in pfx$ do
13                 $Jnt \leftarrow$ join_Pattern($p, q$);
14                 forall $r \in Jnt$ do
15                     if $subTrees(r) \not\subset pfxSet$ then
16                         continue;
17                     if $k \leq EV$ then
                           /* Join indices and get
                              frequency as well as NTC
                              */
18                         $w \leftarrow$ join_Indices($p, q$);
19                         if $w.freq < MIN\_FREQ$ then
20                             continue;
21                         CT[$r$] $\leftarrow w.\hat{I}$;
22                     else
23                         CT[$r$] $\leftarrow$ approximate$\hat{I}(r)$;
24                     if $k \neq MQL$ then
25                         $nextPfx$.add($r$);
26             if $k \neq MQL$ then
27                 $nextPfxSet$.add($nextPfx$);
28     $pfxSet \leftarrow nextPfxSet$
29 return $CT$;
```

Figure 5: Algorithm to compute NTCs

describe a pattern t as $t = (px, pa, l)$, where px is its prefix-tree, pa is its last path, and l is the level of its last path. We use the following operation in line 13 of the algorithm in Fig. 5 to generate sorted patterns of size n from sorted patterns of size $n-1$.

Definition 8. Given patterns $t = (px, pa_t, l_t)$ and $r = (px, pa_r, l_r)$, we define their **join** $\oplus$ as follows:

- If $l_r < l_t$, then $t \oplus r = (t, pa_r, l_r)$.
- If $l_r = l_t$, then $t \oplus r$ is a set of patterns of the form $s = (t, pa_r, l_r)$. For each common prefix path c of pa_t and pa_r such that $c \neq pa_t$ and $c \neq pa_r$, we include pattern $s = (t, pa_s, l_s)$, where pa_s is created by removing the nodes of c from n_r. The level of pa_s is $l_r + |c|$.
- If $l_r > l_t$, then $t \oplus r$ is null.

For example, the patterns *bib paper author -1 -1* and *bib paper title -1 -1* have the same prefix-tree (null) and the same levels (0). Therefore their join contains *bib paper author -1 -1 paper title -1 -1* and *bib paper author -1 title -1 -1*. The two generated patterns belong to the same prefix class. The join of the first of them with the second is null, as the level of the first (0) is less than the level of the second (1). The result of joining the second pattern with the first one is *bib paper author -1 title -1 -1 paper title -1 -1*. The algorithm in Fig. 5 only needs to join patterns from the same prefix class (lines 12 and 13). The join of two sorted patterns is not generally sorted, so the call to join_Pattern in line 12 must check the result patterns to ensure that they are sorted. We call a pattern a **node-subtree** of pattern t at node v if it is induced by removing all nodes from t except for v and its descendants (if any). To see if the pattern is sorted, we must compare every node-subtree on the rightmost path of the pattern with its left sibling (if any). This function takes $O(m(d-1))$ time, where m is the number of nodes in the pattern and d is the length of its rightmost path. We ignore patterns at level 0 in our implementation, because their LCA is the tree root, so their NTC is zero. The join operation itself takes $O(m)$ time. Therefore, the exact time complexity of each join and check operation is $O(m(d-1))$. In the long version of this paper [14], we prove that the algorithm in Fig. 5 generates each pattern exactly once.

Since the process of generating patterns proceeds level by level, it can take advantage of the Apriori technique. That is, it can prune patterns of size m that have subtrees that do not have any instance in the DB or have very few instances in the DB (lines 15 and 19), before finding their instances in the DB. Patterns of very low frequency often indicate noise in the DB. Line 13 of the algorithm in Fig. 5 calls the join_Pattern algorithm to generate the sorted patterns. All the patterns found so far are stored in a hash table in main memory. The function *subTrees* in line 15 finds all subtrees of size $k-1$ for a pattern of size k. For patterns of size larger than EV, the algorithm in Fig. 5 uses the approximation technique discussed in the previous section: it finds all the subtrees of the pattern, then uses formula (4) to approximate the total correlation. This step takes $O(s^3)$ time, where s is the size of the pattern. The algorithm continues until it reaches the MQL size limit.

The second challenge in computing ranks is to efficiently find the instances of the generated patterns in the DB. Since the most important parts of patterns are root-path patterns, line 1 of the algorithm in Fig. 5 scans the DB in a depth first manner and stores the instances of each root-path pattern in a separate path index. As explained in the long version of this paper [14], we use a form of Dewey encoding to make this process efficient. Path indices can join at different levels. Since the path indices are sorted on their Dewey codes, we compute the instances of each pattern p by a zigzag join on the path indices of p's root-path patterns (line 18). Thus,

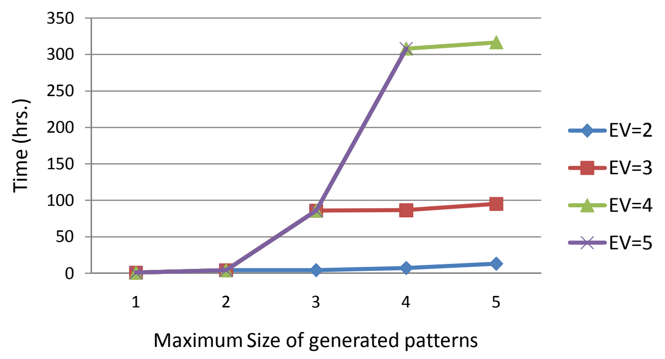

Figure 6: Preprocessing performance for IMDB

EV	DBLP	IMDB
2	0.290	0.250
3	0.126	0.106
4	0.061	0.052

Table 1: Approximation error for DBLP and IMDB

Data set	Coherency	XSearch	XReal	PN	XRank
IMDB	0.715	0.642	0.612	0.511	0.421
DBLP	0.834	0.794	0.790	0.621	0.591

Table 2: MAP for DBLP and IMDB queries

we do not need to scan the DB to find the pattern instances. Also, since we join the path indices in a certain order, we generate each result at most once. When joining the root-path pattern indices in line 18, the *JoinIndices* routine inserts the values of the new pattern into a temporary table. Then, the algorithm computes the exact NTC for the new pattern at line 21, based on the values stored in the temporary table. If the NTC will be approximated for larger patterns, then line 21 must also compute and store the entropy of this pattern. If a pattern has EV or more root-path patterns, the algorithm approximates the value of NTC at line 23.

The NTC value for each pattern resides in a hash table in main memory during query processing. The query processing system finds each candidate answer, generates its pattern, looks up the NTC for that pattern, and then ranks the answer accordingly. The problem of finding all candidate answers given a keyword query has been addressed in previous work. The SA algorithm [7] returns the LCA, leaf nodes, and leaf node parents of each candidate answer. We modified SA to also produce the pattern of a candidate answer, without affecting the asymptotic time complexity of SA. Since the NTC of the candidate answers whose LCA is the root of the tree is zero, we changed SA to omit the DB root as a candidate LCA. This optimization helped the modified SA perform better than the original SA.

6. EVALUATION

We implemented and evaluated the preprocessing algorithm and query processing system on a Linux machine with 4 GB memory and an Intel Pentium D 2.80 GHz. Our experiments use two real-world and one synthetic data sets: DBLP (207.2 MB, max depth 5), IMDB (799 MB, max depth 7), and Xmark (1172.3 MB, max depth 11). The

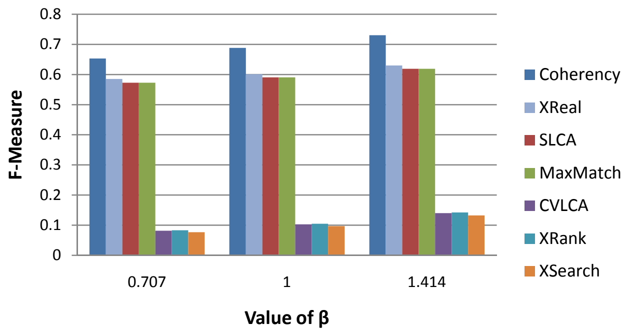

Figure 7: Average F-measure for IMDB queries

IMDB data set has information about movies and TV shows from www.imdb.com. No previous approach has dealt with data sets as large as IMDB, or as large and deep as Xmark. Except where otherwise noted, the results for DBLP and IMDB were very similar. Thus due to space considerations, the DBLP results are included in the long version of this paper [14], but largely omitted here.

6.1 Preprocessing Performance

We set the minimum frequency threshold to 50 for all data sets; any value between 2 and 50 would have produced the same results. IMDB and Xmark have no patterns with frequencies between 1 and 49, and DBLP has only one such pattern. We set MQL to 5 for all data sets, which is a reasonable threshold for all domains. Fig. 6 shows the time for computing exact and approximate NTCs for IMDB, broken down by the value for EV. This includes the time to parse the XML data and create path indices. The time for exact computation increases exponentially as EV increases. Though the shape of their curves is very similar, the exact computation takes much longer for IMDB than for DBLP, because IMDB is larger and its structure is more nested than DBLP's, so IMDB has many more patterns and pattern instances. Exact preprocessing for Xmark data sets with $MQL = 5$ took too long to run to completion. Approximate preprocessing for Xmark with $EV = 3$ and $MQL = 5$ took 128 hours.

Fig. 1 shows the effectiveness of approximation. The figure excludes Xmark, as its content is generated randomly and is not meaningful. We sorted the results returned by exact and approximate computation, based on their NTC values. Then we computed the approximation's error rate, using Kendall's tau distance between two permutations of the same list of NTCs [9]. This measure penalizes the sorted approximate-computation list whenever a pattern occurs in a different position in the exact-computation list. In the figure, Kendall's tau is normalized by dividing by its maximum value, $n(n-1)/2$, where n is the total number of patterns in the list of NTCs. As shown in Fig. 1, the error rate decreases as EV increases. $EV = 3$ gives an error rate close to 10%. Some patterns with different positions in the approximated lists for both data sets have zero frequency. As these patterns will never be used at query time, the actual error rate is less than the reported values. In many domains, the preprocessing will only be done once; in domains with more fluid structure, preprocessing will still be a rare event, e.g., once a year. Our preprocessing times are quite reasonable for this rare task. For $MQL = 5$, the resulting table of NTCs occupied under 2 MB for IMDB and less for DBLP.

Thus CR imposes only a modest memory overhead at query processing time.

6.2 Query Processing

There is no standard benchmark to evaluate the effectiveness of keyword search methods for data-oriented XML. To address this problem, we used a TREC-like evaluation method [13]. We asked 15 users who were not conducting this research to provide up to 5 keyword queries for IMDB and DBLP. This resulted in 40 queries for IMDB and 25 for DBLP, listed in the long version of this paper [14]. We developed a keyword query processing prototype based on the baseline algorithm that returns every LCA for each query, without any ranking. Our users submitted their queries to this system and judged the relevancy of each answer. The prototype's user interface merged all the equal answers (subtrees with the same labels and contents), and did not present the user with any subtrees rooted at the root of the DB. We use two evaluation tasks to examine the effectiveness of CR. The first task compares CR with the methods: SLCA [18], XRank [6], CVLCA [10], MaxMatch [12], XReal [1], and the best-performing version of XSearch [4] for our data sets to evaluate unranked retrieval sets. We measured the recall, precision, F-measure, and fall-out [13]. Our query system uses the NTCs computed using $MQL = 5$, $EV = 3$, and minimum frequency 50. The use of larger values for EV did not affect the results, which is consistent with our error rate analysis for EV. None of the queries matched more than 4 leaf nodes, which shows that our choice of $MQL = 5$ was reasonable.

Due to space limits, we only show Fig. 7, which compares the F-measure of the aforementioned methods for IMDB. The long version of this paper [14] gives precision and recall graphs for both data sets, the F-measure graph for DBLP, and a more detailed discussion of the experimental results. The precision of CR is as high or higher than every other method, for every query over IMDB and DBLP. CVLCA, XSearch, and XRank show the lowest precision overall. They return many irrelevant subtrees whose LCA is the root of the DB. For instance, for *VLDB Korea*, these methods return subtrees describing a paper published in VLDB and a conference (not VLDB) held in Korea. SLCA, MaxMatch, and XReal show better precision than CVLCA, XSearch, and XRank. SLCA and MaxMatch show almost identical precision and recall. For the query *Fight Club*, only movies with *Fight Club* in the title were marked relevant by the users, but there are many candidate answers having these words in their keywords or production company names. Subtrees containing a title, keyword, or production company all have the same size: one non-leaf node. The same issue arises for queries *Beautiful Mind, Pearl Harbor, Momento, The Watchmen*, and *Return of the Mummy*. No method had good precision for these queries. For the DBLP query *Han Data Mining*, the user wanted the book written by Han, but got data mining papers by Han as well. Later we show how CR solves these problems through ranking. For very specific queries, like *Comedy Woody Allen Scarlett Johansson*, the candidate answers are the desired ones, and SLCA, MaxMatch, XReal, and CR are equally precise. CR shows perfect recall for all queries. SLCA and MaxMatch filter out relevant large subtrees, resulting in very low recall. For instance, in some movies *Edward Norton* acted and also appeared on the sound track. SLCA and MaxMatch return only the sound track nodes. XReal filters out the entity types that have relatively low query term frequencies. For some queries, users marked subtrees containing other relevant subtrees as relevant. For instance, all methods except CR return a director subtree or composer subtree for *Satyajit Ray*. However, our users also marked as relevant the subtrees containing *both* director and composer information, as they were interested in movies that Satyajit Ray directed and composed. XSearch, CVLCA, and XRank show better recall than XReal, SLCA, and MaxMatch, for the reasons explained in Section 2. Fig. 7 shows that CR delivers better precision and recall than other methods for IMDB. The same is true for the DBLP data set.

Consider a DBLP query with no relevant answers: *SASI Protocol Attack*. CR returned no answer, while XSearch and CVLCA return 121 answers, SLCA, MaxMatch, and XRank return 460, and XReal return 82 answers. The DL filtering technique of XSearch and CVLCA and the filtering approach of XReal improved their performances, but they still gave many unhelpful answers. Thus, CR provides better fall-out than the other methods.

In the second task, we evaluated the ranking effectiveness of CR, XSearch, XReal, and XRank using *mean average precision* (MAP), which shows how many of the relevant answers appear near the beginning of the returned list of answers. By combining CR with IR-style ranking methods, we can take advantage of both structural and content information in the database. We combined CR with a version of the pivoted normalization method [13] customized for XML, to determine the rank $r(t)$ of answer t:

$$r(t) = \alpha \hat{I}(t) + (1 - \alpha) ir(t), \tag{5}$$

where $ir(t)$ is:

$$ir(t) = \sum_{w \in Q, E_l} \frac{1 + \ln(1 + \ln(tf(w)))}{(1 - s) + s(el_l/avel_l)} * qtf(w) * \ln(\frac{N_l + 1}{ef_l}). \tag{6}$$

Here, E_l is an element whose label is l and is the parent of the leaf node m that matches term w from the input query Q. $tf(w)$ and $qtf(w)$ are the number of occurrences of w in m and Q, respectively. el_l is the length of m, and $avel_l$ is the average length of the contents of all leaf nodes m whose parent has label l. N is the count of all non-leaf elements, and ef_l is the number of nodes whose label is l. s is a constant; the IR community has found that 0.2 is the best value for s [13]. α is a constant that controls the relative weight of structural and contextual information in ranking. If α is set to 1, the formula uses only structural information. We used the parameter settings given in the previous section to compute $\hat{I}$. For our queries, CR with pivoted normalization has better MAP than plain CR. The MAP for both data sets changes only slightly (at most .01%) for α between 0.1 and 0.9. It drops by more than 25% if we set α to 0. The reason is that the DBLP and IMDB are data-oriented, so fields are relatively short and words rarely repeat within a single field. Thus, structural information is more important than contextual information. Table 2 shows that the MAP of CR is considerably higher than that of other methods for the Wilcoxon signed rank test at a confidence level of at least 95%. This confirms that NTC is better than subtree size as a basis for ranking decisions. Also, XSearch's filtering strategy lowers its ranking quality. As mentioned in Section 2.1, XReal cannot recognize the important entities in DBLP. For a deep data set like IMDB, XReal highly ranks relatively large subtrees whose relationships are not strong enough. XSearch has almost the same MAP as XReal. Pivoted normalization without CR ($\alpha = 0$), shown as PN, has lower MAP than all but one other method, because PN uses only contextual information. XRank has the lowest MAP, as its filtering strategy lowers its precision. Also, PageRank-based approaches are not appropriate for all types of relationships. For instance, a paper with many authors is not necessarily an important paper. In the remainder of this section, we explain how CR ranks several query answers that had low precision in the first evaluation task. The long version of this paper [14] extends the explanation to cover all the low-precision queries from the first task, and also discusses every query where CR did not rank the desired answer(s) first.

Since a book has fewer authors than a paper in a conference or journal, the NTC of the author and title of a book is higher than the NTC of the author and title of a paper. Therefore, for the query *Han Data Mining*, CR returns the textbook at the first result, as the user desired. XReal filters the book entity and XSearch ranks the desired answer very low.

We consider an example of how CR determines users' intentions when the query terms are ambiguous. For query *CCS 2007* on DBLP, our users wanted information about the 2007 ACM Conference on Computer and Communications Security (CCS). However, many 2007 papers have *CCS* (change control system) in their titles. CR ranked the title of the proceedings of CCS 2007 first, as both words occurred in the title. XSearch ranked the irrelevant papers higher and XReal filtered out the proceedings node, which had very low query term frequencies. For the query *Maude*, CR ranked papers whose titles mention Maude above papers written by authors named *Maude*. XSearch ranked the latter first.

Subtrees containing two *keyword* nodes, and subtrees containing a *keyword* node and a literature reference, have lower NTCs than subtrees containing a *title* and a *keyword* node. So for query *Godfather Crime*, CR ranks crime movies whose title contains *Godfather* above movies with *Godfather* and *Crime* as keywords and/or literature references. XSearch ranks the subtree with two keywords first, as it is smaller. XReal ranks the subtree containing keyword and literature reference nodes higher, as the frequencies of the query terms in these nodes are higher than their frequencies in the title node. Queries *Beautiful Mind, Pearl Harbor, Momento, The Watchmen*, and *Return of the Mummy* have the same problems.

Although CR provided a ranking close to users' expectation, it did not deliver exactly the ranking that the user wanted for some queries. For DBLP query *Authenticated Dictionary*, the user wanted titles where these two words occur next to each other. Also, our IMDB users preferred the most recent movies. We leave these issues for future work.

7. CONCLUSIONS

We have proposed *coherency ranking*, a new ranking method for XML keyword search that ranks candidate answers based on statistical measures of their cohesiveness. Coherency ranks are computed based on a one-time preprocessing phase that exploits the structure of the data set. For each type of subtree in the data set, the preprocessing phase computes its *normalized total correlation* (NTC), a new statistical measure of the tightness of its relationship. NTC is invariant under equivalence-preserving schema transformations, thus avoiding overreliance on shallow structural details. As it is too expensive to compute NTCs for all subtrees of an XML data set, we developed approximation and optimization methods to make preprocessing affordable. For query answering, we ranked candidate answers based on the precomputed NTCs. For queries over two real-world data sets, coherency ranking showed considerable improvements in precision, mean average precision and recall, compared to six previously proposed and one customized IR based methods.

8. REFERENCES

[1] Z. Bao et al. Effective XML Keyword Search with Relevance Oriented Ranking. In *ICDE 2009*.

[2] G. Bhalotoa et al. Keyword Searching and Browsing in databases using BANKS. In *ICDE 2002*.

[3] S. Chakrabarti, K. Puniyani, and S. Das. Optimizing Scoring Functions and Indexes for Proximity Search in Type-annotated Corpora. In *WWW 2007*.

[4] S. Cohen et al. XSearch: A Semantic Search Engine for XML. In *VLDB 2003*.

[5] T. Cover and J. Thomas. *Elements of Information Theory*. Wiley, 1983.

[6] L. Guo et al. XRANK: Ranked Keyword Search over XML Documents. In *SIGMOD 2003*.

[7] V. Hristidis et al. Keyword Proximity Search in XML Trees. *TKDE*, 18(5):525–536, 2006.

[8] Y. Ke, J. Cheng, and W. Ng. Mining Quantitative Correlated Patterns Using an Information-Theoretic Approach. In *SIGKDD 2006*.

[9] M. Kendall and J. D. Gibbons. *Rank Correlation Methods*. 1990.

[10] G. Li et al. Effective Keyword Search for Valuable LCAs over XML Documents. In *CIKM 2007*.

[11] Y. Li, C. Yu, and H. V. Jagadish. Schema-Free XQuery. In *VLDB 2004*.

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

[13] C. Manning, P. Raghavan, and H. Schütze. *An Introduction to Information Retrieval*. 2008.

[14] A. Termehchy and M. Winslett. Effective Ranking of XML Keyword Search Results (Extended Version). Technical Report UIUCDCS-R-2009-3043, 2009.

[15] S. Watanabe. Information Theoretical Analysis of Multivariate Correlation. *IBM Journal of Research and Development*, 4(1):66.

[16] J. Wen, J. Nie, and H. Zhang. Clustering User Queries of a Search Engine. In *WWW 2001*.

[17] I. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. 2005.

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

[19] X. Yin, J. Han, and J. Yang. Searching for Related Objects in Relational Databases. In *SSDBM 2005*.

[20] M. Zaki. Efficiently Mining Frequent Trees in a Forest. *TKDE*, 17(8):1021–1035, 2005.