Using Structural Information in XML Keyword Search Effectively

ARASH TERMEHCHY
University of Illinois at Urbana-Champaign

and
MARIANNE WINSLETT
University of Illinois at Urbana-Champaign

The popularity of XML has exacerbated the need for an easy-to-use, high precision query interface for XML data. When traditional document-oriented keyword search techniques do not suffice, natural language interfaces and keyword search techniques that take advantage of XML structure make it very easy for ordinary users to query XML databases. Unfortunately, current approaches to processing these queries rely heavily on heuristics that are intuitively appealing but ultimately ad hoc. These approaches often retrieve false positive answers, overlook correct 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 concepts of data dependencies and mutual information. With coherency ranking, the results of a keyword query are invariant under a class of equivalency-preserving schema reorganizations. We analyze the way in which previous approaches to XML keyword search approximate coherency ranking, and present efficient algorithms to process queries and rank their answers using coherency ranking. Our empirical evaluation with two real-world XML data sets shows that coherency ranking has better precision and recall and provides better ranking than all previous approaches.

Categories and Subject Descriptors: D.2.7 [**Database Management**]: Systems—*query processing*

General Terms: Algorithms, Performance

Additional Key Words and Phrases: XML, Keyword Queries, Correlation Mining

1. INTRODUCTION

Huge volumes of XML data are available in domains ranging from science [Stein et al. 2002] to business [OASIS 2006] to text databases [Denoyer and Gallinari 2006].

This is a preliminary release of an article accepted by ACM Transactions on Database Systems. The definitive version is currently in production at ACM and, when released, will supersede this version.
Author's address: 2119 Siebel Center for Computer Science, 201 N. Goodwin Ave., Urbana, IL 61801.

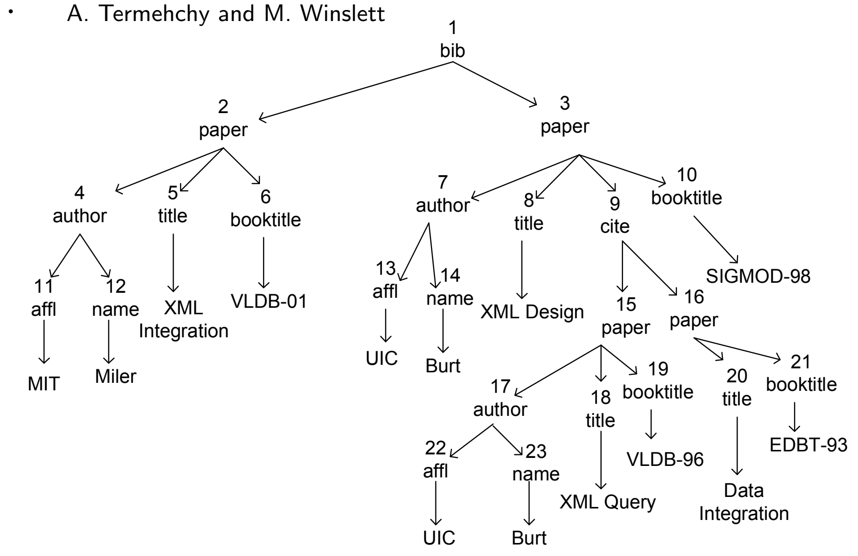


Fig. 1. DBLP database fragment, with DBLP's native XML schema

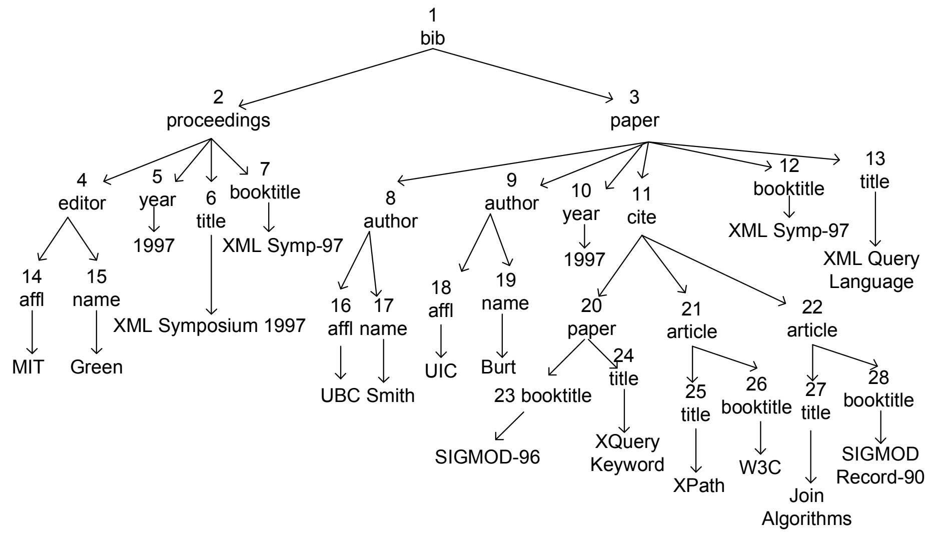


Fig. 2. DBLP database fragment, with DBLP's native XML schema

As most users in these domains are not familiar with concepts such as schemas and query languages, they need an easy query interface. Visual interfaces are excellent for many situations [Win et al. 2003], but must be customized individually for each domain, which is very expensive. Keyword search [Cohen et al. 2003; Guo et al. 2003; Hristidis et al. 2006; Hristidis et al. 2003; Kimelfeld and Sagiv 2005; Li et al. 2007; Schmidt et al. 2001; Sun et al. 2007; Trotman and Geva 2006; Xu and Papakonstantinou 2005; 2008] and natural language interfaces [Li et al. 2006] have been proposed as less costly options; 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 [Robertson 1977].

In this paper, we propose a ranking approach for keyword queries that exploits XML structure while avoiding overreliance on shallow structural details. This new ranking approach has higher precision and recall and better ranking quality than previous approaches. We make the following 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 the 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. In particular, we show that 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 first name.
- We show how to deploy CR in XML query processing, 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 the substructures in the DB that contain the query keywords. Precomputation 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 precomputation phase are not affected by non-structural updates to the content of a populated DB, so the precomputation phase rarely or never needs to be repeated as the DB content evolves.
- Since naive methods are prohibitively inefficient for the precomputation step, we present and evaluate optimization and approximation techniques to reduce precomputation costs. Our experiments show that these optimizations improve precomputation performance by orders of magnitude while introducing negligible approximation errors.
- 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

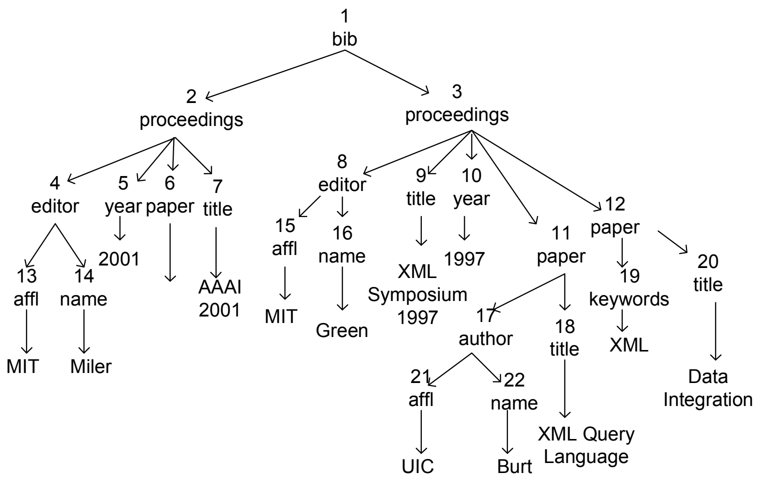


Fig. 3. Reorganized DBLP database

(TF) and inverse document frequency (IDF) [Jones 1972] or PageRank [Brin and Page 1998] can be helpful in ranking query answers, CR can be combined with such measures to improve the precision of query answers. We provide an empirical study of how to combine information-retrieval-style methods and 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 precomputation algorithm could also be used for purposes other than keyword search, such as finding highly correlated elements and patterns [Ke et al. 2006], or discovering approximate functional dependencies [Ilyas et al. 2004] in XML databases.

This paper is organized as follows. Section 2 discusses problems with current XML keyword query processing. Section 3 introduces the basic principles of CR and Section 4 provides the mathematical tools to quantify CR scores. Section 5 explores and formalizes the design independence properties of CR. Section 6 presents optimization and approximation techniques for DB precomputation. Section 7 describes our implementation, Section 8 presents empirical results, and Section 9 concludes the paper.

2. MOTIVATION AND RELATED WORK

In this section, we analyze current approaches to XML keyword search, in terms of their precision, recall, and ranking capabilities.

2.1 Basics

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 has path */bib/paper/title*. Each *subtree* $S = (r_s, V_s, E_s, L_s, C_s, D_s)$ of T is a tree such that $r_s \in V_s$, $V_s \subseteq V$, $E_s \subseteq E$, $L_s \subseteq L$, $C_s \subseteq C$, and $D_s \subseteq D$.

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, although our experiments do not use these.) 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 [Manning et al. 2008], 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.

2.2 Current Approaches

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 [Agrawal et al. 2002; Balmin et al. 2004; Bhalotoa et al. 2002; Cohen et al. 2003; Guo et al. 2003; Hristidis et al. 2006; Hristidis et al. 2003; Kimelfeld and Sagiv 2005; Kourtika et al. 2006; Li et al. 2007; Li et al. 2004; Liu et al. 2006; Schmidt et al. 2001; Xu and Papakonstantinou 2005]. 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 ([Schmidt et al. 2001], with modest subsequent refinements [Guo et al. 2003; Hristidis et al. 2006; Xu and Papakonstantinou 2008]). For instance, consider the DBLP fragment from `www.informatik.uni-trier.de/` shown in Fig. 1. 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, 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 either not return these answers, or rank them below the 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 [Sun et al. 2007; Xu and Papakonstantinou 2005]; 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, 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 [Liu and Chen 2008] improves the precision of SLCA, by eliminating an SLCA answer if it has the same root as another SLCA answer A, and its matched keywords are a subset of A's matched keywords. But it still has SLCA's effectiveness problems: it still returns node 9 for Q_2, and does not return nodes 3 for query Q_3. It does not rank its answers either.

XSearch [Cohen et al. 2003] 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. 2 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. 2. 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 Q_6 = *Smith Burt* in Fig. 2, as it filters out node 3. XSearch's distance-based ranking scheme does not help to avoid DL's pitfalls.

We also review the MLCA approach [Li et al. 2004] used to retrieve meaningful relationships from XML documents with a natural language query interface [Li et al. 2006]. Similar to DL, MLCA assumes that only the closest nodes of different types are meaningfully related. For instance, node 16 in Fig. 2 will not be associated with node 15 because node 16 *has a different node with the same label (name) closer to it*. Therefore, the MLCA approach successfully filters out the undesirable answers for queries like Q_5.

However, MLCA has many of the DL and SLCA problems. MLCA does not identify uninteresting relationships between nodes of the same type; e.g., it returns undesirable node 1 for query Q_7 = *Smith Green*. Second, it can return undesirable answers when a child is null due to being optional. For example, consider Fig. 1, but with node 10 null. For query Q_8 = *XML Design VLDB*, MLCA returns the undesirable tree rooted at node 1 and including nodes 8 and 6 in Fig. 1. Third, MLCA is not appropriate to use with databases that contain instances of more than one entity type. Consider Fig. 2 with *proceedings*'s *title* renamed to *venue*. For the query *editor XML Query*, MLCA returns the subtree rooted at the root of the database containing nodes 4 and 13. Since there is no meaningful relationship between nodes 13 and 4, MLCA should not return this answer. These precision problems are important, as MLCA does not rank its answers. Fourth, MLCA misses desirable answers when similar nodes have a meaningful relationship. In Fig. 3, MLCA finds no interesting relationship between nodes 9 and 17, as there is another node with the label *title* closer to node 17. To improve precision, CVLCA combined MLCA and DL into one approach [Li et al. 2007]. But the combination still has the precision problems described above, and lower recall than either approach alone. [Li et al. 2007] also introduced another approach, called VLCA. VLCA uses the DL heuristic; therefore, VLCA has the same problems as the XSearch method. CVLCA and VLCA do not rank their returned answers.

XRank [Guo et al. 2003] 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. For instance, all the *Proceedings* tags in Fig. 1 have the same PageRank.

XReal [Bao et al. 2009] 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 Q_{10} = SIGMOD 1997, XReal ranks the subtree rooted at node 3 in Fig. 2 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. In general, IR ranking techniques do not take advantage of XML's structural properties effectively.

Another method is to determine the interesting entity types manually and return only the instances of these entities that contain the keyword query. For instance, assume that DBLP contains only entity types *paper* and *proceedings*. Then, this method returns only the subtrees rooted at *paper* and *proceedings* nodes that contain the input keywords. However, we still have to rank the candidate answers, as usually more than one instance of each entity type match the keyword query. For example, both papers rooted at node 2 and 3 are candidate answers for the query *XML VLDB* in Fig. 1. Nevertheless, the desired answer is the paper rooted at node 2. Furthermore, this method is domain dependent and appropriate only for the databases that consist of relatively small number of entity types.

The first key shortcoming of all these methods is that they *filter out answers instead of ranking them* and they *rely on very shallow structural properties to rank answers*. As relevance is a matter of degree, good ranking schemes are generally more effective than filtering [Robertson 1977]. 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.

In previous work, we developed an XML keyword search system that addresses many of these issues [Termehchy and Winslett 2009]. However, the precomputation

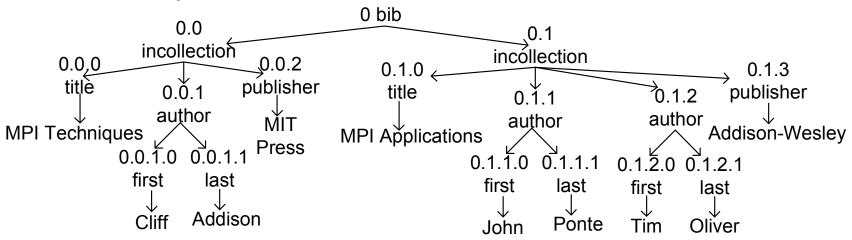


Fig. 4. DBLP database fragment

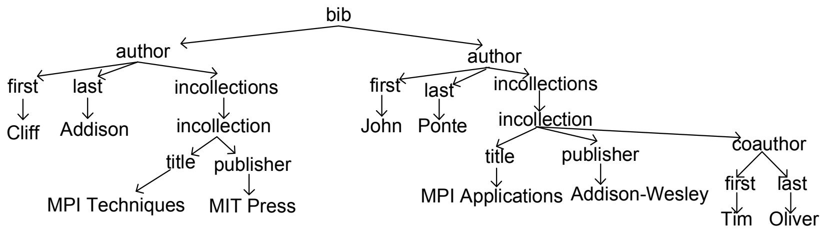


Fig. 5. Redesigned DBLP database fragment

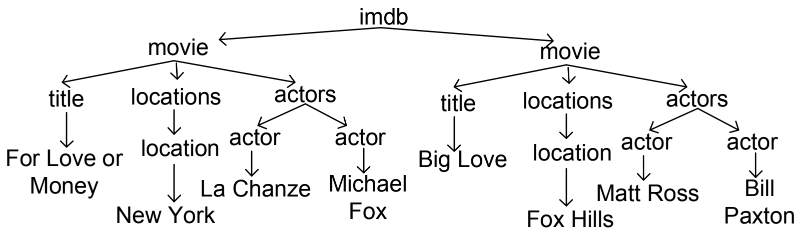


Fig. 6. IMDB database fragment

algorithm provided in that work does not scale well to very large data sets. For instance, our experiments showed that that approach takes between one week and over three weeks to preprocess a data set of size 800MB, depending on the content of the data set. That previous work also does not provide rigorous proofs for the correctness of its algorithms and optimization techniques, which we do provide here. In addition, in this paper we explore the effect of equivalence-preserving schema transformations on query answer ranking, and the relationship between CR and approximate functional dependencies. We define the concepts of entropy and mutual information for the XML data model, and explore their properties. We also provide more details about the precomputation algorithm and query time algorithms. For example, this paper analyzes the flaws of using current data mining techniques for the precomputation stage. We also provide more detailed analysis of our user study. Finally, this paper contains more experimental results and compares the effectiveness of CR with the current approaches more thoroughly.

3. COHERENCY-BASED RANKING

In this section, we propose an approach that uses deep structural information to rank the candidate answers instead of filtering them. First, we analyze previously proposed structural properties that could be used for this purpose in XML, relational, and graph databases and show they deliver poor ranking. We argue that the meaningfulness and interestingness of a candidate answer depends on the types of its nodes and their relationships. We show that the interestingness of the relationship between two entity types can be measured by the information content and correlation between its nodes. We also show that our proposed approach does not have the problems of the previously proposed approaches and delivers better ranking. Furthermore, we argue that as this approach takes advantage of deeper structural information, it provides almost the same answers for different designs of the same database.

Current XML keyword search systems implicitly assume that all relationships between nodes in a candidate answer are equally relevant to the input queries [Cohen et al. 2003; Bao et al. 2009; Guo et al. 2003]. The *distance* between nodes n and m is the number of nodes in the path between n and m. Thus, they rank higher the subtrees where the nodes containing query terms are less distant from each other. This heuristic is too ad hoc to work well in general: consider the subtrees rooted at *cite* and *bib* in Fig. 1, which contain paper titles. They have the same distance, but in a large bibliographic database with papers on many topics, such as the original DBLP data set, the relationship between the *title*s under *cite* is more meaningful than between the *title*s under *bib*. Even in a small bibliographic database where all the *paper*s under *bib* are on the same narrow topic, this rule will still hold for citations of papers that are themselves *title*s under *bib*, though it may be violated for citations of external papers. Our work is targeted toward large diverse data sets, where ranking keyword query answers is especially important and challenging. Also, our experiments show that many candidate answers have the same maximum distance. This 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 [Denoyer and Gallinari 2006; Chakrabarti et al. 2007]. Most candidate subtrees have the same distance in flat XML DBs, so we need to be able to rank subtrees with the same distance.

One may propose that the subtrees that occur deeper in the database (i.e. the root of the subtree is further from the DB root) present more meaningful relationships. However, this method has the same problems as the distance-based ranking. For instance, node 2 is at a higher level than node 9 in Fig. 1, but node 2's subtree represents a more meaningful relationship (information about one paper) than node 9's (information about different papers cited by the same paper).

A similar line of research has been followed for keyword queries in relational DBs [Bhalotoa et al. 2002; Agrawal et al. 2002; Balmin et al. 2004; Liu et al. 2006; Kourtika et al. 2006]. These approaches create a join tree whose tuples contain the query terms, and rank the tree based on its maximum distance. The ranking can also consider DBA-specified weights on the edges in the join tree [Balmin et al. 2004; Kourtika et al. 2006]; however, we need an automated approach.

Researchers have found that different types of relationships have different degrees of importance to the users in relational and graph DB keyword search systems [Bhalotoa et al. 2002; Yin et al. 2005; Golenberg et al. 2008]. They suggest that the fewer instances of type B associated with each instance of type A, the stronger their relationship is. One approach ranks a tuple higher if many tuples link to it, but it links to just a few [Bhalotoa et al. 2002]; the same idea can be used for graph databases [Golenberg et al. 2008]. Another employs the notion of join selectivity to measure the degree of closure among a set of tables [Yin et al. 2005]. We call this group of methods *association based approaches*. We can also extend these approaches to work with the XML data model. For instance, this approach can recognize that the relationship between two *title*s under *bib* in a large bibliographic database covering multiple topics (such as the original DBLP database) is more meaningful than the relationships between *title*s under *cite*. This is because each *title* under *bib* is associated with many more other *title*s than the number of *title*s associated with a *title* under *cite*. This heuristic improves the ranking quality of distance based approaches; however, as explained in the following paragraphs, these heuristics are misleading.

First consider Fig. 6 which illustrates some information about movies and TV shows from `www.imdb.com`. Node type *location* shows the filming locations of a movie. The number of actors in each movie greatly exceeds the number of its filming locations on average, considering all instances of the original IMDB data set. The fragment shown in Fig. 6 has twice as many number of actors as filming locations per movie which is typical of IMDB data set. Hence, according to the association based heuristics, the relationship between *title* and *location* is more interesting to users than the relationship between *title* and *actor*. Therefore, for query *Love Fox* these methods rank the *movie* subtree on the right higher than the *movie* subtree on the left in Fig. 6. However, we argue that the relationship between the title of a movie and its actor(s) is more interesting to most users than the relationship between its title and its filming location(s). The main search query form on the IMDB home page of IMDB web site (`www.imdb.com`) enables users to search on several attributes of a movie, such as titles and actors. However, it does not query the database based on the filming locations of a movie. There is a query form in the bottom of another page of the web site (`www.imdb.com/search`) that enables users to find movies based on their filming locations. Even in that form, the filming location has been placed after other attributes. It is reasonable to assume that the developers of the IMDB web site put the more important and more frequently queried attributes on the main query form of the database. Thus, the relationship between the title of a movie and its actor(s) is more important to most users of IMDB than the relationship between the title of a movie and its filming location(s). Of course, in some cases that users may be interested in the filming locations of a movie, but they are in the minority and our goal is to deliver the best possible ranking for the majority of users. Interestingly, distance-based approaches will give the same rank to both candidate subtrees for query *Love Fox*.

As another example, consider the fragments of DBLP shown in Fig. 4. If we compare the relationship of *title* with *author* and *publisher* in Fig. 4, we see that there are more authors associated with a title than there are publishers for the title on average. The same argument holds for the original DBLP data set, as there is more than one author for each paper but only one publisher per *incollection* (i.e. an article in a book) on average. Hence, association based approaches assume that the relationship between *title* and *publisher* is more important than the relationship between *title* and *author*. Thus, they rank the right *incollection* subtree higher than the left one for query Q_{11} = *MPI Addison*. However, we argue that the relationship between *title* and *author* of an *incollection* is more interesting than the relationship between its *title* and its *publisher* to most users. Developers of graphical user interfaces place the most queried database fields in the query forms. They make these decisions based on user feedback, query logs, and their own experiences. The DBLP data set has some query forms that perform search over specific fields, where users choose their field(s) of interest before submitting their queries at `dblp.uni-trier.de`, `dblp.mpi-inf.mpg.de/dblp`, and `dblp.l3s.de`. All these forms support searches based on author names, but none of them support searches via the publisher fields. They also show *author* as a facet when returning the candidate answer, so that users are able to filter the results based on specific author(s). However, they do not show the *publisher* field as a facet. This indicates that the *publisher* field is less interesting to usual DBLP users than *author*. In other words, the relationship of a book or article's *title* to its *author* is generally more relevant for query answering than the relationship between the book or article's *title* and its *publisher*. Of course, there are exceptions to this rule (e.g., an author looking for a potential publisher for her book), but it holds in general for large bibliographic databases. Note also that distance-based approaches will give the same rank to the candidate subtrees for query Q_{11} as both candidate answers have the same maximum distance. Consider the redesign of Fig. 4, shown in Fig. 5. The new design categorizes articles and books based on their authors, which makes the title closer to the publisher than it is to the article's author – even though the new design is still normalized [Arenas and Libkin 2004; Yu and Jagadish 2006]. Thus, distance-based ranking methods rank the candidate subtree in the right higher than the candidate subtree on the left for query Q_{11} in Fig. 5.

A second problem with association based methods is that 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. We can use this idea to measure the strength of the relationships in a candidate answer and rank the candidate answers according to these strengths. We will show how to quantify these relationships in Section 4. We will measure the correlation between node types, considering all instances of the node types in the complete underlying database. Using this information, we rank the most highly correlated candidate answers first. We call this approach **coherency ranking** (CR).

CR correctly ranks the subtree on the left higher than the subtree on the right in Fig. 4 for Q_{11} = *MPI Addison*, as the number of articles and books a publisher publishes is far more than the number of the publications of an author (considering all the instances in the original DBLP database). Thus, knowing the name of an author predicts the title of its publications more precisely than knowing the publisher of an article or a book. CR also gives more importance to the relationship between *title* and *actor* than the relationship between *title* and *location* in Fig. 6. CR handles the cases where association based methods deliver acceptable ranking. For instance, according to CR the relationship between *title*s under *cite* is more meaningful than the relationship between the *title*s under *bib* in the original DBLP database whose fragments shown in Fig. 2.

CR gives the intuitively desirable ranking for all the queries considered in Section 2. The *title* of a *paper* is associated with only one *booktitle* and there are on average fewer than 50 papers in each proceedings in the original DBLP database. However, each citation is co-cited with with around 40 other citations on average in the original DBLP database. Thus, *title* and *booktitle* are more correlated than two citations in the same paper. Thus, CR ranks the candidate answer rooted at node 2 first and the candidate answer rooted at node 9 second for Q_2 = *Integration VLDB* in Fig. 1. It also ranks the candidate answer rooted at node 1 last. As we will discuss in Section 4, users can set the minimum acceptable amount of correlation for the returned candidate answers. If this value is set to be greater than zero, CR will not return any spurious candidate answers such as the candidate answers rooted at the root of the DB. CR delivers a similar ranking for Q_4 = *SIGMOD XPath* and ranks a paper about "XPath" in "SIGMOD" higher than the subtree rooted at node 11 in Fig. 2. CR also ranks a paper published by an *author* named "Green" and affiliated with "UBC" higher than the candidate answer rooted at the root of the DB in Fig. 2 for Q_5 = *UBC Green.* This is because each institute has a limited number of authors and each author is affiliated with a few institutes, but all authors and institutes in the original database are associated with each other. The correlation between each pair of author and editor in the original DBLP database is very low, as almost every author in the database can be associated with each editor and vice versa. Thus, CR recognizes that the candidate answer rooted at node 1 shown in Fig. 2 is not a meaningful answer to Q_7 = *Smith Green* and will not return it or will return it as the last candidate answer in the ranked list of answers. Even if node 10 in Fig. 1 is null, CR still omits or ranks last the subtree rooted at node 1 for Q_8 = *XML Design VLDB.* This is because considering all information in the original DBLP database, the correlation of *title* of one *paper* and *booktitle* of another *paper* whose only relationship is that they are placed in the same database is very low. CR will deliver a similar result for Q_9 =*editor XML Query* in Fig. 2. Each *booktitle* is associated with only one *year* value, as each conference DBLP database is held once per year. Each *year* value is associated with many *booktitle* values in DBLP. However, *year* value is associated with many more papers and even more papers that are cited by these papers. On the other hand, each paper can be cited by other papers published in different years. Thus, the *booktitle* and *year* of a paper are more correlated than its *year* and its cited papers. Hence, CR ranks a paper that is published in "1997" in "SIGMOD" higher than the candidate answer rooted at node 3 in Fig. 2 for query Q_{10} = SIGMOD 1997.

As for recall, unlike previous methods, CR correctly returns both nodes 3 and 15 for Q_3 = *XML Burt* in Fig. 1, and returns node 3 as an answer to query Q_6 = *Smith Burt* in Fig. 2. CR's recall will be equal to that of the baseline approach and higher than that of all other approaches discussed earlier, as long as the minimum correlation threshold is reasonably low. Users of approximate retrieval systems sometimes like to see as many relevant answers as possible (i.e. high recall) and sometimes they are interested in only the most relevant answers (i.e. high precision) [Manning et al. 2008]. CR enables users to control the tradeoff between the quantity (recall) and quality (precision) of the returned answers.

The idea of CR is close to the concept of functional dependencies (FDs) and approximate FDs [Huhtala et al. 1998; Dalkilic and Robertson 2000], which have been used to distinguish fine-grained entities in a DB [Maier 1983]. Researchers have extended the concept of a FD to apply to XML data [Arenas and Libkin 2004; Yu and Jagadish 2006]. However, two obstacles prevent us from using FDs here. First, FDs are unidirectional, and we need a bilateral relationship. Second, FDs are defined on two specific sets of attributes. For instance, to identify the most meaningful subtrees, should we use the FD *title* → *booktitle author*, *author* → *booktitle title*, or another FD? We remedy this problem using an extended version of mutual information.

Mutual information and other statistical measurements are used in the data mining community to find correlated data items [Brin et al. 1997; Ma and Hellerstein 2002; Ke et al. 2006; Morishita and Sese 2006]. However, these approaches typically consider only binary variables, and our variables (XML types, or paths) are usually not binary. Second, these approaches look for sets of elements whose correlation exceeds a specific threshold; the exact value of the correlation is not of interest. However, to correctly rank query answers, we may need to compute exact correlations. Third, their measurements are upward closed, meaning that if a set of variables is correlated, at least one subset of them is correlated [Brin et al. 1997; Ma and Hellerstein 2002; Ke et al. 2006; Morishita and Sese 2006]. This makes the correlation easy to compute. But in keyword search, we may prefer one answer over another that contains it, even if the larger tree has a higher correlation. Thus our measurement should not be upward closed. Third, these approaches redefine the correlation measurement to make it easier and faster to compute. For example, one approach only considers mutual information among pairs of variables in a set, rather than among all possible subsets of the set [Ma and Hellerstein 2002]. But we must consider the correlation among all nodes in a subtree, not just pairs. For example, consider a bibliographic DB with conference names, years, and paper titles. The correlation between conference names and years will be low, as each conference is held in multiple years and each year has many different conferences.

But the correlation between the three types conference name, year, and title is quite high, as the conference name and year together are associated with only a small set of paper titles. Overall, we conclude that we cannot simply apply previous work from the data mining community to rank candidate answers.

Mutual information has been used to improve retrieval effectiveness for unstructured and semi structured documents [Jones et al. 2006; Petkova et al. 2009; Jang et al. 1999; Schenkel and Theobald 2006]. Jones et al. and Petkova et al. use mutual information between terms to extract phrases in unstructured and semi-structured documents. Petkova et al. also use information gain to discover the correct ancestral order among tag labels in ambiguous XML structural queries and refine the queries. Schenkel and Theobald use mutual information to expand the structural queries over XML documents using relevance-feedback techniques. Jang et al. use mutual information to eliminate query ambiguities in cross-language information retrieval. These approaches consider mutual information between terms, rather than between schema elements. Our work is orthogonal to these methods and can use these methods to improve its effectiveness.

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. As mentioned earlier, our techniques are for data-centric XML, not text-centric XML. Nodes with a lot of text, such as the body of an article, are likely to be very highly correlated with certain other fields. 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.

4. MEASURING COHERENCY

In this section, we develop a ranking framework for XML query answers, based on the concept of CR. For brevity, we will use the term *DB* to refer to XML DBs. Also, in this section we ignore all non-content leaf nodes, as they do not affect rankings.

4.1 Preliminaries

Consider the depth-first traversal of a tree, where we always visit the children of a node in 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 [Zaki 2005]. For instance, the prefix string for the subtree rooted at node 4 in Fig. 1 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. P_1 is a subpattern of P_2 if P_1's trees are all subtrees of P_2's trees.

Definition 4.1. 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. 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 4.2. 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 [Cover and Thomas 1983]. 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, 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. Generally, one value of a root-path can be associated with one or more values of other root-path(s). Before showing how to apply information-theoretic concepts to patterns, we quickly review the traditional definitions of entropy and joint entropy.

Definition 4.3. 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 $H(p) = \sum_{1 \le j \le n} P(a_j) \lg (1/P(a_j))$.

Intuitively, the entropy of a random variable indicates how predictable the variable is. The entropy is minimal (zero) when all instances of p have the same value, and maximal ($\lg n$) when no two instances of p have the same value. When the pattern is not a root-path pattern, we refer to its entropy as *joint* entropy as well, because it explains the behavior of a joint random variable. We refer to the *joint* entropy of pattern t both by $H(t)$ and $H(p_1, \ldots, p_n)$ where $p_1, \ldots, p_n$ are the root-path patterns of t. The properties of XML entropy are the same as for tabular data, except for the following property:

PROPOSITION 4.4. *If root-tree pattern t has exactly one root-path pattern p, then*

$H(p) \leq H(t)$.

This property does not hold for traditional joint entropy, where $H(x,x) = H(x)$.

In the context of the relational model, researchers [Dalkilic and Robertson 2000] have defined FDs based on conditional entropy [Cover and Thomas 1983]. We extend conditional probability and conditional entropy to apply to XML variables:

Definition 4.5. Given pattern t containing two root-path patterns p_1 and p_2 that take values $a_i, 1 \leq i \leq n$ and $b_j, 1 \leq j \leq m$. respectively, the **conditional probability** of $p_1|p_2$ is: $P(p_1 = a_i|p_2 = b_j) = \frac{P(p_1=a_i, p_2=b_j)}{P(p_2=b_j)}$.

Definition 4.6. Given pattern t including two root-path patterns p_1 and p_2 that take values $a_i, 1 \leq i \leq n$ and $b_j, 1 \leq j \leq m$ respectively, the **conditional entropy** of $p_1|p_2$ is:

$$H(p_1|p_2) = \sum_i \sum_j P(p_1 = a_i|p_2 = b_j) \lg (1/P(p_1 = a_i|p_2 = b_j)).$$

The conditional entropy of two root-path patterns is the entropy of the first one given information about the value of the second. The next property follows from the properties of conditional entropy in the original formulation of entropy:

PROPOSITION 4.7. *Given pattern t with two root-path patterns p_1 and p_2, $H(p_1|p_2) = H(p_1, p_2) - H(p_2)$. Furthermore,*

$$H(p_1, \ldots, p_n) = \sum_{1 \leq i \leq n} H(p_i|p_1, \ldots, p_{i-1}).$$

4.2 Coherency Ranking & NTC

Mutual information [Cover and Thomas 1983] measures the correlation between two random variables:

Definition 4.8. The **mutual information** of random variables A and B that take values $a_i, 1 \leq i \leq n$ and $b_j, 1 \leq j \leq m$ respectively, is:

$$M(A,B) = \sum_{1 \leq i \leq n} \sum_{1 \leq j \leq m} p(a_i, b_j) \lg (p(a_i, b_j)/p(a_i)p(b_j)). \tag{1}$$

where variable A, B takes value (a_i, b_j) with probability $p(a_i, b_j)$, for $1 \leq i \leq n$ and $1 \leq j \leq m$.

We have:

$$M(A,B) = H(A) + H(B) - H(A,B). \tag{2}$$

We extend the traditional definition of mutual information to apply to XML.

Definition 4.9. The **mutual information** of a pattern t with size two is

$$M(t) = \sum_{1 \leq i \leq n} \sum_{1 \leq j \leq m} P(a_i, b_j) \lg (P(a_i, b_j)/P(a_i)P(b_j)) \tag{3}$$

where t takes value (a_i, b_j) with probability $P(a_i, b_j)$, for $1 \leq i \leq n$ and $1 \leq j \leq m$.

According to the definition of the values of a pattern, the a_is and b_js are the values of the root-paths of pattern t. For instance, in Fig. 6 pattern *imdb movie title -1 locations location -1 -1 -1* takes values ((“For Love of Money”,“New York”),(“Big Love”,“Fox Hills”)). Therefore, we have P(“For Love of Money”,“New York”) = 1/2 and P(“Big Love”,“Fox Hills”) = 1/2. Also, we have P(“For Love of Money”) = 1/2, P(“Big Love”)= 1/2, P(“New York”) = 1/2, P(“Fox Hills”)= 1/2. Thus, the value of mutual information for this pattern is 1.

Mutual information measures the reduction of uncertainty in the values of one root-path pattern that comes from knowing the value of the second one. Mutual information is minimal when each value of one root-path pattern is associated with every value of the other root-path pattern, and is maximal when each value of one root-path pattern is associated with just one value of the other and vice versa. For instance, the mutual information of pattern *bib paper title -1 author -1 -1* whose paths are *bib paper title -1 -1* and *bib paper author -1 -1* is higher than the mutual information of *bib paper booktitle -1 author -1 -1* whose paths are *bib paper booktitle -1 -1* and *bib paper author -1 -1* in Fig. 1. Therefore, the relationship between the first two variables is stronger.

The next proposition follows immediately from the properties of mutual information, and is useful in computing mutual information.

Proposition 4.10. *Given pattern t containing root-path patterns p_1 and p_2, we have $M(t) = H(p_1) + H(p_2) - H(t)$.*

To compute the mutual information of t, we consider only the instances of the root-path p_i that are subtrees of the instances of the root-pattern t. Notice that with non-XML mutual information we have $M(A, A) = H(A)$, but as redefined for XML data, we have $M(t) \leq H(p)$, where t includes only one root-path pattern p.

Total correlation [Watanabe 1960] is closely related to mutual information; it measures the correlation between more than two random variables.

Definition 4.11. The **total correlation** of the random variables $A_1, \ldots, A_n$ is:

$$I(t) = \sum_{1 \leq i \leq n} H(A_1) - H(A_1, \ldots, A_n). \tag{4}$$

We extend this definition to apply to XML DBs:

Definition 4.12. Let $p_1, \ldots, p_n$ be the root-path patterns of pattern t. The **total correlation** of t is:

$$I(t) = \sum_{1 \leq i \leq n} H(p_i) - H(p_1, \ldots, p_n). \tag{5}$$

Total correlation does not consider the possibility that a pattern may have higher entropy just because it has more values. Similarly, total correlation 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. 3, 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 $I(t) > I(t')$. NTC addresses this, in a manner similar to that used to normalize mutual information [Witten and Frank 2005]. We define the **normalized total correlation (NTC)** of t as:

$$\hat{I}(t) = f(n) \times (\frac{I(t)}{H(p_1, \ldots, p_n)}). \tag{6}$$

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. A systematic exploration of the options for $f(n)$ is an interesting area for future work.

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 our $f(n)$ function is undefined for them.) If desired, one can set a low threshold $NTC_{min} \geq 0$ appropriate for the domain, and include only candidate answers whose patterns have an NTC greater than 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 candidate answers.

For example, consider the query *XML SIGMOD* in Fig. 1. 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, 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). CR also correctly differentiates between the relationship of nodes 15 and 16 and nodes 2 and 3 in Fig. 1, 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. In this paper, we set $NTC_{min} = 0$ unless otherwise noted, which is an extremely conservative setting.

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.

5. EQUIVALENCE-PRESERVING SCHEMA TRANSFORMATIONS

In this section, we show that as the current approaches discussed in Section 2.2 rely on shallow schema properties, they give different rankings for different designs of the same data set. We also show that CR performs well on different designs of the same DB, by avoiding *overreliance* on schema details and discovering the internal relationships itself.

Usually, there are multiple "good" schemas for representing the same set of information [Arenas and Libkin 2004; Yu and Jagadish 2006]. For example, Figures 2 and 3 show two different schemas for the DBLP data set. An ideal query answer filtering and ranking approach will provide the same answers in the same order, no matter which schema is used. However, the current approaches discussed in Section 2.2 give different rankings for different designs of the same data set. For instance, XSearch [Cohen et al. 2003] ranks smaller subtrees higher. Therefore, it ranks the pattern *bib paper title -1 booktitle -1 -1* higher than the pattern *bib paper title -1 author name -1 -1 -1* in Fig. 1. The former is transformed to the pattern *bib proceedings title -1 paper title -1 -1 -1* and the latter is transformed to the pattern *bib proceedings paper title -1 author name -1 -1 -1 -1* in the redesign of the DBLP data set shown in Fig. 3. Nevertheless, XSearch gives the same rank to the transformed patterns in the redesign of the DBLP data set, as they have the same distance. XSearch delivers different rankings for the fragments of DBLP shown in Fig. 4 and its redesign in Fig. 5 for Q_{11} = *MPI Addison*. However, if an article's title really is more closely related to its author than to its publisher, then we need to be able to recognize this and reflect it in our ranking of the answers to queries, *no matter what schema is used.*

XRank [Guo et al. 2003] has the same deficiency. It uses the idea of PageRank [Brin and Page 1998] and, roughly speaking, ranks the nodes having more descendants higher. Fig. 4 and Fig. 5 show different designs for the same data set. XRank ranks the authors who write more articles higher in the design shown in Fig. 5. However, XRank gives all authors the same rank when searching over the data set shown in Fig. 4. XReal [Bao et al. 2009] has the same problem, as it relies on the distances, depths, and the numbers of children of the roots of the candidate answers to rank and/or filter them. Thus, current approaches may give high precision or desirable rankings for one database design, but low precision and poor ranking for another design of the same data set. Since they rely on schema design details, small changes in the schema can cause big changes in query answers.

CR has the nice property of delivering the same ranking for different designs of the same data set, if the designs meet the basic conditions presented below. For brevity, we will use the term *mapping* to refer to *bijective mappings* in the reminder of this section.

Definition 5.1. The pattern instance S_a of database DB_a is **value isomorphic** to the pattern instance S_b of database DB_b if and only if there is a mapping between the value of S_a and the value of S_b that maps each member v_a of S_a to a member v_b of p_b, where v_a and v_b are lexicographically equal.

For example, the pattern instance *1 3 10 -1 13 -1 -1* shown in Fig. 1 and the pattern instance *1 3 10 -1 11 18 -1 -1 -1* shown in Fig. 3 are value isomorphic. According to Definition 5.1, if two pattern instances are value isomorphic, they have the same number of members. Thus, their patterns must have the same size.

Definition 5.2. The pattern t_a of XML database DB_a and the pattern t_b of database DB_b are **subtree isomorphic** if and only if:

—If their sizes are greater than one, there is a mapping m_1 between the root-path patterns of t_a called p_a and t_b called p_b, such that if $m_1(p_a) = p_b$, then p_a and p_b are subtree isomorphic.

—Otherwise, there is a mapping m_2 between the instances of t_a and t_b such that if $m_2(t_a) = t_b$, then n_a and n_b are value isomorphic.

For instance, root-path patterns *bib incollection author last -1 -1 -1* and *bib incollection title -1 -1* depicted in Fig. 4 are subtree isomorphic to root path patterns *bib author last -1 -1* and *bib author incollections incollection title -1 -1 -1 -1* depicted in Fig. 5, respectively. Hence, root-subtree pattern *bib incollection author last -1 -1 title -1 -1* shown in Fig. 4 and root-subtree pattern *bib author last -1 incollections incollection title -1 -1 -1 -1* shown in Fig. 5 are subtree isomorphic.

THEOREM 5.3. *Subtree isomorphic patterns have the same NTC value.*

PROOF. Since the instances of subtree isomorphic patterns are value isomorphic, they have the same size. Hence, if one of them is a root-path pattern, the other one must be a root-path pattern also. Consider the case where the patterns are both root-path patterns. Since they have the same number of instances and their instances are value isomorphic, their entropies are equal. Therefore, they have the same NTC value. If the patterns are not root-path patterns, then similar to the previous case, their joint entropies must be equal. Moreover, according to the definition of subtree isomorphism, their root-path patterns are subtree isomorphic, and therefore have the same entropy values. Thus, the NTC values of subtree isomorphic patterns are equal. □

According to Theorem 5.3, CR gives the same rank to subtree isomorphic patterns. Since the instances of subtree isomorphic patterns are value isomorphic, they appear in databases containing the same content but under different designs. As we will discuss in Section 6, we only need to consider the candidate answers whose patterns' sizes are less than or equal to a relatively small number. Hence, if all patterns of size up to a maximum answer size of databases DB_a and DB_b are subtree isomorphic, then CR delivers the same ranking for all keyword queries over databases DB_a and DB_b.

6. PRECOMPUTING COHERENCY RANKS

NTCs can be computed offline in advance with a populated version of the DB, and then used at query time. In this section we introduce methods to make this precomputation efficient.

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 [Zaki 2005; Wang and Liu 1998; Nijssen and Kok 2003]. These works find trees in the forest that have at least one subtree isomorphic to the candidate; but to compute its NTC, we must find *all* instances of a pattern. To do this, we can express a newly found pattern as an XML twig query and use existing algorithms to find its instances [Bruno et al. 2002; Lu et al. 2005].

Unfortunately, the naive method is so inefficient as to be impractical. First, this method generates and finds all types of subtrees, but we only want root-path subtrees, and there are relatively few root-paths in XML DBs [Choi 2002]. Twig query algorithms examine each node in the DB, while we only need to consider root-paths to find all the instances of a root-tree pattern. Second, to compute the entropy of a pattern, we will have to find and store all its values in a table-like data structure and then scan the stored values. This table can be much larger than the DB itself. If the XML file is huge and has a deep nested structure, the size of the table could be prohibitively large. For example, consider the size of such a table for the pattern *bib paper cite paper title -1 -1 -1 cite paper title -1 -1 -1 -1* in Fig. 1.

Third, each NTC must be computed separately; there is no upward closure property [Brin et al. 1997] to help us, no general way to detect that NTC is below a cutoff threshold [Morishita and Sese 2006], no way to use sampling [Ilyas et al. 2004] (because the number of distinct values is quite close to the number of instances for some patterns), and no way to use XML query selectivity estimation techniques (because they assume that the conditional entropy between root-paths is zero). Fourth, as there is no limit on the size of a candidate pattern, the process of finding all patterns in a large database can take an extremely long time. To address this problem, the tree mining methods use the number of instances of a pattern as a way to prune unlikely candidate answers. Unfortunately, we do not expect our target patterns to be relatively infrequent. There could be very few infrequent patterns, due to noisy data. Finally, these methods scan the whole database to find the instances of each pattern, which is very costly.

We use four approaches to speed up precomputation. First, we exploit the typical characteristics of users' keyword queries. Second, we use properties of NTC and entropy to reduce computation. Third, we use compressed indexes to save memory space and speed up pattern generation. Fourth, we introduce methods to estimate entropy and approximate NTC efficiently. We discuss each of these below.

Keyword Search Parameters. Previous studies of internet document retrieval have shown that the average query has roughly 2.5 keywords [Wen et al. 2001]. 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 MAS (maximum answer size) on the value of n (pattern size) considered when precomputing NTCs. Since the average length of keyword queries for different bibliographic databases is between 1.81 to 3.66 including stop words [Wolfram 2001], setting MAS 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. We show that if a pattern's LCA has only one instance in the DB (e.g., the DB root), its NTC is very close to zero.

A **prefix-path** is a path whose root is the root of DB and its last node is not a leaf node or a parent of a leaf node. For instance, */bib/paper* is a prefix-path in Fig. 1. If the last node of prefix-path s is the LCA of pattern p, s is a prefix-path of p. For instance, */bib/paper* is a prefix-path of *bib paper title -1 booktitle -1 -1* in Fig. 1. Prefix-paths such as u that are created by adding nodes to prefix-path s are the **super-paths** of s. For instance, */bib/paper* is a super-path of */bib* in Fig. 1. Every prefix-path has at least one **instance** in DB. For example, */1/2* is an instance of */bib/paper* in Fig. 1. Each instance of prefix-path s is a **prefix-path instance** of at least one instance of s's patterns. Given i that is an instance of prefix-path s, the distinct values of root-path p whose prefix-path instance is i is **projection** of p under i which is shown as V_p^i. $|V_p^i|$ shows the number of such values.

Consider pattern t that contains paths $p_1, \ldots, p_n$ and its prefix-path s has only one instance in DB. If $p_1, \ldots, p_n$ do not share any other prefix-path that is a super-path of s, each instance of p_i is associated with every instance of p_j, for $i \neq j$ and $1 \leq i, j \leq n$. Therefore, for the joint probability of $(p_1, \ldots, p_n)$ we have: $P(p_1, \ldots, p_n) = P(p_1) \cdots P(p_n)$. Hence, $H(t) = \sum_{1 \leq i \leq n} H(p_i)$. Considering formula (6), the value of NTC of t will be zero. Assume that $p_1, \ldots, p_n$ share at least one prefix-path l other than s. l is a super-path of s. Therefore, each instance of p_i will be associated with at least $|V_{p_j}^s| - |V_{p_j}^l|$ instances of p_j, for $i \neq j$ and $1 \leq i, j \leq n$ where $|V_{p_j}^l| = \max_{m \in l} |V_{p_j}^l|$ and m is an instance of l. If the value of $|V_{p_j}^l|$ is far less than the value of $|V_{p_j}^s|$, similar to the previous case, we can assume that: $P(p_1, \ldots, p_n) = P(p_1) \cdots P(p_n)$ and therefore, the value of NTC of t will be zero. This usually happens for the patterns whose LCAs is the root of DB. For instance, the number of distinct values for */bib/paper/title* or */bib/paper/author* under the root of DB are by far larger than the number of distinct values for the same paths under */bib/paper/*. In other words, knowing the value of root-path p_i does not help to narrow down the value of another root-path p_j when their only relationship is that they belong to the same DB.

During the parsing of the input XML data set, we identify the prefix-paths such as s that have only one instance in the DB and store the numbers of the distinct values of their paths under all possible super-paths. If the numbers of the distinct values of all paths under s are by far larger than the numbers of their distinct values under all super-paths of s, we ignore all patterns whose LCA is s during NTC computation.

Entropy Computation. Since we need only to compare different values of a pattern to compute its entropy, we can use the hashed values of the pattern instead of its actual values. This reduces the memory consumption of the entropy computation process and speeds up the comparison operation, as the hashed values generally occupy less space than the original values. The precision of this method depends on how conflict-free the hash function is. Our experiments use the FNV-64 hash function [FNV Hash Function 2009], whose output is relatively short and is almost conflict-free.

Approximation. To reduce the number of patterns to find, we adapt an approximation approach designed for mutual information [Kojadinovic 2005]. We use another correlation measurement in information theory called **interaction infor-**

mation [McGill 1954]:

Definition 6.1. The **interaction information** of random variables $A, \ldots, A_n$ is:

$$Intr(t) = -\sum_{T \subseteq \{A_1, \ldots, A_n\}} (-1)^{n-|T|} H(T). \tag{7}$$

We extend the definition of interaction information [McGill 1954] to work with XML:

Definition 6.2. Let $p_1, \ldots, p_n$ be the root-path patterns of pattern t. Then the **interaction information** of t is:

$$Intr(t) = -\sum_{T \subseteq \{p_1, \ldots, p_n\}} (-1)^{n-|T|} H(T). \tag{8}$$

In contrast to total correlation, this metric subtracts the correlation inside the patterns defined by T from the correlation of T itself. For instance, consider (bib/book/title, bib/book/author, bib/book/price), where all share the same *book* parent node, in a bibliographic DB. Their interaction information is close to zero because the correlation of (title, author) with the price of the book is almost equal to the sum of the correlations of (title, price) and (author, price). On the other hand, the interaction information between root-path patterns (bib/book/ISBN, bib/book/title, bib/book/author), where they share the same *book* parent, is negative. This is because the combination of ISBN and title does not provide more information about the author than ISBN alone.

We can compute the entropy of a pattern t with root-paths $p_1, \ldots, p_n$, by using the entropies and interaction information of its subtrees T as follows [Kojadinovic 2005]:

$$H(t) = \sum_{1 \leq i \leq n} H(p_i) + \sum_{T \subseteq \{p_1, \ldots, p_n\}} (-1)^{|T|} Intr(T). \tag{9}$$

We can approximate the entropy of pattern t by computing the information interaction of subtrees T of size m, up to a value $m \leq EV$, where $0 < EV < n$. The formula works well if the sum of the correlations of the subtrees T is close to the total correlation of the pattern, as for title, author, and price in the last example. However, its error is quite high for cases like ISBN, title, and author, where there are multiple keys or quasi-keys (e.g., ISBN and book title) in the pattern t. While keyword queries do not usually provide multiple key or quasi-key values, experiments with real-world queries showed us that the errors were high enough in practice for us to prefer a different formula, based on the fact that the maximum value of the entropy of a pattern t is the summation of the entropies of its root-paths $p_1, \ldots, p_n$:

$$H(t) = \sum_{1 \leq i \leq n} H(p_i) + \min\,(0, \sum_{T \subseteq \{p_1, \ldots, p_n\}} (-1)^{|T|} Intr(T)) \tag{10}$$

Using the above approach, we compute the exact NTC of patterns up to a certain size $EV \leq MAS$. Then, we approximate the NTC of larger patterns using formula (10).

How should we choose the value of EV? Even if the information interaction starts to approach zero for subset T of size $m < n$, it may increase for the subsets of size $m+1$. Thus, one should set EV to be as large as possible, based on the computational time available for precomputation (e.g., half a day). Our experiments in Section 8 on multiple data sets show that even low values of EV are very effective.

7. SEARCH & RANKING ALGORITHMS

7.1 Ranking Precomputation Algorithms

The algorithm in Fig. 7 shows the overall process 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. For efficiency, we should use information gained in previous stages to find instances of future patterns. As discussed earlier, modern tree mining algorithms produce new candidates by joining current candidate trees together, and adding new nodes to current candidate trees. When extending a candidate, these algorithms cannot use information about the positions of the instances of the current candidate in the DB. Therefore, they have to scan all the information in the DB to enumerate the number of instances for the new candidate. Furthermore, they generate many candidates that are not patterns. Therefore, we extend current tree mining approaches to generate patterns more efficiently.

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 7.1. 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.

The first part is counterintuitive, but it will be justified when we prove the correctness of our algorithm.

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

Definition 7.2. A pattern t is **sorted** when for every pair of sibling proper subtrees u and u', $u \leq u'$ 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.

LEMMA 7.3. *Every prefix of a sorted pattern is sorted.*

The proof of the lemma uses the concept of the **last path** of tree t, which 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. The *last rightmost node* of a subtree is the last node in its last path.

PROOF. Suppose we have $s \geq r$ for subtrees s and r. Let s' be obtained from s by removing its last rightmost node. Then $s' \geq r$. Removing the last rightmost

```
Input: XML data file $data$
Input: Maximum size $EV$ of patterns to compute exact NTC for
Input: Maximum size $MAS$ 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 */
$indxs$ = Depth_First_Scan($data$, $MIN\_FREQ$);
// Compute the entropy for root-path patterns
forall $p \in indxs$ do
    $p$.entropy();
/* Initialize the set of prefix classes */
$pfxSet \leftarrow \emptyset$;
/* Add all root-path patterns as one prefix class */
$pfxSet$.add($indxs$);
for $k = 2$ to $MAS$ do
    $nextPfxSet \leftarrow \emptyset$;
    last = ();
    forall $pfx \in pfxSet$ do
        forall $p \in pfx$ do
            /* Compute all prefix classes whose prefixes are $p$ */
            $nextPfx \leftarrow \emptyset$;
            forall $q \in pfx$ do
                $Jnt \leftarrow$ join_Pattern($p, q$);
                forall $r \in Jnt$ do
                    if $subTrees(r) \not\subset pfxSet$ then
                        continue;
                    if $k \leq EV$ then
                        /* Join indices and get frequency as well as NTC */
                        $w \leftarrow$ join_Indices($r, indxs$);
                        if $w.freq < MIN\_FREQ$ then
                            continue;
                        CT[$r$] $\leftarrow w.\hat{I}$;
                    else
                        CT[$r$] $\leftarrow$ approximate$\hat{I}(r)$;
                    if $k \neq MAS$ then
                        $nextPfx$.add($r$);
            if $k \neq MAS$ then
                $nextPfxSet$.add($nextPfx$);
    $pfxSet \leftarrow nextPfxSet$
return $CT$;
```

Fig. 7. Algorithm to compute NTCs

node of a sorted tree t will remove the rightmost node of the proper subtrees of t whose roots are in the rightmost root-path of t. Therefore, these subtrees will still be greater than or equal to their left siblings. □

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 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 NTC computation algorithm in Fig. 7 to generate sorted patterns of size n from sorted patterns of size $n - 1$.

Definition 7.4. 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 NTC algorithm in Fig. 7 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 $O(m(d-1))$.

Theorem 7.5. *Given the set of all root-path patterns in the DB, the join operation followed by the check operation will generate every sorted pattern exactly once.*

Proof. We prove that the algorithm generates all possible sorted patterns by induction on the size of the pattern. Assume that all possible sorted patterns of size $n-1$ can be generated by the algorithm. According to Lemma 7.3, the prefix of a sorted pattern is sorted. Thus, the prefixes of patterns of size n are in the set of generated patterns of size $n-1$. All possible last paths are also in the generated set. Therefore, the algorithm generates all the sorted patterns. Since the sorted patterns are unique, they will not be generated more than once. The exception is the case when sibling node subtrees are equal. However, these patterns are generated from two equal patterns of size $n-1$. As we join each pattern with itself just once, the resulting patterns are generated only one time. $\square$

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 NTC algorithm in Fig. 7 calls the join_Pattern algorithm to generate the sorted patterns. The time complexity of generating new patterns of a particular size is $O(n^2)$, where n is the number of patterns of size $2 \leq k \leq MAS$. 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. Therefore, checking whether all subtrees of a pattern are frequent takes $O(s)$ time, where s is the size of the pattern. For patterns of size larger than EV, the NTC algorithm in Fig. 7 uses the approximation technique discussed in the previous section: it finds all the subtrees of the pattern, then uses formula (10) to approximate the total correlation. This step takes $O(s^3)$ time. The algorithm continues until it reaches the MAS answer 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. 7 scans the DB in a depth first manner and stores the instances of each root-path pattern in a separate path index. We use Dewey encoding [Tatarinov et al. 2002] to store the root-path patterns in their path indices. Dewey encoding assigns a vector of numbers to each node. It is a combination of the Dewey number of the parent of the node and the relative position of the node among its siblings. Fig. 4 shows the Dewey numbers for nodes of the DB tree. Each entry in the path index contains the Dewey number [Tatarinov et al. 2002] of the parent of the leaf child of an instance, as well as the hash of the leaf child data value. For example, the path index for path */bib/incollection/title* in Fig. 4 is (0.0.0, hash(MPI Techniques)), (0.1.0, hash(MPI Applications)). To reduce the index size, we use bitmaps to represent Dewey codes, and store the index sorted on Dewey number. In line 19, the algorithm prunes all the patterns with fewer instances than a minimum frequency threshold. The patterns with very few instances tend to have very high entropy and therefore higher NTC than more frequent patterns.

Path indices can join at different levels. The entries whose Dewey codes have the same prefix of size $l+1$ can join at level l. The join of the path index given above with itself at level 1 is given by (0.1, {hash(Moore), hash(Tob)}). Since the path indices are sorted on their Dewey codes, we can compute the instances of each pattern p by performing a zigzag join [GarciaMolina et al. 2008] on the path indices of p's root-path patterns (line 18). Thus, we do not need to scan the DB to find the instances of a pattern. Also, since we join the path indices in a certain order, we generate each result at most once. When joining the root-path pattern

	DBLP	IMDB	XMark
Size (in MB)	207.2	1038.8	233.7 - 2113.5
Max Depth	5	7	11

Table I. Data set statistics

indices in line 18, the *JoinIndices* routine inserts the values of the new pattern into a temporary hash table. Then, the algorithm computes the exact NTC for the new pattern at line 21, based on the values stored in the temporary hash 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, instead of computing the exact NTC.

7.2 Query Processing

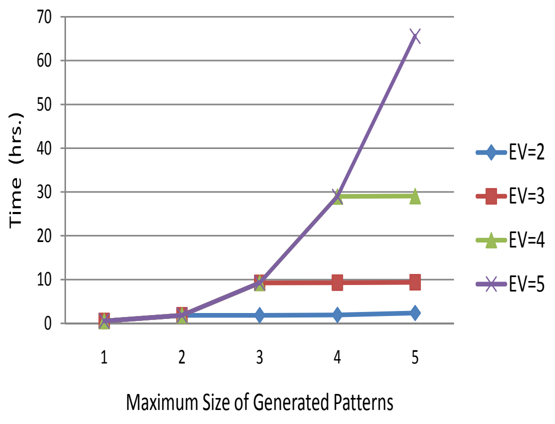
The NTC value for each pattern resides in a hash table in main memory during query processing. The query processing system finds each candidate answer, generates its pattern, looks up the NTC for that pattern, and then ranks the answer accordingly. The problem of finding all candidate answers given a keyword query has been addressed in previous work [Hristidis et al. 2006; Sun et al. 2007]. The algorithm proposed in [Sun et al. 2007] returns only the root of the subtree, while we need the whole structure of the subtree in order to find its pattern. The SA algorithm [Hristidis et al. 2006] returns the LCA, leaf nodes, and leaf node parents of each candidate answer. We extended SA to also produce the pattern of a candidate answer.

The performance of SA has been studied before [Hristidis et al. 2006; Sun et al. 2007; Li et al. 2007], and our extensions do not affect 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 extended SA to perform better than the original SA for our queries. As shown in [Hristidis et al. 2006], in some circumstances the number of relevant answers is asymptotically larger than the number of candidate answers that the SA algorithm returns. However, we decrease the number of candidate answers considerably by never considering the root of the XML tree as a candidate answer.

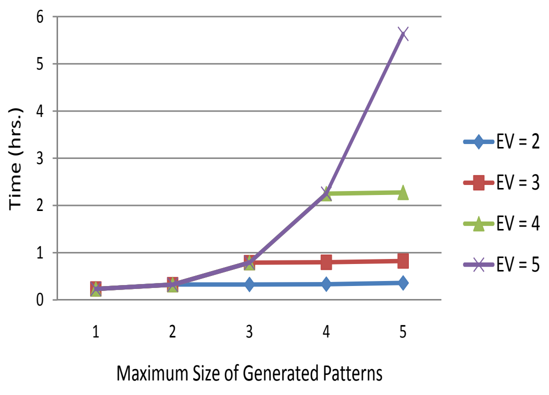
We used Berkeley DB 4.6 [Berkeley DB 2008] to store the XML data sets. Each node occupies a separate entry in the database table. The key of the node entry is the node's position in the depth-first traversal of the DB, and the value of the entry is its parent Dewey code and its content (if the node is a content node). We built an index on the parental information of the nodes to facilitate finding the parents of a node, as the SA algorithm requires. Also, we store an inverted index for the content stored in the database table.

8. EVALUATION

We implemented and evaluated the precomputation 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 data sets and one synthetic data set: DBLP,

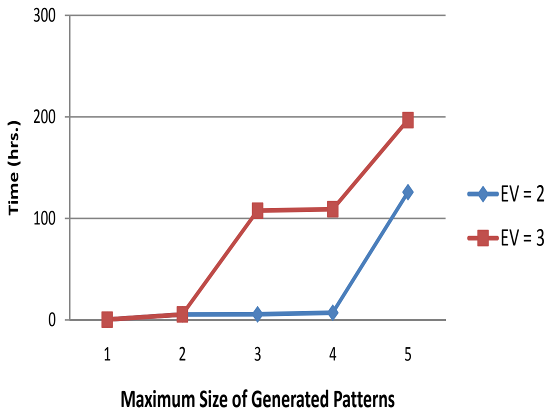


(a)

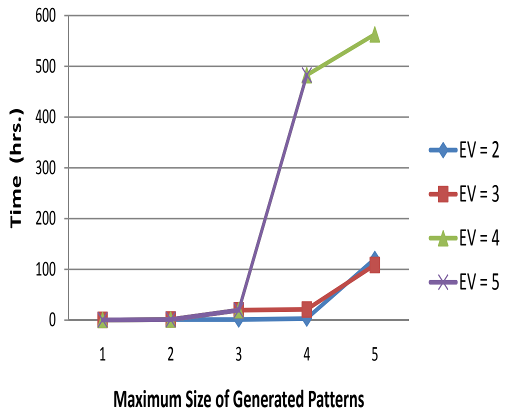


(b)

Fig. 8. Precomputation performance for DBLP without hashing (a) and with hashing (b)



(a)



(b)

Fig. 9. Precomputation performance for IMDB without hashing (a) and with hashing (b)

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

Table II. Approximation error for DBLP and IMDB

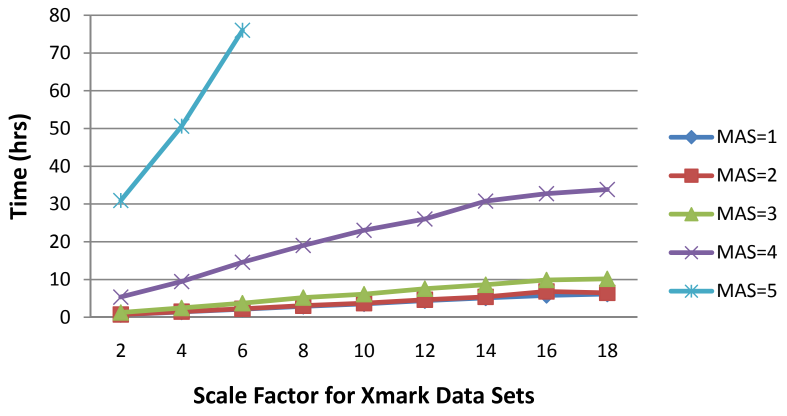


Fig. 10. Performance of exact NTC computation for Xmark data sets

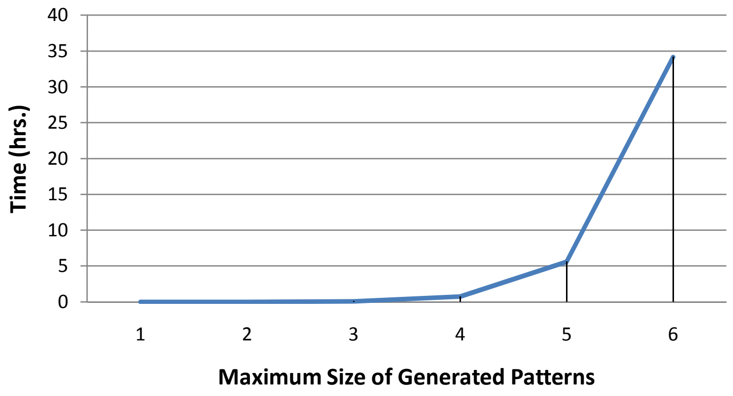


Fig. 11. Performance of approximate NTC computation for Xmark data sets for $EV = 3$

IMDB, and XMark [Schmidt et al. 2002]. Figure 9 (in Appendix) shows part of the IMDB DTD. Table I presents the properties of the data sets. We used 14 different Xmark data sets whose sizes are in the range of 233.7MB to 2113.5MB to evaluate the scalability of the precomputation algorithm. We did not consider the font tags such as *emp* as elements in XMark. No previous approach has been tried on data sets as large as our version of IMDB, or as large and deep as our XMark. Because CR is not intended for use with long text fields, we did not include the long IMDB fields such as *plot* and *bloopers* in our experiments.

8.1 NTC Precomputation Performance

We implemented the NTC precomputation algorithm in C++ using Apache's Xerces SAX parser. We set the minimum frequency threshold to 50 for all data sets; any value between 2 and 50 would have produced the same results, because IMDB and

XMark have no patterns with frequencies between 1 and 49, and DBLP has only one such pattern. We set the maximum answer size MAS to 5 for DBLP and IMDB, which is a reasonable threshold for both domains as argued earlier.

Fig. 8(a) and Fig. 9(a) show the time spent on the exact and approximate computations for DBLP and IMDB, broken down by the value for EV, when we used the original values of the content nodes to compute NTC. This includes the time to parse the XML data and create path indices. As expected, the time for exact computation increases exponentially as EV increases. 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. The exact computation for $MAS = 3$ in XMark of size 1117MB took about a week. For DBLP, the exact computation for $MAS = 5$ took about three days. The exact computation for $EV > 3$ in IMDB and XMark larger than 800MB took too long to run to completion. Overall, although approximation helps to reduce the precomputation time, precomputation still takes a long time. Thus, we repeated the experiments using the hashing technique described in Section 6. Fig. 8(b) and Fig. 9(b) depict the time spent on the exact and approximate NTC computations for the same data sets. Hashing reduces the exact NTC computation time for DBLP by more than a factor of 10 and the exact computation time for IMDB by about a factor of 5. The computation time for $EV = 3$ and $MAS = 5$ is less than for $EV = 2$ and $MAS = 5$ in IMDB data set. The algorithm does not check wether the generated patterns whose sizes are more than EV have any instances in the database. Thus, it produces more spurious patterns for smaller EV. The exact NTC computation for $MAS = 3$ in XMark took about 87 hours. The approximate computation for $EV = 3$ and $MAS = 5$ took less than two weeks. The exact computation for $MAS = 5$ in IMDB took more than three weeks. Since the number of root-paths in an XML database is relatively small [Choi 2002], the algorithm can cache all or most of the path indexes in memory. This makes choosing larger values for EV and even exact computation practical, depending on the size of the data set and the available time and resources.

Fig. 10 shows the scalability of the exact computation for different values of MAS for Xmark data sets of size 233.7MB to 2113.5MB. The exact precomputation for $MAS = 5$ for Xmark data sets larger than 800MB took too long to run to completion. The figure shows that the precomputation algorithm scales linearly with respect to the size of the data set. Fig. 11 shows the time spent on approximate computation of a typical Xmark data set for different values of MAS with $EV = 3$. Since the approximation time depends only on the number of pattern in the database, it takes the same time for XMark data sets of different sizes. As we see, the time scales exponentially with respect to the maximum size of generated patterns. The approximation algorithm took too long to run to completion for patterns larger than 6. Fortunately, keyword queries whose desired answers include more than 6 separate leaf nodes are very rare in practice [Wen et al. 2001; Wolfram 2001]. Thus, the NTC computation for such queries for very large and deep data sets (such as Xmark) can be done at query time. The query time algorithm keeps the precomputed patterns up to a given size, for instance 4, in main memory. If the size of pattern p of a candidate answer was more than the maximum size of the

	Coherency Ranking	**XSearch**	**SLCA**	**MaxMatch**	**CVLCA**	**XRank**	**XReal**
Avg. Precision	0.687	0.189	0.679	0.679	0.164	0.166	0.360
Avg. Recall	1.0	1.0	0.915	0.915	1.0	1.0	0.250

Table III. Average precision and recall of original methods for simple queries over DBLP

precomputed patterns, the query time algorithm finds all precomputed subtrees of p and approximates its NTC. Since the precomputed patterns reside in a hash table in main memory in the query-time, this step can be done very fast.

Table II shows the effectiveness of the NTC approximation technique. The table excludes XMark, because XMark content is generated randomly and is not meaningful, so the concept of an approximation error does not make sense for XMark. For the other two data sets, we sorted the list of all patterns of size up to MAS based on their exact or approximate NTC values for different values of EV. Then we computed the approximation error rate using Kendall's tau distance between two permutations of the same list of NTCs [Kendall and Gibbons 1990]. This measure penalizes the sorted approximate-computation list whenever a pattern occurs in a different position in the exact-computation list. Thus, if two lists have almost the same ordering, they have a low Kendall's tau distance. In the table, 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.

One problem in using Kendall's tau is that the approximate and exact lists have different lengths. This is because the approximation procedure cannot distinguish and eliminate certain patterns whose frequency is less than the frequency threshold. For instance, it is not possible for an approximation run with $EV = 1$ to eliminate all patterns containing a paper with two titles. Our solution to this problem was to add these patterns to the exact list, with a zero NTC. As shown in Table II, the error rate decreases as EV increases. $EV = 3$ gives an error rate close to 10%. The approximation is a bit better for IMDB than for DBLP, because there are fewer key/semi-key elements in IMDB than in DBLP. Of course, the real test is wether approximation affects the ranking of actual user queries. In the next section, we show that is does not.

In many domains, NTC precomputation will only be done once; in domains with more fluid structure, NTC precomputation will still be a rare event, e.g., once a year. The NTC precomputation times seem quite reasonable for this rare task. For $MAS = 5$, the table of NTCs produced by precomputation occupied less than 2 MB for IMDB and about 1 MB for DBLP. Thus CR imposes only a modest memory overhead at query processing time. If desired, the table size could be further reduced by removing patterns whose correlations are below a threshold value.

8.2 Query Processing

To the best of our knowledge, 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 [Manning et al. 2008]. 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,

	Coherency Ranking	XSearch	SLCA	MaxMatch	CVLCA	XRank	XReal
Avg. Precision	0.592	0.054	0.558	0.558	0.057	0.058	0.211
Avg. Recall	1.0	0.975	0.798	0.798	0.975	0.976	0.202

Table IV. Average precision and recall of original methods for simple queries over IMDB

	Coherency Ranking	XSearch-M	SLCA-M	MaxMatch-M	CVLCA-M	XRank-M	XReal-M
Avg. Precision	0.687	0.687	0.679	0.679	0.687	0.687	0.360
Avg. Recall	1.0	1.0	0.915	0.915	1.0	1.0	0.250

Table V. Average precision and recall of modified methods for simple queries over DBLP

	Coherency Ranking	XSearch-M	SLCA-M	MaxMatch-M	CVLCA-M	XRank-M	XReal-M
Avg. Precision	0.592	0.586	0.558	0.558	0.586	0.592	0.211
Avg. Recall	1.0	0.975	0.798	0.798	0.975	1.0	0.202

Table VI. Average precision and recall of modified methods for simple queries over IMDB

	Coherency Ranking	XSearch-M	SLCA-M	MaxMatch-M	CVLCA-M	XRank-M	XReal-M	Baseline-M
Avg. Precision	0.892	0.893	0.892	0.892	0.893	0.892	0.121	0.892
Avg. Recall	1.0	1.0	1.0	1.0	1.0	1.0	0.106	1.0

Table VII. Average precision and recall of modified methods for complex queries over DBLP

	Coherency Ranking	XSearch-M	SLCA-M	MaxMatch-M	CVLCA-M	XRank-M	XReal-M
Avg. Precision	0.545	0.548	0.545	0.545	0.548	0.545	0.404
Avg. Recall	1.0	1.0	1.0	1.0	1.0	1.0	0.446

Table VIII. Average precision and recall of modified methods for complex queries over IMDB

	Coherency Ranking	XSearch	PN	PN-M	XRank	XRank-M	XReal
IMDB	0.715	0.642	0.511	0.627	0.421	0.609	0.420
DBLP	0.834	0.794	0.621	0.739	0.591	0.723	0.380

Table IX. Mean Average Precision (MAP) for simple queries over DBLP and IMDB

	Coherency Ranking	XSearch	PN	PN-M	XRank	XRank-M	XReal
IMDB	0.706	0.601	0.507	0.612	0.465	0.598	0.348
DBLP	0.956	0.939	0.785	0.956	0.629	0.920	0.106

Table X. Mean Average Precision (MAP) for complex queries over DBLP and IMDB

shown in Table XI (in Appendix). There are more queries for IMDB than for DBLP because some of the users are not computer science researchers, and could not provide meaningful DBLP queries. We developed a keyword query processing prototype based on the baseline algorithm that returns every LCA for each query, without any ranking [Schmidt et al. 2001]. Our users submitted their queries to this system and judged the relevancy of each answer as relevant or irrelevant by selecting a checkbox in front of each returned 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 eliminated stop words from the queries before submitting them to the baseline algorithm.

Many submitted queries are relatively short. For instance, users submitted the main subject or parts of the title of a movie, with at most one related predicate, when searching for a movie in IMDB database. This complies with the previous user studies on the length of keyword search queries [Wen et al. 2001]. There are query assistance tools for keyword search over relational databases such as [Nandi and Jagadish 2007] that provide a ranked list of schema elements that match the keywords as user types keywords. Users can use the suggested schema elements and translated his/her keyword query to the form $(e_1 : k1, \cdots, e_n : k_n)$ where $e_i, 1 \leq n$ is a schema element and $k_i, 1 \leq n$ is a keyword from input query. The keyword search interface can determine the related answers for the translated query more precisely. In order to build such a system, DBA must make up user-friendly names for the schema elements. Also, since there may be hundreds and thousands of schema elements in the database, DBA must group semantically similar schema elements. For instance, even a proper name such as*Clooney* matches *actor*, *director*, *producer*, *writer*, *especial-thanks*, *miscellaneous-crew*, *cinematographer*, *soundtrack-info*, *character*, *keyword*, *art-department*, and *literature* in IMDB. DBA can group these schema elements in 6 groups such as *people*, *soundtrack*, *companies*, *characters*, *keywords*, and *literature*. Nevertheless, CR is a domain independent keyword query interface. Moreover, due to human minds limit to consider options [Miller 1968], the number of the suggested groups should not be more than approximately 7. Groups schema elements considering this limitation for a database with relatively complex schema is a challenging task. Moreover, since there are more than one schema element in a group, keyword search system still does not know the exact desired schema element(s) for each keyword. For instance, there are 7 schema elements in *people* group and some of them such as *especial-thanks*, *miscellaneous-crew*, and *cinematographer* are not likely to be desirable. The same limitation applies to similar works such as [Li et al. 2006]. However, we also wanted to measure the effectiveness of our technique for structurally rich keyword queries, i.e. a keyword query that deals with at least two attributes of the desired entities where each of its keywords matches more than one type of attribute in the desired entities. For instance, query *Clooney Crime Warner* submitted to IMDB database, deals with three different attributes of a movie. *Clooney* matches 12 attributes, *Crime* matches 11 attributes, and *Warner* matches 12 attributes. The movies acted in or directed by *Clooney*, having the term *Crime* in their title, and distributed or produced by *Warner Brothers* company are among the desired answers. Some combinations of matching attributes are not desirable. For example, movies produced by *Clooney*, having the term *Crime* in their location, and having the term *Warner* in their camera crew are not ideal answers. These complex combinations of desired attributes cannot be captured by query assistance tools such as [Nandi and Jagadish 2007]. In order to collect such queries, we have performed another round of user study. We provided our users with the definition of complex queries and some examples. Table XII (in Appendix) shows the collected complex queries for DBLP and IMDB.We determined the effectiveness of our technique and previous methods and compared them for both sets of queries in separate evaluation tasks.

We defined two different evaluation tasks for each query set to examine the effectiveness of CR. The first task compares CR with the methods discussed in Section 2.2: SLCA [Xu and Papakonstantinou 2005], XRank [Guo et al. 2003], CVLCA [Li et al. 2007], MaxMatch [Liu and Chen 2008], XReal [Bao et al. 2009], and XSearch [Cohen et al. 2003] for our data sets to evaluate unranked retrieval. XRank, XSearch, and XReal have tunable parameters. We have explained these parameters and how we have set them in our experiments in Appendix.

The effectiveness measurements used in the first task are recall, precision, and fall-out [Manning et al. 2008]. *Recall* gives the fraction of the relevant candidate answers that are included in the actual answer returned to the user. *Precision* gives the fraction of the returned answers that are relevant. We consider the precision of a method to be zero whenever its recall is zero. *Fall-out* shows the proportion of non-relevant answers in the returned answers. Fall-out is closely related to precision. When there is no relevant answer, a method that returns no answers has the same precision as one that returns many irrelevant answers – even though the first method is more useful than the second. Fall-out captures this difference.

We implemented the CR query processing system in Java 1.6. The query system uses the NTCs computed using $MAS = 5$, $EV = 3$, 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 the choice of maximum answer size $MAS = 5$ was reasonable. We set $NTC_{min} = 0$ for the evaluations in this paper. Larger values for NTC_{min} increase the precision but decrease the recall. If desired, one can control the recall-precision tradeoff by tuning the value of NTC_{min} in a manner similar to that used to tune the parameters of IR retrieval methods [Manning et al. 2008]. Finding the best value for NTC_{min} is a subject for future study.

Table III and Table IV show the average precision and average recall of all seven approaches on IMDB and DBLP for the simple query collection, respectively. The precision of CR (shown as "Coherency" in the figure) is as high or higher than every other method, for almost every query in both data sets. 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 *William Yurcik*, these methods return subtrees describing a paper written by an author whose name contained "William" and an article authored by a different person whose name contained "Yurcik". SLCA and MaxMatch show better precision than CVLCA, XSearch, and XRank. Our queries did not reveal any differences between SLCA and MaxMatch in terms of their precision and recall. As mentioned in Section 2.2, XReal prunes too many candidate answers. Therefore, it delivers lower precision than other methods.

The root of the database has the highest sum of the query term frequencies for most queries having more than two terms, such as *Kate Winslet 1997*, *Peter Sellers Blake Edwards*, and *Comedy Woody Allen Scarlett Johansson* in IMDB and *Closed Frequent Pattern*, *Social Network Evaluation*, and *Cis Regulatory Module 2008* in DBLP. This also happens for the relatively short queries whose terms are very frequent in the data set, such as *VLDB Korea* in DBLP. Thus, XReal returns the root of the database as the only answer for these queries, which is not the desired result.

CR shows perfect recall for all queries. For instance, in some movies Edward Norton acted and also appeared on the sound track. SLCA and MaxMatch return only the sound track nodes. For some queries, the users marked both large and small subtrees as relevant. For instance, all methods except for CR return a director subtree or composer subtree for *Satyajit Ray.* However, our users also marked as relevant the subtrees that contain *both* director and writer information, as they were interested in movies that Satyajit Ray directed and composed. Since DBLP is flatter than IMDB, the recall of SLCA and MaxMatch is better for DBLP than for IMDB. XReal filters out the entity types that have relatively low query term frequencies. For instance, it does not return the movies produced by *Edward Norton*, as *Edward Norton* appears more frequently in the *actor* element than the *producer* element. For the same reason, XReal returns only the *inproceedings* elements containing the keyword *Hoarding* and ignores the *article* and *incollections* elements about *Hoarding.* Thus, XReal shows the lowest recall.

Our users initially proposed many queries with no relevant answers. All methods but CR presented a long list of irrelevant answers. When the users protested at having to sift through so many unwanted answers, we removed all but one of those queries from the workload: *SASI Protocol Attack* for DBLP. While quite a few papers have been written about attacks on SASI protocols, this overspecified keyword query will not find any of them in DBLP. CR returned no answer for this query, and was similarly helpful for many other queries with no relevant answers. XSearch and CVLCA return 121 answers for this query, all irrelevant; SLCA, MaxMatch, and XRank return 460 answers; and XReal returns 82 answers. The DL filtering technique of XSearch and CVLCA and the filtering approach of XReal improve their performance, but they still give many unhelpful answers. Thus, CR provides better fall-out than the other methods.

Since most methods return many irrelevant answers that are rooted at the root of the DB, we reran our experiments after modifying all the algorithms to filter out answers rooted at the DB root. This modification improves the precision and ranking quality of these methods without changing their recall. We use "-M" to distinguish these methods form their original method. For instance, we call the modified version of SLCA as SLCA-M.

Table V and Table VI show the average precision and average recall of all modified approaches on IMDB and DBLP for the simple query collection, respectively. CR shows the same precision and recall as before, as it automatically filters the unrelated answers rooted at the root of the DB. XRank-M has the same precision and recall as CR, as it did not get a chance to remove a relevant answer. XSearch-M and CVLCA-M have the same recall as CR for DBLP but slightly lower recall for IMDB. Since IMDB is deeper than DBLP, XSearch-M and CVLCA-M filtered some relevant answers using DL heuristics. SLCA-M and MaxMatch-M still have lower recall than CR. The modification did not have any effect on precision and ranking quality of XReal.

XReal return elements in lower levels of the database for some queries. For example, XReal returns only the element *actor* for the query *Christian Bale* in IMDB. Our users deem this answer not to be informative enough and want to see its ancestor element *movie.* XReal provide better precision than CR for five IMDB queries and five DBLP queries. However, XReal omits many relevant answers for these ten queries, as explained below. For the query *Fight Club*, only movies having *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, as they all have one non-leaf node. Thus the approaches that rely on subtree size are unable to filter out the unwanted answers. The same thing happens for queries *Beautiful Mind, Jackie Chan 2007, Pearl Harbor, Crime the Godfather, Momento, The Watchmen, Thumbs Up, Return of the Mummy, High School Musical*, and *Basic Instinct.* Our users were looking for information about movies with these titles, but there are many movies that have these words in their keywords. None of the methods had good precision for these queries. In the second evaluation task, we will see how CR addresses this problem through returning the most relevant answer first.

For query *Edward Norton*, all methods return many movies that have one actor whose first or last name is *Edward* and another actor whose first or last name is *Norton.* The same happens for *Kate Winslet 1997.* Almost the same thing happens with queries like *Han Data Mining*, where the user wants information on the book written by *Han*, but gets papers about *Data Mining* written by *Han* as well. Later we will show how CR solves this problem through ranking. For very specific queries, like *Comedy Woody Allen Scarlett Johansson*, the candidate answers are the desired ones, and SLCA, MaxMatch, and CR are equally precise. However, as explained below, SLCA and MaxMatch omit many relevant answers. This lowers not only their recall but also their precision. Modified XReal has the lowest recall among the modified methods.

Table VII and Table VIII show the average precision and average recall on the complex queries, for all modified approaches on IMDB and DBLP, respectively. Overall, we found that asking users for more complex IMDB and DBLP queries simply led to more *constrained* queries, greatly reducing the number of candidate answers, lessening the potential margin for filtering and ranking errors, and diminishing the distinctions between methods. This can most clearly be seen in the results for DBLP, where users crafted complex DBLP queries by giving more and more details about authors, titles, and venues, shrinking the set of candidate answers with each additional term. The final set of candidate answers had cardinality 1 in some cases; the largest set, which was for *Xifeng Yan Jiawei Han*, had perhaps 20 papers authored by the gentlemen in question. The users judged over 90% of the candidate answers to the DBLP queries to be relevant. If almost all candidate answers are relevant, then almost all methods will have excellent precision. If all candidate answers are good, then ranking does not matter very much either. In such a situation, even the baseline method that returns *all* candidate answers has good precision (and by definition good recall). Finally, when there are very few candidate answers and almost all of them fit the same pattern (e.g., papers coauthored by Xifeng Yan and Jiawei Han), then there is very little room for error on the part of the heuristics that filter out patterns. In such situations, almost all the methods will have good recall. Table VII shows that the modified baseline algorithm that returns all candidate answers except for the ones rooted at the root of DB (shown in the table as *Baseline-M*)has almost the same precision and recall as other methods. Thus in our results, every method except XReal (discussed below) had almost identical precision and recall. When the baseline method works very well, then no other method's benefits are very visible; tightly constrained queries do not allow any method to show its power.

The differences between methods are slightly more visible in the IMDB data set, as it has a more nested structure with more patterns, and therefore a greater margin for error in filtering and ranking. With complex queries, XReal has the lowest recall on both data sets. XSearch-M and CVLCA-M do show slightly better precision than other methods, as they successfully filter some unrelated candidate answers using DL heuristics. Since the precision and recall of almost all methods are still almost equal, the methods differ mainly in their ranking quality, discussed below.

In the second task, we evaluated the ranking effectiveness of CR, XSearch, XRank, XRank-M and XReal using *mean average precision* (MAP). Intuitively, MAP measures how many of the relevant answers appear near the beginning of the returned list of answers [Manning et al. 2008]. To compute MAP, we first consider each query Q separately. We compute the precision of all returned answers for Q, up to and including the ith relevant answer, for each value of i. The average of these precisions is called the *average precision* for Q. The MAP is the mean of the average precisions for all queries in the workload. Since XSearch uses the size of the candidate answers to rank them, it always ranks the candidate answers rooted at the root of the DB last. Thus, the MAP values of the original and modified versions of XSearch are equal. Also, since XReal returns the instances of the same entity type for an input query, the modification does not have any impact in its ranking quality.

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 the pivoted normalization method [Manning et al. 2008] to determine the rank $r(t)$ of answer t:

$$r(t) = \alpha \hat{I}(t) + (1-\alpha) ir(t), \tag{11}$$

where $ir(t)$ is:

$$ir(t) = \sum_{w \in Q, E_l} \frac{1 + \ln(1 + \ln(tf(w)))}{(1-s) + s(el_l/avel_l)} \times qtf(w) \times \ln\left(\frac{N_l + 1}{ef_l}\right). \tag{12}$$

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_l is the count of nodes whose parent has label l, and ef_l is the number of nodes whose parent has label l and contain w. s is a constant; the IR community has found that 0.2 is the best value for s [Manning et al. 2008]. 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 DBLP and IMDB are data-oriented; thus, structural information is more important than contextual information.

Table IX illustrates the MAP values of modified methods for the simple query collection for DBLP and IMDB. The table shows that the MAP of CR is higher than that of other methods for simple queries, 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. Pivoted normalization without CR ($\alpha = 0$), shown as PN, has lower MAP than all but one other method, because PN uses only contextual information. The modified PN method (shown in the tables as *PN-M*) has better ranking quality than PN, but it has lower MAP than CR. XRank has lower MAP than PN, as its filtering strategy lowers its precision. XRank-M shows better ranking than XRank. The ranking quality of XRank and XRank-M changes considerably (about 15%) for different values of d_1, d_2, and *decay*. In order to be fair to all methods, we randomly selected 1/10 of DBLP and IMDB and 1/4 of the queries of the simple query set to tune the parameters of XRank-M. Then, we used these parameters for the real data set and query set. If we use the same parameters for DBLP and IMDB, the modified version of XRank will have a lower MAP value than CR for DBLP. Nevertheless, CR returns almost the same MAP values for different values of its parameter. XReal shows the lowest MAP. For most queries, the elements with higher query term frequencies are not the best answers. For instance, for the query *Crime the Godfather*, XReal ranks the movies whose *keyword* element contains *Godfather* higher than the movies whose title is *Godfather*, but the latter are the desired answers. XReal also ranks the movies produced at *Pearl Harbor* higher than the movies whose titles or keywords contain *Pearl Harbor*.

Table X illustrates the MAP values of modified methods for the complex query collection for DBLP and IMDB. CR has the highest MAP value for this query collection for DBLP and IMDB. Similar to ranking task for simple queries, we trained XRank and XRank-M parameters over randomly selected 1/10 of DBLP and IMDB and 1/4 of the complex queries. Generally, the MAP values of different methods are closer to each other for complex queries than for simple queries. This is because the complex queries have few matching subtrees (candidate answers). For complex queries, CR does specially well compared to the other methods in picking the best entity types. For instance, for query *Elmasri Navathe Database*, the user is looking mainly for the book and articles written by *Elmasri* and *Navathe* about database systems. CR successfully ranks the instance of these entity types at the top of its returned list. XSearch returns the first related answer at the fourth position of its list. XRank returns some articles written by others that cite the publications of *Elmasri* and *Navathe* at the top of its ranked list. The reason is that those articles have lots of citations and have larger PageRank value. As mentioned in Section 1, CR is orthogonal to PageRank style algorithms such as XRank; combining the two ia an interesting topic for future work.

In the remainder of this section, we explain how CR ranks the query answers that had low precision in the first evaluation task. We also discuss every query where CR did not rank the desired answer(s) first.

Since a book typically 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.

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, Basic Instinct*, and *Return of the Mummy* have the same problems.

Since the title of a movie is a semi-key but there are many production company nodes with the same content, the NTC of *title* is higher than that of *production company.* Therefore CR returns the movie with the title *Fight Club* as the first result for the query *Fight Club.* The other methods do not find the proper ranking for this query.

The NTC of a subtree containing only one actor node is higher than the NTC of a subtree containing two actor nodes. Therefore, CR places the subtree with one actor node at the top of the result list for the query *Edward Norton.* Also, since the NTC of an actor node is higher than the NTC of a producer node, CR ranks the producer nodes containing *Edward Norton* after the actor nodes. This ranking was the desired ranking for our users, who were not interested in the producer nodes.

Since the *location* node has less diverse values than the *keyword* node in IMDB, the NTC of title and keyword is less than the NTC of title and location. Therefore CR ranked the subtree containing title and location above the subtree containing title and keyword in the result list, for the query *Lara Croft Cambodia.* The same happens for the subtree containing the year and producer nodes versus the subtree containing the year and actor nodes, when CR answers the query *Jackie Chan 2007.* The NTC of the former is higher than the NTC of the latter, so CR ranks the movies that Jackie Chan acted in in 2007 higher than those produced by him in 2007. This ranking was what the users wanted. Because the number of producers is lower than the number of actors in each movie subtree, heuristics such as join selectivity [Yin et al. 2005] give higher rank to the movies that Jackie Chan produced in 2007.

IR-style ranking helped with some queries, such as *Artificial Intelligence.* Multiple movies include those words in their titles. Pivoted normalization correctly penalized the titles that were longer than the desired answers, and returned the desired answer at the beginning of the list.

We consider an example of how CR determines users' intentions when the query terms are ambiguous. For query *CCS 2008* on DBLP, our users wanted information about the 2008 ACM Conference on Computer and Communications Security (CCS). However, many 2008 papers have *CCS* (change control system) in their titles. CR successfully 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. CR, however, ranked the subtree containing the title and year of the proceedings fourth, after two articles about CCS, but above the conference papers about change control systems. For the query *Maude* on DBLP, since the NTC of *title* is higher than that of *author*, CR correctly ranked papers whose titles mention Maude above papers written by authors named *Maude*. XSearch ranks the latter first. XReal provides almost the same ranking as CR for this query. Nevertheless, it misses many relevant answers due to its pruning strategy.

Although CR provided a ranking close to users' expectations, it could not deliver exactly the ranking that the user wanted for some queries. We discuss each of these queries here.

The user who issued the query *Dakota Fanning Animation* was looking for all the animated movies voiced by Dakota Fanning. The word *Animation* matched both the *special effects company* and *genre* nodes. The desired answer is the subtree containing the *actress* and *genre* nodes. However, *genre* has less diverse values in IMDB than *special effects company*. Therefore CR chooses the subtree including the actress and the special effects company as its top answer, and ranks the subtree containing the actress and genre nodes second. One potential solution to this problem would be to exploit the NTC of individual words in the final ranking formula, and we plan to investigate this approach in future work.

As another problematic query, the user who submitted *Pearl Harbor* was looking for the most recent movie with this name. CR ranked the most recent movie fourth, as it could not distinguish which was the most recent. This problem also occurred for the queries *True Dreams*, *Social Visualization*, and *Stereo Vision* in DBLP and IMDB. Addressing this issue is beyond the scope of this paper; either a domain-specific hard-coded heuristic or a heuristic learned from click-through behavior could be used to help CR break ranking ties between movies of different vintages. A similar problem arose with *Basic Instinct*, where the user was looking for the first and most famous movie of the series, and marked the others as irrelevant. CR returns this movie second in the result list, rather than first. The IMDB database does not have enough information to determine popularity measures for the movies, though such information could be learned by click-through behavior.

In the query *Authenticated Dictionary* over DBLP, the user was looking for the titles where these two words occur next to each other. CR does not consider word proximity in its ranking formula, but it can be addressed by incorporating more advanced IR-style rankings [Liu et al. 2006].

9. 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 precomputation phase that exploits the structure of the data set. For each type of subtree in the data set, the precomputation 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 precomputation affordable. For query answering, we extended the SA algorithm to rank candidate answers based on the precomputed NTCs. For user-supplied queries over two real-world data sets, coherency ranking showed considerable improvements in precision, mean average precision and recall, compared to six previously proposed methods. In the future, we plan to extend coherency ranking to work with text-centric and graph-structured DBs, and experiment with its use in schema integration and natural language query processing.

ACKNOWLEDGMENTS

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

REFERENCES

Agrawal, S., Chaudhuri, S., and Das, G. 2002. DBXplorer: A System for Keyword-based Search over Relational Databases. In *Proceedings of the Eighteenth International Conference on Data Engineering.*

Arenas, M. and Libkin, L. 2004. A Normal Form for XML Documents. *ACM Transactions on Database Systems 29,* 1, 195–232.

Balmin, A., Hristidis, V., and Papakonstantinou, Y. 2004. ObjectRank: Authority-based Keyword Search in Databases. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases.*

Bao, Z., Ling, T. W., Chen, B., and Lu, J. 2009. Effective XML Keyword Search with Relevance Oriented Ranking. In *Proceedings of the Twenty Fifth International Conference on Data Engineering.*

Berkeley DB. 2008. http://www.oracle.com/technology/products/berkeley-db.

Bhalotoa, G., Hulgeri, A., Nakhe, C., Chakrabarti, S., and Sudarshan, S. 2002. Keyword Searching and Browsing in databases using BANKS. In *Proceedings of the Eighteenth International Conference on Data Engineering.*

Brin, S., Motwani, R., and Silverstein, C. 1997. Beyond Market Baskets: Generalizing Association Rules to Correlations. In *Proceedings of the Twenty Third ACM SIGMOD International Conference on Management of Data.*

Brin, S. and Page, L. 1998. The Anatomy of a Large-Scale Hypertextual Web Search Engine. In *Proceedings of the Seventh World Wide Web Conference.*

Bruno, N., Koudas, N., and Srivastava, D. 2002. Holistic Twig Joins: Optimal XML Pattern Matching. In *Proceedings of the Twenty Eighth ACM SIGMOD International Conference on Management of Data.*

Chakrabarti, S., Puniyani, K., and Das, S. 2007. Optimizing Scoring Functions and Indexes for Proximity Search in Type-annotated Corpora. In *Proceedings of the Sixteenth International World Wide Web Conference.*

Choi, B. 2002. What are real DTDs like? In *Fifth International Workshop on the Web and Databases*.

Cohen, S., Mamou, J., Kanza, Y., and Sagiv, Y. 2003. XSearch: A Semantic Search Engine for XML. In *Proceedings of the Twenty Ninth International Conference on Very Large Data Bases*.

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

Dalkilic, M. M. and Robertson, E. L. 2000. Information Dependencies. In *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*.

Denoyer, L. and Gallinari, P. 2006. The Wikipedia XML Corpus. *SIGIR Forum 40,* 1, 64–69.

FNV Hash Function. 2009. http://www.isthe.com/chongo/tech/comp/fnv/.

GarciaMolina, H., Ullman, J., and Widom, J. 2008. *Database Systems: The Complete Book.* Prentice Hall.

Golenberg, K., Kimelfeld, B., and Sagiv, Y. 2008. Keyword Proximity Search in Complex Data Graphs. In *Proceedings of the Thirty Fourth ACM SIGMOD International Conference on Management of Data*.

Guo, L., Shao, F., Botev, C., and Shanmugasundaram, J. 2003. XRANK: Ranked Keyword Search over XML Documents. In *Proceedings of the Twenty Ninth ACM SIGMOD International Conference on Management of Data*.

Hristidis, V., Koudas, N., Papakonstantinou, Y., and Srivastava, D. 2006. Keyword Proximity Search in XML Trees. *IEEE Transactions on Knowledge & Data Engineering 18,* 5, 525–536.

Hristidis, V., Papakonstantinou, Y., and Balmin, A. 2003. Keyword Proximity Search on XML Graphs. In *Proceedings of the Nineteenth International Conference on Data Engineering*.

Huhtala, Y., Kerkkeinen, J., Porkka, P., and Toivonen, H. 1998. Efficient Discovery of Functional and Approximate Dependencies Using Partitions. In *Proceedings of the Fourteenth International Conference on Data Engineering*.

Ilyas, I. F., Markl, V., Haas, P., Brown, P., and Aboulnaga, A. 2004. CORDS: Automatic Discovery of Correlations and Soft Functional Dependencies. In *Proceedings of the Thirtieth ACM SIGMOD International Conference on Management of Data*.

Jang, M.-G., Myaeng, S. H., and Park, S. Y. 1999. Using Mutual Information to Resolve Query Translation Ambiguities and Query Term Weighting. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistic*.

Jones, K. S. 1972. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation 28*, 11–21.

Jones, R., Rey, B., Madani, O., and Greiner, W. 2006. Generating Query Substitutions. In *Proceedings of the 15th international conference on World Wide Web*.

Ke, Y., Cheng, J., and Ng, W. 2006. Mining Quantitative Correlated Patterns Using an Information-Theoretic Approach. In *Proceedings of the Twelfth Annual SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Kendall, M. and Gibbons, J. D. 1990. *Rank Correlation Methods.* Edvard Arnold, London.

Kimelfeld, B. and Sagiv, Y. 2005. Finding and Approximating Top-k Answers in Keyword Proximity Search. In *Proceedings of the Twenty Fourth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*.

Kojadinovic, I. 2005. Relevance Measures for Subset Variable Selection in Regression Problems Based on K-additive Mutual Information. *Computational Statistics and Data Analysis 49*, 1205–1227.

Kourtika, G., Simitsis, A., and Ioannidis, Y. 2006. Precis: The Essence of a Query Answer. In *Proceedings of the Twenty Second International Conference on Data Engineering*.

Li, G., Feng, J., Wang, J., and Zhou, L. 2007. Effective Keyword Search for Valuable LCAs over XML Documents. In *Proceedings of the Sixteenth ACM Conference on Information and Knowledge Management*.

LI, Y., YANG, H., AND JAGADISH, H. 2006. Constructing a Generic Natural Language Interface for an XML Database. In *Proceedings of the Tenth International Conference on Extending Database Technology.*

LI, Y., YU, C., AND JAGADISH, H. V. 2004. Schema-Free XQuery. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases.*

LIU, F., YU, C., MENG, W., AND CHOWDHURY, A. 2006. Effective Keyword Search in Relational Databases. In *Proceedings of the Thirty Second ACM SIGMOD International Conference on Management of Data.*

LIU, Z. AND CHEN, Y. 2008. Reasoning and Identifying Relevant Matches for XML Keyword Search. In *Proceedings of the Thirty Fourth International Conference on Very Large Data Bases.*

LU, J., LING, T. W., CHAN, C.-Y., AND CHEN, T. 2005. From Region Encoding To Extended Dewey: On Efficient Processing of XML Twig Pattern Matching. In *Thirty First International Conference on Very Large Data Bases.*

MA, S. AND HELLERSTEIN, J. L. 2002. Mining Mutually Dependent Patterns. In *Proceedings of the 2002 IEEE International Conference on Data Mining.*

MAIER, D. 1983. *Theory of Relational Databases.* Computer Science Press, Reading, Massachusetts.

MANNING, C., RAGHAVAN, P., AND SCHTZE, H. 2008. *An Introduction to Information Retrieval.* Cambridge University Press.

MCGILL, W. J. 1954. Multivariate Information Transmission. *Psychometrika 19*, 97–116.

MILLER, R. 1968. Response time in man-computer conversational transactions. In *Proceedings of the AFIPS Fall Joint Computer Conference.*

MORISHITA, S. AND SESE, J. 2006. Traversing Itemset Lattices with Statistical Metric Pruning. In *Proceedings of the Twelfth Annual SIGKDD International Conference on Knowledge Discovery and Data Mining* .

NANDI, A. AND JAGADISH, H. V. 2007. Assisted Querying using Instant-Response Interfaces. In *Proceedings of the ACM SIGMOD International Conference on Management of Data.*

NIJSSEN, S. AND KOK, J. N. 2003. Efficient Discovery of Frequent Unordered Trees. In *First International Workshop on Mining Graphs, Trees, and Sequences.*

OASIS. 2006. Electronic Business Standard using XML. *http://www.ebxml.org*.

PETKOVA, D., CROFT, W. B., AND DIAO, Y. 2009. Refining Keyword Queries for XML Retrieval by Combining Content and Structure. In *Proceedings of the 31st European Conference on Information Retrieval.*

ROBERTSON, S. E. 1977. The Probability Ranking Principle in IR. *Journal of Documentation 33,* 4, 294–304.

SCHENKEL, R. AND THEOBALD, M. 2006. Feedback-Driven Structural Query Expansion for Ranked Retrieval of XML Data. In *Proceedings of the 10th International Conference on Extending Database Technology.*

SCHMIDT, A., KERSTEN, M., AND WINDHOUWER, M. 2001. Querying XML Documents Made Easy: Nearest Concept Queries. In *Proceedings of the Seventeenth International Conference on Data Engineering.*

SCHMIDT, A. R., WAAS, F., KERSTEN, M. L., CAREY, M. J., MANOLESCU, I., AND BUSSE, R. 2002. XMark: A Benchmark for XML Data Management. In *Proceedings of the Twenty Eighth International Conference on Very Large Data Bases.*

STEIN, L., EDDY, S., AND DOWELL, R. 2002. Distributed Sequence Annotation System. *http://www.biodas.org/documents/spec.html*.

SUN, C., CHAN, C., AND GOENKA, A. 2007. Multiway SLCA-based keyword search in XML data. In *Proceedings of the Sixteenth International World Wide Web Conference.*

TATARINOV, I., VIGLAS, S., BEYER, K., SHANMUGASUNDARAM, J., SHEKITA, E., AND ZHANG, C. 2002. Storing and Querying Ordered XML using a Relational Database System. In *Proceedings of the Twenty Eighth ACM SIGMOD International Conference on Management of Data.*

Termehchy, A. and Winslett, M. 2009. Effective, Design-Independent XML Keyword Search. In *Proceedings of the Eighteenth ACM Conference on Information and Knowledge Management.*

Trotman, A. and Geva, S. 2006. Report on the SIGIR 2006 Workshop on XML Element Retrieval Methodology. *ACM SIGIR Forum 40.*

Wang, K. and Liu, H. 1998. Discovering typical structures of documents: A Road Map Approach. In *Proceedings of the twenty First Annual International ACM SIGIR Conference on Research and Development in Information Retrieval.*

Watanabe, S. 1960. Information Theoretical Analysis of Multivariate Correlation. *IBM Journal of Research and Development 4,* 1, 66–82.

Wen, J., Nie, J., and Zhang, H. 2001. Clustering User Queries of a Search Engine. In *Proceedings of the Tenth International World Wide Web Conference.*

Win, K., Rosasillfiani, E., Ng, W., and Lim, E. 2003. A Visual Interface for Native XML Database. In *Proceedings of the Fourteenth International Conference on Database and Expert Systems Applications.*

Witten, I. and Frank, E. 2005. *Data Mining: Practical Machine Learning Tools and Techniques.* Morgan Kaufmann.

Wolfram, D. 2001. Search characteristics in different types of Web-based IR environments: Are they the same? In *Proceedings of the Tenth International World Wide Web Conference.*

Xu, Y. and Papakonstantinou, Y. 2005. Efficient Keyword Search for Smallest LCAs in XML Databases. In *Proceedings of the Thirty First ACM SIGMOD International Conference on Management of Data.*

Xu, Y. and Papakonstantinou, Y. 2008. Efficient LCA based Keyword Search in XML Data. In *Proceedings of the Eleventh International Conference on Extending Database Technology.*

Yin, X., Han, J., and Yang, J. 2005. Searching for Related Objects in Relational Databases. In *Proceedings of the Seventeenth International Conference on Scientific and Statistical Database Management.*

Yu, C. and Jagadish, H. V. 2006. Efficient discovery of XML data redundancy. In *Proceedings of the Thirty Second International Conference on Very Large Data Bases.*

Zaki, M. 2005. Efficiently Mining Frequent Trees in a Forest. *IEEE Transactions on Knowledge & Data Engineering 17,* 8, 1021–1035.

THIS DOCUMENT IS THE ONLINE-ONLY APPENDIX TO:

Using Structural Information in XML Keyword Search Effectively

ARASH TERMEHCHY
University of Illinois at Urbana-Champaign

and
MARIANNE WINSLETT
University of Illinois at Urbana-Champaign

ACM Transactions on Database Systems, Vol. V, No. N, Month 20YY, Pages 1–0??.

Parameter Setting. XRank uses parameter *decay* which ranges between 0 and 1 and limits the depth of the LCAs of candidate answers. XRank also uses the parameters d_1 and d_2 as damping factors in its customization of the PageRank algorithm

$$e(v) = \frac{1 - d_1 - d_2}{N(v)} + d_1 \sum_{(u,v)\in CE} \frac{e(u)}{N_c(u)} + d_2 \sum_{(u,v)\in CE^{-1}} e(u) \tag{13}$$

Here, CE is the set of edges from parents to their children and $CE^{-}1$ is the set of edges from children to parents in the DB tree. If there is only one type of edge in the database, then $d_2 = 0$ and the formula becomes the original PageRank formula. Geo et al. did not report effectiveness results for XRank and there has not been any study of the effect of tunable parameters on the effectiveness of XRank. We experimented with values for *decay* between 0 and 1 our experiments. Since the damping factor in PageRank is 0.85 [Brin and Page 1998], we used non-zero values for d_1 and d_2, where $d_1 + d_2 = 0.85$ similar to [Guo et al. 2003]. According to our experiments, the ranking quality of XRank depends on the value of its tunable parameters (i.e. about 15%). Our experiments also show that these parameters must be selected based on the properties of the data set and query set. In order to get realistic results and be fair to other methods, we trained XRank tuning parameters on smaller training set and used those values for the actual data set and query set. Thus, we randomly selected 1/4 of each data set and query set to tune the parameters of XRank. Then, we used these parameters for the real data set and query set.

XSearch uses two tunable parameters α and β that control the effect of the IR-style ranking and the size of the candidate answer, respectively. There has not been

```
<!ELEMENT imdb (movie*, show*)>
<!ELEMENT movie (title, year, keywords, actors, directors, writers, genres, rating*, literature*, sound_track*>
<!ELEMENT show (title, year, actors, release_dates,...)>
<!ELEMENT sound_track (title, info*)>
<!ELEMENT rating (votes, rank)>
```

Fig. 12. Part of the IMDB database DTD

any deep study on the effect of these parameters on the effectiveness of XSearch method. We have performed an effectiveness experiments using values for α and β between 1 and 10, and compared the results to CR. Our experiments showed that changing the values of these parameters has very low impact on the effectiveness of XSearch i.e. less than 2%. The reason is that the size of most candidate answers were the same for both data sets. XSearch shows its best ranking quality for both data sets when the values of α and β are equal. We report only the best effectiveness results for XSearch in this section.

Similar to XRank, XReal uses a tunable parameter r that limits the depth of the LCAs of the candidate answers. [Bao et al. 2009] proposes 0.8 for r and uses this value in their experiments. We have used different values in the range (0,1] for r for XReal and compared the results to CR. Generally, different values of r deliver the same results in our experiments. The only exception is $r = 1$, which reduces precision, recall, and ranking quality of XReal by about 2%. We report only the maximum values for precision, recall, and ranking quality of XReal in this section.

Num	IMDB	DBLP
1	Beautiful Mind	Cis Regulatory Module 2008
2	Edward Norton	William Yurcik
3	Artificial Intelligence	Radu Sion
4	True Dreams	Barbara Liskov
5	Jackie Chan 2007	Hoarding
6	Basic Instinct	Authenticated Dictionary
7	Comedy Woody Allen Scarlett Johansson	Language Based Security
8	Peter Sellers Blake Edwards	CCS 2008
9	Belgian Detective	Closed Frequent Pattern
10	Slumdog Millionaire 2008	Social Network Evaluation
11	Crime the Godfather	Personalized Web Search
12	Forrest Gump	SASI Protocol Attack
13	Kate Winslet 1997	Trust Management Web Services
14	Pearl Harbor	Authorization Web Services
15	Susan Sarandon	VLDB Korea
16	The World of Apu	Web Conversation
17	Satyajit Ray	Maude
18	Doctor Zhivago	TinyOs Motes
19	Thumbs Up	Data Aggregation Sensys 2004
20	Grand Torino	Madden TinyDB
21	Brad Pitt Movie	TEEVE
22	Painted Skin	Stereo Vision
23	Ang Pamana	Social Visualization
24	The Owl and the Sparrow	The Dynamics of Mass Interaction
25	Hayden Christensen	Han Data Mining
26	Ben Barnes	
27	Christian Bale	
28	Fight Club	
29	The Watchmen	
30	Momento	
31	Return of the Mummy	
32	High School Musical	
33	Kate Winslet Award	
34	Britney Spears Disney	
35	Lara Croft Cambodia	
36	The Wizard of Speed and Time	
37	Mike Jittlov	
38	The Unforgiven Clint Eastwood	
39	Dakota Fanning Animation	
40	Disney Channel Movie Surfing	

Table XI. IMDB and DBLP queries

Num	IMDB	DBLP
1	Lasseter Hanks Docter	Jiawei Han Xifeng Yan
2	Brad Pitt Catherine George Clooney	Aiken Foster Kodumal
3	Boyle Trainspotting Vidler	Aiken Kodumal PLDI
4	Animation Adventure Comedy	Banerjee Amtoft Bandhakavi POPL
5	Terminator Arnold Claire	Amtoft Banerjee
6	Psycho Thriller Japanese	Hasan Winslett Radu
7	Kate Winslet Drama 2003	Hasan Winslett Policy
8	Clooney Pitt Roberts	Boyer Lee
9	Mckellen Bloom Jackson	Mittal Borisov Security
10	Drama Lynch Watts	Minami Lee
11	Clooney Pitt Mac	Han SIGMOD Frequent Patterns
12	Gould Affleck Caan	Keyword Search Databases
13	Jemison Cheadle Qin	Cabello Chambers
14	Reiner Damon Garcia	Planar Shortest Path
15	Robert Clooney Pitt	Klein Planar Flow
16	Tsunami Earth Ship	Surface Graph Computer Vision
17	Ghost Hunt Drama	TCC Public Key Cryptography
18	Family Life Nature	Bioinformatics Markov Chain
19	Romance Comedy Death	Xifeng Yan Graph Jiawei Han
20	Military World Comedy	SIGMOD Database Storage
21	Clooney Crime Warner	Binner Chen Networks
22	Julia Roberts Romance Fox	Ullman Molina SIGMOD
23	Brad Pitt Crime Warner	IPv6 Mobile Security
24	Hugh Grant Columbia	Cryptography Key Authentication
25	Clooney Action Fox	Ullman Molina Widom
26	Hijacking English	Lazebnik Schmid Ponce
27	World War Japanese	Agarwal Awan Roth
28	Mathematics Professor USA	Cootes Edwards Taylor
29		Nordstrom Nelson Phillips
30		Patterson Hennessy Architecture
31		Agarwal Roth Object Detection
32		Elmasri Navathe Database

Table XII. IMDB and DBLP complex queries