Refining keyword queries for XML retrieval by combining content and structure

Desislava Petkova
Department of Computer Science
University of Massachusetts Amherst
Amherst, MA 01003
petkova@cs.umass.edu

W. Bruce Croft
Department of Computer Science
University of Massachusetts Amherst
Amherst, MA 01003
croft@cs.umass.edu

Yanlei Diao
Department of Computer Science
University of Massachusetts Amherst
Amherst, MA 01003
yanlei@cs.umass.edu

ABSTRACT

The structural heterogeneity and complexity of XML repositories makes query formulation challenging for the average user, even when she has an unambiguous information need. Although finding approximate answers to XML queries is an active area of research, it is often assumed that the user will provide some useful and correct structural information to describe the type or level of target XML elements. However, in practice the users of an XML retrieval system, such as the interface to a digital library, have varying levels of experience and knowledge of XML, and many will have difficulties in exploiting rich structural information.

Therefore, we believe that a more practical and user-friendly XML retrieval system would have a keyword-based or natural language interface and would relegate the task of combining textual and structural clues to the retrieval algorithm. As one step towards achieving this goal, we propose an automatic query refinement method to generate a scored list of well-formed XML queries that capture the original information need and conform to the underlying XML data. We formulate query generation as a search problem and this allows solving it with the A* algorithm. We evaluate our method on two XML datasets to demonstrate its effectiveness in integrating query and collection statistics and generating accurate content-and-structure queries.

1. INTRODUCTION

The functionality to mark up text with user-defined, self-descriptive tags makes XML a versatile framework for storing, processing and sharing data. The markup indicates structure and XML documents can be represented as trees or graphs of XML elements, where the structure complements the content and provides additional context, characterization and semantics.

The expressiveness of XML comes at the price of complexity, both in terms of query formulation and query evaluation. The former issue is important because it influences how easily and successfully users interact with an XML search engine. Formal XML query languages such as XPath and XQuery have precise, non-intuitive syntax and offer less straightforward (from user point of view) means to express an information need than a natural language or keyword query. But even if the user is knowledgeable about XML, she also has to be familiar with the schema underlying the XML repository in order to write effective queries.

Therefore, we argue that it is desirable to keep XML query interfaces keyword-based and leave the task of dealing with the complexity of XML structure to the retrieval system. This requires an automatic method for extracting structural clues from the query and integrating them with structural information from the XML repository in order to infer what the user is looking for.

In this work, we develop a query refinement technique that generates content-and-structure queries from plain-text queries. Our algorithm first uses the relationship between structural elements and their content to add structural constraints to each query term, independently of other terms, thus creating a set of *targets*. The algorithm then transforms the target components iteratively and combines them into a single target, which can be directly written as a formal XPath query. For most inputs, there are different ways of combining the same set of targets into a singleton consistent with the XML data. Therefore, the algorithm uses information gain to score individual transformations and thus compute a final score for each generated structured query. The score can be interpreted as a measure of the degree to which a query represents the original information need.

Our technical contributions include

- Defining query generation as a search problem where the goal is to find the highest scoring structured query.
- Developing transformation operators to generate successors in the search space of query targets.
- Introducing the metric information gain to measure the probability of query transformations.

Our technique performs automatic rewriting of keywords queries, a structural version of query expansion with related content terms [15]. It has two main advantages: the user is not required to have any special knowledge of XML nor to specify appropriate structural restrictions of the retrieved XML fragments. Automatically generated struc-

CIKM '08 Napa Valley, CA USA

tured queries can be used either as hard constraints for pruning results, or in a probabilistic, relevance-based framework – as hints for improving ranking.

We performed experiments to evaluate the accuracy of automatically generated content-and-structure queries as well as their effectiveness for ranked XML retrieval. The results show that the method can improve precision because it does not rely on the users to be experts at exploiting XML structure. The method is entirely data-driven and does not use heuristic rules for how the information need is specified. It is also applicable when the XML collection is not homogeneous and consistent with an existing schema.

2. BACKGROUND AND RELATED WORK

Although structure is integral to how information is encoded in XML documents, there is evidence that users are not familiar enough with the structural aspect of XML data and are not able to take advantage of it. Trotman *et al.* [17] report that even experienced users do a poor job at giving structural hints, as shown by comparing the performance of an XML system across two retrieval scenarios on the same topic, one with structural constraints and the other without. Trotman *et al.* conclude that helpful structural hints are a function of the collection and not the query.

To assist the user in exploring an XML collection and reduce the complexity of query formulation, automatic methods for visualizing XML schemas have been developed [6, 19]. There have also been attempts to develop extensions for integrating user preferences as ranking functions via soft constraints [10], or to define simpler, more intuitive XML query languages [2].

Even if an XML retrieval system provides support for query formulation, interaction or data exploration through the use of visualization techniques such as drop-down menus could be time-intensive and therefore impractical. Many users are familiar with performing simple keyword search due to the popularity of web search engines, and might prefer to use a structure-free query interface even when searching a semi-structured database such as a digital library.

However, the complexity and ambiguity of natural language makes the task of ranking answers in the context of keyword queries extremely challenging. For example, Li *et al.* [12] have created a generic interactive query interface for translating free-text queries into XQuery. It uses the output of a dependency parser to map the proximity of word tokens into semantic relationships but the technique relies heavily on heuristic rules.

Another heuristic framework for structure-free XML retrieval is based on the concept of a *lowest common ancestor* (LCA). By definition, an LCA elements for a given query is the root of an XML subtree which contains all keywords but has no descendans which also contain the query terms. In this case, the descendant is assumed to be a better, more specific answer. Many LCA variations have been proposed [20, 13, 11] but they are designed to find XML fragments that certain conditions of non-redundancy rather than a particular query. Therefore the result of LCA-based XML retrieval is not a ranked list of answers but a set of XML elements that all satisfy the heuristic condition equally well though possibly they satisfy different information needs. An alternative solution for supporting keyword search for XML retrieval is to expand a keyword query with structural constraints, and then retrieve answers for the expanded query.

For example, Hsu *et al.* [8] propose a method which adds structural constraints derived from the *context* of a keyword query, where context is defined as the set of XML elements which contain all input query terms. Path expressions derived from the context set are assumed to capture the contextual meaning of a keyword query. To improve accuracy, paths are weighted according to their relevance estimated as a function of node proximity and number of descendants. However, this technique does not consider structure and term distribution statistics that can be automatically gathered from the XML corpus.

Most similar to our work is the top-k keyword search over a relational database approach proposed by Hristidis *et al.* [7]. Since the search is performed over structured data, this requires converting the query into a formal SQL expression. Hristidis *et al.* propose a rank-aggregation algorithm that generates trees of joined relations which contain all query terms and assigns them relevance scores as a combination of TF×IDF-like scores estimated for individual text values. The first step is to run the query against all relations in the database to find a set of tuples that partially match the query. These are joined using the database schema to generate parametrized SQL queries, which are then evaluated by a DBMS with support for text-search functionality to retrieve top-k results.

3. XML QUERY REFINEMENT

We define the process of XML query refinement as automatic generation of content-and-structure queries from a given content-only query. We assume that the user has provided sufficient description of her information need in the form of a keyword query. Our goal is to construct XML queries that the user might have written if she knew XML, the schema of the data and an XML query language. To achieve this, our algorithm uses the measure of information gain to construct queries that are likely structured formulations of the input query given the available collection information. Here 'collection information' is broadly defined to include content information such as statistics of term occurrences and co-occurrences, as well as structure information such as a schema summary or a thesaurus for the names of XML elements and their attributes.

We use the following notation in the rest of the paper. We use the term *target component* or simply *target* to describe a piece of query information. There are two types of target components – bound and unbound. An unbound target, written as //a, describes an XML element of type `a`, i.e. a segment enclosed between an opening tag <a> and a closing tag . A bound target, written as //a[∼'t'], describes an XML element of type `a` that (approximately) satisfies the condition given in the square brackets. The filtering condition ∼'t' says that the content of `a` is topically relevant to 't'. The semantics of ∼ is similar to that of the `about` filter in the NEXI language [18]. Two targets can also be linked together by designating one target as an additional condition on the other. For example, //a//b adds a restriction on the ancestors of target //b, while //a[.//b] adds a restriction on the descendants of //a.

The generation process proceeds as follows:

1. Break up the query into units (terms or phrases of related terms) since keywords are not necessarily intended to appear all in the same element. For each

unit, find what types of elements are likely to contain it, and create one target component for each possible binding of content term and structural element.

2. Generate initial target sets as combinations of input target components. For most inputs, multiple initial sets are created because of the uncertainty to what type elements the user is actually interested in.

3. Generate queries by modifying a target component or combing two components until a target set is reduced to a singleton. A single target describes the set of relevant elements by specifying the type of elements to retrieve as well as structural or content conditions.

In this XML query generation framework, a query is not treated as a single semantic unit. Instead, the algorithm divides the intput into targets and searches for the "best" way to combine them. A sequence of transformations is applied to reach of the goal of a single target component, and each transformation is assigned a score based on the metric information gain. The scores of consecutive transformation are multiplied to compute a final score for the generated query, which can be interpreted as the probability that it expresses the original information need.

We point out that query refinement is a bottom-up process that can be considered the reverse of query relaxation framework for XML retrieval. Relaxation is designed to modify user queries when they are too restrictive and none or only a few tuples are retrieved initially. The method can be applied to rank elements in semi-structured top-k retrieval with a relevance `about` operator, as a way of finding approximate answers to a query by finding exact answers to approximate queries.

For example, Amer-Yahia *et al.* [1] propose a query relaxation method where the initial structured query is modified by relaxing structural constraints – e.g. by removing leaf nodes, converting a child to a descendant or moving a subtree a level higher in the query tree. This results in a set of modified queries that are similar to the original query Q_o to a different degree. The relaxed queries are assigned scores based a modified TF×IDF computation. Elements which satisfy at least one relaxed query are retrieved and ranked according to how "close" they are to the original – the more Q_o is relaxed in order to retrieve an element x, the more uncertain it is that x satisfies the user's information need.

Relaxation is a top-down process: it starts with an XPath query that contains both textual and structural constraints and then iteratively removes or relaxes some of the constraints. In terms of the query as a representation of the information need this process makes the query more ambiguous. Relaxation also requires the user to formulate structured queries in the first place, and then assumes that the submitted query is a prefect representation of the need, since answers to the Q_o are always retrieved first.

On the other hand, query refinement starts with a plain-text query (the ultimate relaxed query) and expand it with structural hints. Following the example of Amer-Yahia *et al.* [1], in the following section we define a set of transformation operators, adapt widely-used IR measures and techniques to assign scores to transformations, and we propose an algorithm that modifies intermediate results to generate well-formed XML queries.

4. SEARCHING FOR THE BEST QUERY

Before we describe the query refinement algorithm in details, we outline two important assumptions that the method is built upon. These are

1. *Keyword query non-ambiguity.*
2. *Availability of XML thesaurus.*

Assumption 1 implies that the user query contains sufficient specification of the information need. The concept of ambiguity itself is not clearly defined and is hard to quantify [4]. For XML data, where the semantics of documents is captured by both structure and content, we assume that the user is interested in one type of XML elements and that she has provided some structural clues to indicate that.

Assumption 2 implies the existence of prior information about the names of XML elements and their attributes. For example, in our experiments we use a list of high-precision synonyms for each unique tag which we have created based on our knowledge of the XML collection. In general, an XML schema defines only a small number of entities (very small compared to the number of documents instantiated according to the schema), so we consider the manual construction of such a thesaurus to be a feasible task.

4.1 Initialization

First, the algorithm breaks up the input query into terms that refer to structure (XML tags or attributes) and terms that refer to text (content terms). We also apply the method described by Jones *et al.* in [9] to identify consecutive content terms with high point-wise mutual information and group them into phrases. Stop words are ignored. For example, in the query "papers by jennifer widom", the term 'paper' refers to an XML element that represents a scientific article (not necessarily tagged as `paper`); 'jennifer widom' is a phrase which refers to the content of the element; and 'by' is a stop word.

Using the above terminology, a node label is an unbound target, and a content term binds a target. Since most terms appear in different types of elements, we create one bound target for each possible binding. Let say 'widom' can occur in the `author` or `editor` field; then we would create two bound targets //author[∼'widom'] and //editor[∼'widom']. We score the variants based on the probability of observing the term in a given XML element, according to unigram language models (not smoothed with the collection frequencies) for elements of each type.

Query term classification (as either a tag or a content term) and target construction are the basis of initialization. To continue the example "papers by jennifer widom", we present the result of its initialization in Table 1. Note that the algorithm has "translated" the term 'paper' into 'article'. The functionality to recognize structural terms such as tags assumes the availability of prior information provided by the creator of the database (Assumption 2).

After each query term has been converted into a target (or several alternative targets), we combine the targets for the entire query into a target set by multiplying their probabilities (Table 1). Thus one target expresses a piece of relevant information, and a target set expresses all information provided by the user query. A target set is not a well-formed XML query but next we define several operators that will allows us to transform target sets into XML queries compliant with the data.

papers	{//article}	0.5000	{//article} {//author[∼'jennifer widom']}	0.3421
	{//inproceedings}	0.5000	{//inproceedings} {//author[∼'jennifer widom']}	0.3421
			{//inproceedings} {//editor[∼'jennifer widom']}	0.1577
jennifer	{//author[∼'jennifer widom']}	0.6842	{//article} {//editor[∼'jennifer widom']}	0.1577
widom	{//editor[∼'jennifer widom']}	0.3154	{//inproceedings} {//title[∼'jennifer widom']}	0.0002
	{//title[∼'jennifer widom']}	0.0004	{//article} {//title[∼'jennifer widom']}	0.0002

Table 1: Initial targets (left) and target sets (right) for the input query "papers by jennifer widom".

4.2 Transformation Operators

Next, the query rewriting process recursively transforms target sets in order to build structured queries. The algorithm applies a series of operators to modify intermediate target sets until they contain a single target. The transformation operators are defined and applied so that a simpleton corresponds to a well-formed XML query. Some target sets cannot be reduced to a single target but our goal is to find the top k automatic queries rather than a complete set of possible queries.

We define the following operators for transforming and combining targets:

- **aggregation**: merges two targets with the same tag, combines any filtering conditions
  {//a},{//a[∼'x']} ↦ {//a[∼'x']}
  {//a[∼'x']},{//a[∼'y']} ↦ {//a[∼'x y']}
- **prefix expansion**: adds an ancestor condition
  {//b} ↦ {//a//b}
  {//b[∼'x']} ↦ {//a//b[∼'x']}
  where `a` is an ancestor of `b` according to the observed collection structure
- **ordering**: combines two targets by designating one as a restriction on the other
  {//a},{//b} ↦ {//a//b} or {//a[.//b]}
  {//a},{//b[∼'x']} ↦ {//a//b[∼'x']} or {//a[.//b[∼'x']]}
  where `a` can be an ancestor of `b` according to the observed collection structure

The difference between specification and ordering is that with specification, we take a target u from the current set and create an unbound target v based on the schema to specify it; with ordering, we take two targets u and v from the current set, bound or not, and join them together.

As an example of ordering transformation, let us order the targets //article and //author[∼'jennifer widom']. There are two possible orderings, each asks for different type of XML fragments to be retrieved and therefore expresses very different information needs:

- //article//author[∼'jennifer widom'] – This target says that a relevant element is an `author` that satisfies two conditions: 1) its content is related to 'jennifer widom' and 2) it has an `article` ancestor.
- //article[.//author[∼'jennifer widom']] – This target says that a relevant element is an `article` that satisfies one condition: 1) it has a descendant `author` related to 'jennifer widom'.

To complete the definition, we explain how to compute transformation scores.

Formally, a target u is defined by its schema name (required) $u.tag()$ as well as restrictions (optional) on its content $u.text()$ and its structure $u.path()$.

We take care to observe target compatibility to guarantee that the generated queries respect the organization of the XML data; we write the condition of compatibility as $\mathcal{C}(u.path(), v.path())$. This implies that when joining two targets any constraints on their structure or content are integrated in compliance with the data; we write the result as $\mathcal{I}(u.text(), v.text())$ and $\mathcal{I}(u.path(), v.path())$ for textual and structural restrictions respectively.

Applying and scoring aggregations

Let u and v be two targets to aggregate. The aggregation operator $\mathcal{A}$ is applicable if $u.tag() = v.tag()$ and $\mathcal{C}(u.path(), v.path())$. Then

$$\begin{aligned}\mathcal{A}(u,v) \quad = \quad \{w : & \, w.tag() = u.tag(),\\ & w.text() = \mathcal{I}(u.text(), v.text()),\\ & w.path() = \mathcal{I}(u.path(), v.path()),\\ & p(w) = p(u)p(v)\}\end{aligned}$$

Applying and scoring prefix expansions

Here we introduce a function d that given targets u and v returns the probability that u is a descendant of v. This probability can be estimated based on the proportion of `u` elements that are descendants of an `v` element in the data.

Now let u be a target to specify and v be an unbound target to specify it. The expansion operator $\mathcal{E}$ is applicable if $\mathcal{C}(u.path(), v)$. Then

$$\begin{aligned}\mathcal{E}(u) \quad = \quad \{w : & \, w.tag() = u.tag(),\\ & w.text() = u.text(),\\ & w.path() = \mathcal{I}(u.path(), v),\\ & p(w) = d(u,v)p(u)\}\end{aligned}$$

Applying and scoring orderings

To score orderings, we use the information theoretic measure information gain (also known as KL-divergence). By definition, given two probability distributions $p(x)$ and $p(x|y)$ for the same discrete random variable X, the information gain of y is

$$g(y) \quad = \quad \sum_{x \in \mathcal{X}} p(x|y) \log \frac{p(x|y)}{p(x)}$$

If x and y are two targets to be ordered – for example //a and //b respectively, then $x|y$ is //a[.//b] and $y|x$ is //a//b where $p(x)$ is a distribution over all XML segments that satisfy the first target, and $p(y)$ is a distribution over all XML segments that satisfy the second target.

Now let u and v be two targets to order. The ordering operator $\mathcal{O}$ is applicable if $\mathcal{C}(u.path(), v)$ or $\mathcal{C}(v.path(), u)$.

Then

$$\begin{aligned}\mathcal{O}(u,v) &= \{w : w.tag() = u.tag(),\\ &\quad w.text() = u.text(),\\ &\quad w.path() = \mathcal{I}(u.path(), v),\\ &\quad p(w) = g(v)p(u)p(v)\}\\ &\cup \{w : w.tag() = v.tag(),\\ &\quad w.text() = v.text(),\\ &\quad w.path() = \mathcal{I}(v.path(), u),\\ &\quad p(w) = g(u)p(u)p(v)\}\end{aligned}$$

where $g(u)$ and $g(v)$ are normalized to sum up to one by dividing by their sum. Normalization is necessary since information gain ranges from 0 to ∞ while we want to interpret scores as the probability that a target set represents the same information need as the input keyword query.

By using information gain we attempt to choose the ordering that is more informative and therefore more plausible. Under the assumption of query non-ambiguity (Assumption 1), the user has provided the system specific information to distinguish XML fragments of interest from a much larger set. To show how information gain works in scoring orderings, let us order //article and //author[∼'jennifer widom'].

First, we consider the ordering //article[.//author[∼'jennifer widom']]. In this case, the discrete space is the set of all `article` elements. Given only the unbounded target //article without any further information about the relevant sets, it is reasonable to assume that the probability distribution is uniform – each article has the same chance of being relevant to the user. If we are also given the description //author[∼'jennifer widom'], then most articles will become irrelevant as their authors do contain neither 'jennifer' nor 'widom'. For the rest the probability of relevance can be estimated in terms of their query likelihood score.

The ordering //article//author[∼'jennifer widom'] has different interpretation. In this case, the discrete space is the set of all `author` elements related to 'jennifer widom'. The original probabilities can be estimated as query likelihood scores, and the conditional distribution assigns probabilities depending on whether an `author` has an ancestor `article` or it does not.

In this example, the information gain of specifying that articles to retrieve are written by Jennifer Widom is higher than the information gain of specifying that authors to retrieve have written an article. If most author elements which contain the phrase 'jennifer widom' are descendants of an article, the second transformation is not very "interesting". But if only a small portion of articles have Jennifer Widom as an author, the first transformation is informative as it indicates that the user is interested in the corresponding small set of articles.

4.3 Query generation

The transformation operators are required to preserve certain compatibility conditions. In the context of XML, compatibility implies the algorithm does not create targets that do not comform to the observed XML data. For example, if there are no `title` elements with a descendant `article`, the target //title[.//article] does not agree with the data. Knowledge of the schema is not required (apart from the XML thesaurus, recall Assumption 2) and thus the process of joining and transforming targets is guided entirely by the data.

Maintaining the consistency of transitional targets guarantees that target sets consisting of a single target can be directly written as well-defined, simplified XPath queries. For a given singleton u, $u.tag()$ specifies the type of XML elements to be retrieved, $u.text()$ describes relevant content, and $u.path()$ recursively specifies restrictions on ancestors and/or descendants.

Therefore we can define query refinement as finding the highest-probability singletons, and implement the generation process as an A* search, where successors are generated by performing applicable transformations. A similar application of A* underlines the WHIRL database management system described by Cohen in [3].

For our particular problem, the general A* algorithm is modified to search for the target sets with highest probability rather than the path with shortest length – i.e. we maximize rather than minimize scores. Maximization implies that an admissible heuristic function must overestimate rather than underestimate the score of any transformation. A straightforward upper bound for each transformation is 1 since the scores are probabilities and therefore lie in the interval [0,1]. The search stops when a target set of size 1 is selected for expansion, or continues until k such simpletons are found. Finally, we have an additional stopping condition since the scores are real numbers that can get very close to zero; therefore we stop the search when the probability of target sets in the open list becomes smaller than a specified lower bound.

Algorithm 1 QUERY-A*-SEARCH(INITIAL,k,ϵ)

Require: Initial target sets INITIAL
Number of queries to generate $k \geq 1$
Minimum query probability $\epsilon \in (0, 1)$

```
 1: OPENED ← INITIAL
 2: ANSWERS ← {}
 3: while OPENED ≠ {} do
 4:    s : s ∈OPENED, f(s) = max f(s')
 5:    OPENED ← OPENED - {s}
 6:    if f(s) < ε then
 7:       Return ANSWERS
 8:    end if
 9:    if s is a goal state then
10:       ANSWERS ← ANSWERS ∪ {s}
11:       if |ANSWERS| = k then
12:          Return ANSWERS
13:       end if
14:    else
15:       OPENED ← OPENED ∪ SUCCESSORS(s)
16:    end if
17: end while
```

5. EVALUATION AND DISCUSSION

We evaluated our query generation algorithm on two XML datasets. The first is the freely available DBLP Records, a collection of almost one million bibliographic entries.[1]; the second is a private collection of resumes owned by the job search company Monster[2], which also contains about a million documents.

Both collections have semantically rich XML structure, so their markup contains useful and discriminative information

[1] `http://dblp.uni-trier.de/xml/`
[2] `http://www.monster.com/`

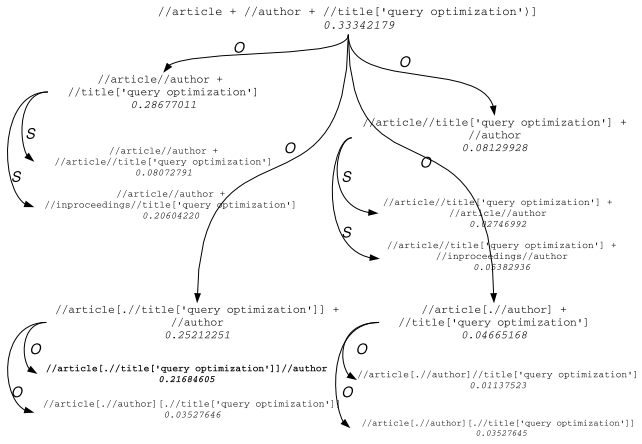

Figure 1: The initial stages of the A* execution for the input query "*people who work on query optimization*" show the consecutive application of ordering and expansion transformations.

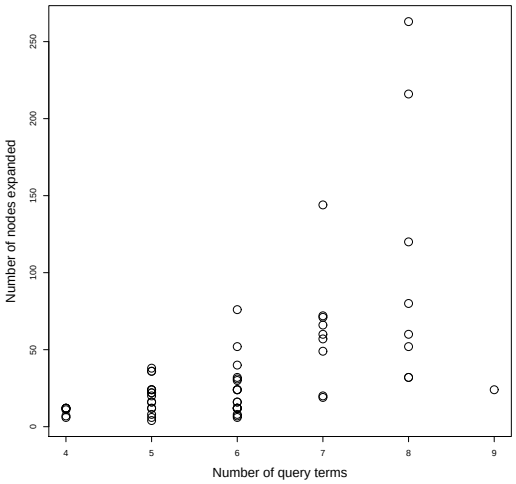

Figure 2: A* search performance on 60 Monster queries in terms of nodes expanded before finding the highest scoring structured query.

and is not merely layout-oriented. In these collections, the tags refer to semantic entities and categories, which is indicated by the fact that XML elements of different types have significantly different term distributions. For example, in DBLP, $p(year)$ is a distribution over integers, $p(author)$ and $p(editor)$ are distributions over proper names, $p(title)$ is a distribution over computer science-related terms. Thus content implicitly characterizes the semantics of the elements. We use simple unigram language models to represent elements but they can be extended with synopses [14] or other advanced schema-aware summaries [5].

5.1 DBLP

We present several example keyword queries and the corresponding automatically generated structured queries in Table 2 and an illustration of how A* is executed to process the query "*people who work on query optimization*" in Figure 1. The input queries contain explicit and implicit structural clues. For example, terms such as 'paper' in Q.1, 'book' in Q.2 and 'people' in Q.3 refer directly to XML elements and thus provide explicit structural information. Content terms provide also some implicit information about the structure of relevant elements. For example, the term '2000' in Q.5 indicates that a particular year or range of years is part of the information need.

When we examine the output of the query refinement process in Table 2, we observe that the highest ranked queries are plausible content-and-structure interpretations of the input query. Although there exist other syntactically correct formulations, the structured queries generated by our algorithm not only adhere to the XPath language specifications, but more importantly, they reflect the organization and the term distributions of the underlying XML documents.

The examples also demonstrate what the notion of query non-ambiguity implies in the context of XML retrieval. We have already stressed that our method assumes the original query is specific, i.e. the user has provided sufficient description of her information need. This assumption is critical since query refinement might not provide an advantage for very short, ambiguous queries when there is not enough information to infer what the user is looking for. For example, given a query which simply says "*jennifer widom*", it is easy to place the phrase in a target of type `author` but it is unlikely that the user is searching for a long list of author elements that all contain the same name.

The benefit of XML query refinement as illustrated by the results in Table 2 is intuitive: the structured queries express a more obvious information need, and could make finding relevant XML fragments easier for a retrieval system that is developed to take advantage of such queries.

Alternative queries can be handled naturally in a probabilistic framework where elements retrieved in response to a generated query with higher probability are given higher score. Structured queries can also be used to organize retrieval results in clusters, by displaying the query in response to which a sub-group of elements is retrieved.

The query refinement process finds the most likely ways to join initial targets given available collection statistics. However, in its current implementation the method does not have any knowledge of natural language. This implies a limitation since the algorithm ignores the order of target components and could create the same set of structured queries for different information needs in cases where the order of query terms carries some semantics. This issue can be addressed by incorporating more sophisticated natural language analysis during intialization (Section 4.1).

5.2 Monster

The second dataset on which we evaluated the accuracy of automatic query refinement is a collection of resumes, with 60 queries provided by the owner of the data, Monster.com. The algorithm successfully generated at least one content-and-structure query for all input queries. The number of nodes expanded by A*, as a measure of the efficiency of the search, is positively correlated with the number of keywords (Figure 2). Examples of automatically generated structured queries for the Monster data are given in Table 3.

#	content-only	content-and-structure queries	scores
Q.1	*papers on query optimization*	//article[.//title[∼'query optimization']]	1.000000
		//inproceedings[.//title[∼'query optimization']]	0.912256
		//article//title[∼'query optimization']	0.243833
		//inproceedings//title[∼'query optimization']	0.087743
		//any[∼'query optimization']//inproceedings	0.013359
Q.2	*books edited by jennifer widom*	//book[.//editor[∼'jennifer widom']]	1.000000
		//book[.//author[∼'jennifer widom']]//editor	0.312400
		//book//editor[∼'jennifer widom']	0.294905
		//book[.//editor[∼'jennifer widom']]//editor	0.225523
		//book[.//editor]//author[∼'jennifer widom']	0.184321
Q.3	*people who work on query optimization*	//article[.//title[∼'query optimization']]//author	1.000000
		//inproceedings[.//title[∼'query optimization']]//author	0.748677
		//inproceedings//author	0.458705
		//inproceedings[.//author][.//title[∼'query optimization']]	0.229194
		//article[.//author][.//title[∼'query optimization']]	0.139917
Q.4	*information retrieval conferences*	//inproceedings[.//title[∼'information retrieval']]//booktitle	1.000000
		//article[.//title[∼'information retrieval']]//journal	0.999613
		//article[.//booktitle]//title[∼'information retrieval']	0.723100
		//article//journal[∼'information retrieval']	0.362296
		//journal[∼'information retrieval']	0.362296
Q.5	*the editors of vldb 2000*	//proceedings[.//booktitle[∼'vldb']][.//year[∼'2000']]//editor	1.000000
		//proceedings[.//booktitle[∼'vldb']][.//editor]//year[∼'2000']	0.075332
		//proceedings[∼'vldb'][.//year[∼'2000']]//editor	0.052991
		//proceedings[∼'vldb']//editor	0.033492
		//proceedings[∼'2000'][.//booktitle[∼'vldb']]//editor	0.025473

Table 2: Highest-scoring automatically generated structured queries for DBLP flatqueries. The scores (which are not comparable across input queries) are normalized so that the top generated query has score of 1.

#	content-only	content-and-structure queries	scores
Q.6	*underwriter mortgage irving texas*	//resume[.//desiredjobtitle[∼'underwriter']][.//company[∼'mortgage']] [.//city[∼'irving']][.//state[∼'texas']]	1.000000
		//resume[.//desiredjobtitle[∼'underwriter']][.//company[∼'mortgage']] [.//location[∼'irving']][.//state[∼'texas']]	0.922259
		//resume[.//title[∼'underwriter']][.//company[∼'mortgage']] [.//city[∼'irving']][.//state[∼'texas']]	0.778866
Q.7	*receptionist microsoft office arizona*	//resume[.//desiredjobtitle[∼'receptionist']][.//skillname[∼'microsoft office']] [.//state[∼'arizona']]	1.000000
		//resume[.//desiredjobtitle[∼'office receptionist']][.//skillname[∼'microsoft']] [.//state[∼'arizona']]	0.691925
		//resume[.//title[∼'receptionist']][.//skillname[∼'microsoft office']] [.//state[∼'arizona']]	0.633407
Q.8	*emergency room registered nurse mesa az with license*	//resume[.//resumetitle[∼'registered nurse']][.//title[∼'emergency room']] [.//educationsummary[∼'license']] [.//city[∼'mesa']][.//stateabbrev[∼'az']]	1.000000
		//resume[.//title[∼'emergency room']][.//resumetitle[∼'registered nurse']] [.//additionalinfo[∼'license']] [.//city[∼'mesa']][.//stateabbrev[∼'az']]	0.939304
Q.9	*sales construction bachelors corona*	//resume[.//desiredjobtitle[∼'sales construction']] [.//educationsummary[∼'bachelor']][.//location[∼'corona']]	1.000000
		//resume[.//desiredjobtitle[∼'construction']][.//title[∼'sales']] [.//educationsummary[∼'bachelor']][.//location[∼'corona']]	0.724707
		//resume[.//desiredjobtitle[∼'sales construction']] [.//educationsummary[∼'bachelor']][.//city[∼'corona']]	0.673541
Q.10	*arabic language translator fluent bilingual los angeles*	//resume[.//skillname[∼'arabic bilingual language']][.//city[∼'los angeles']] [.//additionalinfo[∼'fluent']][.//desiredjobtitle[∼'translate']]	1.000000
		//resume[.//skillname[∼'arabic bilingual fluent language']] [.//desiredjobtitle[∼'translate']][.//city[∼'los angeles']]	0.986548
		//resume[.//skillname[∼'bilingual']][.//educationsummary[∼'arabic language']] [.//desiredjobtitle[∼'translate']][.//additionalinfo[∼'fluent']] [.//city[∼'los angeles']]	0.969456
Q.11	*executive assistant with power point excel torrance ca*	//resume[.//desiredjobtitle[∼'assistant executive']] [.//skillname[∼'excel point power']] [.//city[∼'torrance']][.//stateabbrev[∼'ca']]	1.000000
		//resume[.//desiredjobtitle[∼'executive']][.//title[∼'assistant']] [.//skillname[∼'excel point power']] [.//city[∼'torrance']][.//stateabbrev[∼'ca']]	0.997919

Table 3: Highest-scoring automatically generated structured queries for Monster flat queries.

Query accuracy

To analyze query accuracy, we divided the information contained in the resume queries into three types. Since the service provided by Monster is job searching, the 60 queries express a fairly consistent information need: all ask for resumes and specify a position, a location and/or constraints such as the possession of a certificate or fluency in a particular language.

The first type of information we consider is geographical location: 52 of the 60 test queries contain a city name, a state name or abbreviation, or both. The algorithm is successful at identifying and adding location constraints. In all but 2 queries geographical names are correctly recognized as `city`, `state`, `stateabbrev`, or `location`. One of the incorrectly bound targets is //city[∼'ft'] where 'ft' is an abbreviation for a full-time position. The first error is an example of how the initial processing (Section 4.1), which consists only of stop word removal and simple phrase detection, does not incorporate enough knowledge of natural language.

The second and most common type of input information is the description of a position or a profession. Terms such as 'manager', 'representative', 'engineer', 'assistant', etc. are very common in the collection and there is enough statistics to process this type of information very accurately. Relevant XML elements are `desiredjobtitle` and `resumetitle` (which describe what position the person is looking for) and `title` (which describe positions the person has occupied in the past). The variation in title targets (Table 3) indicates collection heterogeneity. It is mostly due to inconsistency among job seekers on how they choose to fill in their career information. Out of 60 queries, 56 queries contain at least one title-type target.

The third type of information is experience, skills or education requirements. Structural accuracy for this type of information is much lower with 44% correct and 56% incorrect bindings. For example, "*excel power point*" is correctly converted into a bound `skillname` but "*2-5 years of experience*" is split into //yearsexperience[∼'2'] and //completemonth[∼'5']. Since this additional type of information is processed with less accuracy, it could be treated with more uncertainty when computing relevance scores during retrieval – for example, by applying more smoothing.

Retrieval performance

The output of the query generation algorithm we have proposed is a ranked list of structured queries that can be used directly to retrieve XML elements relevant to the user information need.

To evaluate the retrieval effectiveness of the refinement process, we compared the performance of the highest-scoring structured query to that of the original keyword query on a test set of 25 topics. We developed two versions of each topic. One version specifies only a profession (simple information need); the second version adds conditions on qualifications such as years of experience or a degree in particular field (complex information need). We did not include location information because the retrieval system has no notion of geography and the relative distances between locations. When a specified geographical place does not occur in a resume, this renders the resume nonrelevant, even if the person has the required professional qualifications, has willingness to commute and lives within the preferred distance.

rank	simple info need		complex info need	
k	CO	CAS	CO	CAS
1	0.960	0.880	0.600	0.560
2	0.980	0.860	0.580	0.620
3	0.947	0.867	0.520	0.653
4	0.910	0.870	0.480	0.610
5	0.928	0.880	0.464	**0.624**
6	0.913	0.887	0.467	0.607
7	0.909	0.886	0.474	0.583
8	0.905	0.885	0.465	**0.595**
9	0.902	0.876	0.449	**0.582**
10	0.884	0.876	0.456	0.560

Table 4: Precision at rank k for two different formulations of 25 test queries: CO are content-only input queries, CAS are content-and-structure automatically generated queries. Bold indicates statistical significance at level 0.95 with Student's t-test.

To run the queries we used the Indri search engine [16], which is part of the language modeling Lemur Toolkit[3] and supports the NEXI query language. Results for precision at ranks 1 through 10 are presented in Table 4. Automatic structure slightly hurts precision in the case of a simple information need, although the difference is not statistically significant. One query fails almost completely, retrieving only one relevant document in the top 10; the other queries result in three or more relevant documents retrieved. That query is ineffective because it puts an important keyword in a `description` target whose semantics is very general compared to a `title` target.

The claim that highly accurate automatically generated structured queries can benefit retrieval is substantiated in the case of a complex information need, where the refined queries improves precision at the top ranks, with the difference being statistically significant at some ranks. The results shows that by adding structure to the inital keywords, the algorithm creates a more precise expression of a relevant document. This then allows the search algorithm to ignore content matches in non-related fields and perform a more focused retrieval.

6. CONCLUSION

We presented an algorithm for automatically transforming keyword queries submitted to an XML repository into content-and-structure queries. The process does not involve interaction with the user to give structural clues or specify the type of XML elements to return.

Our technique integrates structured data retrieval with probabilistic reasoning based on information gain and relies on analyzing structure and content simultaneously. Given a sufficiently detailed input query, the algorithm exploits statistics derived from the XML collection to infer structural clues and construct probable structured queries. It does not require the collection to be consistent, or to have an explicit schema or DTD. As future work, we plan to implement a topical expansion transformation which adds related terms to the text predicate of target components, as in query expansion.

[3] `http://www.lemurproject.org/lemur/`

Query refinement could provide an alternative to the traditional approach of user interaction with an XML retrieval system, which places much responsibility with the user to formulate good structured queries. Potential applications include systems that work with heterogeneous or dynamic semi-structured collections where variability in the XML documents or the schema renders the task of manually writing successful queries especially challenging for users.

7. ACKNOWLEDGMENTS

This work was supported in part by the Center for Intelligent Information Retrieval, in part by NSF grant #CNS-0454018, and in part by Monster. Any opinions, findings and conclusions or recommendations expressed in this material are the authors' and do not necessarily reflect those of the sponsors.

8. REFERENCES

[1] S. Amer-Yahia, N. Koudas, A. Marian, D. Srivastava, and D. Toman. Structure and content scoring for XML. In *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB)*, 2005.

[2] S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv. XSearch: A semantic search engine for XML. In *VLDB'03: Proceedings of the 29t International Conference on Very Large Data Bases*, 2003.

[3] W. W. Cohen. Data integration using similarity joins and a word-based information representation language. *ACM Transactions on Information Systems (TOIS)*, 18(3), 2000.

[4] S. Cronen-Townsend and W. B. Croft. Quantifying query ambiguity. In *Proceedings of the 2nd International Conference on Human Language Technology Research*, 2002.

[5] J. Freire, J. R. Haritsa, M. Ramanath, P. Roy, and J. Siméon. StatiX: making XML count. In *SIGMOD'02: Proceedings of the 2002 ACM International Conference on Management of Data*, 2002.

[6] R. Goldman and J. Widom. DataGuides: Enabling query formulation and optimization in semistructured databases. In *VLDB'97: Proceedings of the 23rd International Conference on Very Large Data Bases*, 1997.

[7] V. Hristidis, Y. Papakonstantinou, and A. Balmin. Keyword proximity search on XML graphs. In *Proceedings of the 19th International Conference on Data Engineering*, 2003.

[8] W. Hsu, M. L. Lee, and X. Wu. Path-augmented keyword search for XML documents. In *ICTAI'04: Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence*, 2004.

[9] R. Jones, B. Rey, O. Madani, and W. Greiner. Generating query substitutions. In *Proceedings of the 15th international conference on World Wide Web (WWW)*, 2006.

[10] W. Kießling. Foundations of preferences in database systems. In *VLDB'02: Proceedings of the 28thInternational Conference on Very Large Data Bases*, 2002.

[11] G. Li, J. Feng, J. Wang, and L. Zhou. Effective keyword search for valuable LCAs over XML documents. In *CIKM'07: Proceedings of the 16th ACM international conference on Information and knowledge management*, 2007.

[12] Y. Li, H. Yang, and H. Jagadish. Constructing a generic natural language interface for an XML database. In *EDBT'06: Proceedings of the 12 International Conference on Extending Database Technology*, 2006.

[13] Y. Li, C. Yu, and H. V. Jagadish. Schema-free xquery. In *VLDB'04: Proceedings of International Conference on Very Large Data Bases*, 2004.

[14] N. Polyzotis and M. Garofalakis. XSKETCH synopses for XML data graphs. *ACM Transactions on Database Systems (TODS)*, 31(3), 2006.

[15] J. J. Rocchio. Relevance feedback in information retrieval. In G. Salton, editor, *The SMART Retrieval System: Experiments in Automatic Document Processing*, 1971.

[16] T. Strohman, D. Metzler, H. Turtle, and W. B. Croft. Indri: A language model-based search engine for complex queries. Technical report, Department of Computer Science, University of Massachusetts, Amherst, 2005.

[17] A. Trotman and M. Lalmas. Why structural hints in queries do not help XML retrieval. In *SIGIR'06: Proceedings of the 29th International ACM Conference on Research and Development in Information Retrieval*, 2006.

[18] A. Trotman and B. Sigurbjörnsson. *Narrowed Extended XPath I (NEXI)*. Advances in XML Information Retrieval. 2005.

[19] R. van Zwol, J. Baas, H. van Oostendorp, and F. Wiering. Bricks: The building blocks to tackle query formulation in structured document retrieval. In *ECIR'06: Proceedings of the 28th annual European Conference on Information Retrieval*, 2006.

[20] Y. Xu and Y. Papakonstantinou. Efficient keyword search for smallest LCAs in XML databases. In *SIGMOD'05: Proceedings of the 2005 ACM International Conference on Management of Data*, 2005.