Refining Keyword Queries for XML Retrieval by Combining Content and Structure

Desislava Petkova, W. Bruce Croft, and Yanlei Diao

University of Massachusetts, Amherst, MA, USA
{petkova,croft,yanlei}@cs.umass.edu

Abstract. The structural heterogeneity and complexity of XML repositories makes query formulation challenging for users who have little knowledge of XML. To assist its users, an XML retrieval system can have a keyword-based interface, relegating the task of combining textual and structural clues to the retrieval algorithm. In this work, we propose an automatic query refinement method to transform a keyword query into structured XML queries that capture the original information need and conform to the underlying XML data. We formulate query generation as a search problem, and show the effectiveness of the method in generating accurate content-and-structure queries.

1 The problem of adding structure to keyword queries

The functionality to mark up text with user-defined, self-descriptive tags makes XML a versatile framework for storing and sharing data. However, formal XML query languages have precise, non-intuitive syntax and offer less straightforward means to express an information need than a natural language or keyword query.

Although structure is integral to how information is encoded in XML documents, there is evidence that users are not able to take advantage of it. Trotman *et al.* [9] report that even experienced users do a poor job at giving helpful structural hints, and conclude that they are a function of the collection, not the query. On the other hand, many users are familiar with keyword search due to the popularity of web search engines and might prefer to use a structure-free interface even when searching a semi-structured XML database. This motivates interest in the problem of inferring structural constraints from plain-text queries.

Several solutions have been developed for NLP track of the INEX conference. In [8] Tannier presents a system that uses a part-of-speech tagger to create semantic representations of query terms, and corpus-dependent syntactic rules to recognize structural terms. The final query representation is mapped to a general NEXI query, hence the rules are restricted by the expressiveness of NEXI, a simplified version of XPath designed for relevance ranking [10].

The NLPX system by Woodley *et al.* [11] also uses a tagger and lexical knowledge to mark input words as either content, structure or other connotation terms. Tagged queries are matched against templates derived from previous INEX query. However, a fixed set of templates do not guarantee the analysis of

an infinite set of possible queries, and it would have to be continually expanded with new, hand-coded rules.

A different approach by Calado *et al.* [1] assumes simplified structure where objects have a list of attributes and the schema is the set of all attributes. Consequently, input queries are also simplified and contain only content terms rather than both structural and content hints. Under these assumptions, candidate structured queries are generated as possible combinations of query terms and attributes, and ranked according to a Bayesian network model.

To derive structural information from plain-text queries, previous studies [8, 11] incorporate some form of corpus knowledge in the query analysis: DTD, a set of terms associated with the tag names or specific linguistic constructions. Some structural information can be extracted automatically, e.g. in the form of dynamic schemas [4] or statistical summaries [6]. Manual or automatic, such information is necessary to analyze structural hints present in input queries.

Therefore, it might be advantageous to keep XML query interfaces keyword-based and leave the task of dealing with the complexity of XML structure to the retrieval system. In this work, we develop a query refinement technique that generates content-and-structure queries from plain-text queries. Our contributions include defining query generation as a search problem to find the highest scoring structured query, and developing transformation operators to generate successors in the query search space. This implies that the user is not required to have knowledge of XML nor to specify structural restrictions on retrieved XML fragments. Our experiments show that the method can improve precision as it does not rely on the users to be experts at exploiting XML structure.

2 XML query generation as a search problem

XML query refinement is the automatic generation of content-and-structure queries from a given content-only 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. The potential benefit of XML query refinement is that structured queries express a more obvious information need and could make finding relevant XML fragments easier for a retrieval system.

Our algorithm is based on two important assumptions: *keyword query non-ambiguity* and *availability of XML thesaurus*. Assumption 1 implies that the user query contains sufficient specification of the information need. For XML data, where the semantics of documents is captured by both structure and content, we assume that the user is familiar with the domain (not with the exact schema) and that she has provided structural clues to indicate the type of XML elements she is interested in.

Assumption 2 implies the existence of prior information about the tags of XML elements. In our experiments we use a manually created list of domain-specific synonyms for the tag names. For example, for a database of scientific articles, we have the terms *article, publication, paper, poster* as equivalent to an `article` or `inproceedings` tag. The construction of such thesaurus is

domain-specific and not trivial. However, formulating structured queries requires knowledge of the structure and the motivation behind query refinement is to put the responsibility of having this knowledge with the system rather than the user.

We also introduce some notation. We use the term *target* to represent a piece of query information. An unbound target //a describes an XML element of type `a`. A bound target //a[∼'t'] describes an XML element of type `a` satisfying some constraint, e.g. ∼'t' says that the content of `a` is topically relevant to 't'. 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.

Transforming query targets First, the algorithm splits the input query into structure and content terms, based on whether a keyword appears in the structure thesaurus or not. We also apply the method described by Jones *et al.* in [5] to identify consecutive terms with high pointwise mutual information and group them. Stop words are ignored.

In our notation, a node label is an unbound target and a content term binds a target. Since most terms appear in various types of elements, we create one bound target for each possible binding, and score the binding //a[∼'x'] based on the probability that 'x' occurs in the text of an element of type `a`. The probabilities are estimated from unigram language models for each type.

After each keyword has been converted into possible targets, we generate combinations of targets to represent the entire query by multiplying their probabilities. To reduce the exponential complexity, only the highest-probability combinations are processed further. The target sets are then recursively transformed until they contain a single target, which corresponds to a formal XML query. We define three operators for transforming and combining targets.

An *aggregation* merges two targets with the same tag, combines any filtering conditions: {//a} + {//a[∼'x']} ↦ {//a[∼'x']}; {//a[∼'x']} + {//a[∼'y']} ↦ {//a[∼'x y']}. The aggregation operator $\mathcal{A}$ is applicable if targets u and v have the same tag and their structure constraints are compatible. This property is defined with respect to the observed structure. For example, //a//b[∼'x'] and //b[∼'y'] are compatible but //a//b[∼'x'] and //c//b[∼'y'] are compatible only if either //a//c//b or //c//a//b is a valid ancestor-descendant relationship. Compatibility guarantees that the generated queries respect the organization of the XML data. The score of an aggregation is $\mathcal{S}(\mathcal{A}(u,v)) = \mathcal{S}(u)\mathcal{S}(v)$.

A *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. Let u be a target and V be the set of possible ancestors of u. The expansion operator $\mathcal{E}$ creates a new target for each $v \in V$ with v as a structural constraint on u. The score is $\mathcal{S}(\mathcal{E}(u,v)) = \mathcal{D}(u,v)\mathcal{S}(u)$, where the function $\mathcal{D}$ returns the probability that u is a descendant of v, which is estimated based on the proportion of `u` elements that are descendants of a `v` element in the data.

An *ordering* combines two targets by designating one as a restriction on the other: {//a} + {//b} ↦ {//a//b,//a[.//b]}, {//a} + {//b[∼'x']} ↦ {//a//b[∼'x'], //a[.//b[∼'x']]}. We score orderings based on their information gain. By defini-

tion, given two distributions $p(x)$ and $p(x|y)$ for the random variable X, the information gain of y is, $\mathcal{IG}(y) = \sum_{x \in \mathcal{X}} p(x|y) \log \frac{p(x|y)}{p(x)}$. If x and y are two targets to be ordered, say //a and //b, then $x|y$ is //a[.//b] and $y|x$ is //a//b.

Now let u and v be two targets to order. The ordering operator $\mathcal{O}$ is applicable if u and v are compatible, i.e. v is a possible ancestor of u. There are two possible targets: one where u is a descendant constraint on v with score $\mathcal{IG}(u)\mathcal{S}(u)\mathcal{S}(v)$, and another where v is an ancestor constraint on u with score $\mathcal{IG}(v)\mathcal{S}(u)\mathcal{S}(v)$. Since our goal is to choose the ordering that is more informative and therefore more plausible, we normalize $\mathcal{IG}(u)$ and $\mathcal{IG}(v)$ to sum up to one by dividing by their sum. This also guarantees that 1 is an upper bound, so we can interpret scores as the probability of a target set as a representation of the input keywords.

Query generation The transformation operators preserve compatibility. In the context of XML, a compatible transformation results in a target that agrees with the observed XML data. Explicit knowledge of the structure such as a DTD could be incorporated but is not necessary as the compatibility can be tested directly by sending a query to the database. Thus the process of transforming targets is guided entirely by the data. Maintaining target consistency guarantees that a target set consisting of a single target can be directly written as a formal XPath query. For a singleton, its tag specifies the type of XML elements to retrieve, any binding terms describe relevant content, and any links to ancestor or descendant targets recursively specify structural restrictions.

Therefore, we define query refinement as finding the highest-score singletons and implement the process as an A* search, where the successor function creates a new target set for each applicable transformation. A similar application of A* underlines the WHIRL database management system described by Cohen in [2]. 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. 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 singletons are found.

3 Evaluation and discussion

We evaluated our query generation algorithm on two XML datasets. The first is the freely available DBLP Records; the second is a private collection of resumes owned by the job search company Monster. Both collections have semantically rich XML structure where tags refer to categories, so their markup contains useful and discriminative information. The DBLP collection is homogeneous and terms tend to occur in a specific context. The Monster collection is more heterogeneous as the data is created by different people and there is variability in which field they have decided to enter a particular piece of information. For both collections, we use unigram language models to represent elements but they can be extended with synopses [6] or other advanced schema-aware summaries [3].

Table 1. Highest-scoring automatically structured queries for DBLP flat queries. The scores are normalized so that the top generated query has score of 1. They are not predictions for retrieval performance but express the likelihood of the structured queries given the keyword query and the XML collection.

#	content-only	content-and-structure queries	scores
	papers on	//article[.//title[∼'query optimization']]	1.000000
	query	//inproceedings[.//title[∼'query optimization']]	0.912256
Q1	*optimization*	//article//title[∼'query optimization']	0.243833
		//inproceedings//title[∼'query optimization']	0.087743
	books	//book[.//editor[∼'jennifer widom']]	1.000000
	edited by	//book[.//author[∼'jennifer widom']]//editor	0.312400
Q2	*jennifer*	//book//editor[∼'jennifer widom']	0.294905
	widom	//book[.//editor[∼'jennifer widom']]//editor	0.225523
	people	//article[.//title[∼'query optimization']]//author	1.000000
	who write about	//inproceedings[.//title[∼'query optimization']]//author	0.748677
Q3	*query*	//inproceedings[.//author][.//title[∼'query optimization']]	0.229194
	optimization	//article[.//author][.//title[∼'query optimization']]	0.139917
	the editors of	//proceedings[.//booktitle[∼'vldb']][.//year[∼'2000']]//editor	1.000000
	vldb	//proceedings[.//booktitle[∼'vldb']][.//editor]//year[∼'2000']	0.075332
Q4	*2000*	//proceedings[∼'vldb'][.//year[∼'2000']]//editor	0.052991

We present four example keyword queries and the corresponding automatically generated structured queries in Table 1. The input queries contain explicit and implicit structural clues. For example, terms such as 'book' in Q2 and 'people' in Q3 refer directly to XML elements. Content terms can also provide implicit information about the structure of relevant elements. For example, the term '2000' in Q4 indicates that a particular year is part of the information need.

In Table 1, 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 reflect the organization and term distribution of the underlying XML documents. The automatically structured queries can then be submitted to an XML retrieval system to find and rank answers that satisfy the user's information need.

The examples demonstrate how in the context of XML retrieval non-ambiguity implies that the keyword query contains hints to infer both content and structural constraints on the set of relevant elements. It follows from Assumption 1 that query refinement would not provide an advantage for short ambiguous queries. For example, given the query "*jennifer widom*", it is easy to place the phrase in an `author` target but it is unlikely that the user is searching for a list of author elements with the same name. For such an ambiguous query content-only retrieval could achieve similar performance because a structural constraint that associates a keyword with its most frequent structural context would not add much information. However, structural hints in the input query would pose a difficulty for CO retrieval. Even for a homogeneous collection such as DBLP, structural clues indicate information that is not contained in the text, in particular, what type of elements the user is looking for.

The second dataset on which we evaluated the performance of automatic query refinement is a collection of resumes, with 60 queries provided by the owner of the data, Monster.com. These queries reflect an actual information need unlike

Table 2. Highest-scoring automatic structured queries for Monster flat queries.

#	content-only	content-and-structure queries	scores
Q5	*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
Q6	*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
Q7	*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
Q8	*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

the "artificial" queries we came up with for the DBLP data, so our evaluation focuses on the Monster collection. The algorithm successfully generated at least one content-and-structure query for all input queries. Examples of automatically structured queries for the Monster data are given in Table 2.

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 correctly recognizes 50 geographical names and successfully adds location constraints. One incorrectly bound target is for the keyword 'ft' which is an abbreviation for 'full-time'. This error demonstrates that the initial processing, which in our experiments consists only of stop word removal and simple phrase detection, could benefit from incorporating domain knowledge.

The most common type of input information is the description of a position or a profession. 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). Terms such as 'manager', 'engineer', etc. occur frequently, so there are enough statistics to process this type of information very accurately, with 56 out of the 60 queries containing at least one title-type target. The variation in title targets (Table 2) is evidence for the collection heterogeneity, which we attribute to inconsistency among job seekers on how they fill in their career details.

Table 3. Precision at rank k for two versions of 25 Monster queries. CO are content-only input queries, CAS are content-and-structure automatically generated queries. Bold indicates statistical significance at 5% level with Student's t-test.

rank	simple information need		complex information need	
	CO	CAS	CO	CAS
1	0.960	0.880 (-8.33%)	0.600	0.560 (-6.67%)
2	0.980	0.860 (-12.2%)	0.580	0.620 (+6.90%)
3	0.947	0.867 (-8.45%)	0.520	0.653 (+25.6%)
4	0.910	0.870 (-4.40%)	0.480	0.610 (+27.1%)
5	0.928	0.880 (-5.17%)	0.464	**0.624** (+34.5%)
6	0.913	0.887 (-2.85%)	0.467	0.607 (+29.9%)
7	0.909	0.886 (-2.53%)	0.474	0.583 (+23.0%)
8	0.905	0.885 (-2.21%)	0.465	**0.595** (+27.9%)
9	0.902	0.876 (-2.88%)	0.449	**0.582** (+29.6%)
10	0.884	0.876 (-0.90%)	0.456	0.560 (+22.8%)

The third type of information is experience, skill or education requirements. Here structural accuracy is 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']. The reduced accuracy is perhaps due to higher heterogeneity in this type of information, with occurrences in various distinct element types. To handle this, the algorithm could assign more uniform scores to initial bindings instead of preferring the most frequent occurrences, effectively creating more diverse set of target sets to reflect the diversity in the data.

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.

To run the queries we used the Indri search engine [7], which is part of the language modeling Lemur Toolkit and supports the NEXI query language. Results for precision at ranks 1 through 10 are presented in Table 3. Automatic structure slightly hurts precision in the case of a simple information need, although the difference is not statistically significant. On the other hand, the results support our hypothesis that automatically generated structured queries can benefit retrieval in the case of a complex information need, where the refined queries improves precision at the top ranks, with an increase of more than 20% for ranks 2 through 10. Therefore, for queries of higher complexity, there is opportunity to create a

more precise expression of a relevant document by automatically structuring the initial keywords into a formal query. This allows the search algorithm to ignore content matches in non-related fields and perform a more focused retrieval.

4 Conclusion

We presented an algorithm for automatically transforming keyword queries to an XML repository into content-and-structure queries. The process does not require the user to write formal XML queries or even to 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 keyword query as input, the algorithm exploits statistics derived from the XML collection to infer structural clues and construct probable structured queries. It does not assume that the collection is homogeneous and has an explicit schema or DTD but instead uses knowledge about the collection in the form of natural language equivalents of the tag names.

Query refinement could provide an alternative to the traditional approach of user interaction with an XML retrieval system, which places 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.

5 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.

References

1. P. Calado and A. da Silva and R. Vieira and A. Laender, and B. Ribeiro-Neto. Searching web databases by structuring keyword-based queries. In Proceedings of CIKM, 26–33 (2002)
2. W. W. Cohen. Data integration using similarity joins and a word-based information representation language. ACM Trans. Inf. Syst. 18, 288–321 (2000)
3. J. Freire, J. R. Haritsa, M. Ramanath, P. Roy, and J. Siméon. StatiX: making XML count. In Proceedings SIGMOD, 181–191 (2002)
4. R. Goldman and J. Widom. DataGuides: Enabling query formulation and optimization in semistructured databases. In Proceedings of VLDB, 436–445 (1997)
5. R. Jones, B. Rey, O. Madani, and W. Greiner. Generating query substitutions. In Proceedings of WWW, 387–396 (2006)
6. N. Polyzotis and M. Garofalakis. XSKETCH synopses for XML data graphs. ACM Trans. Database Syst. 31, 1014–1063 (2006)
7. T. Strohman, D. Metzler, H. Turtle, and W. B. Croft. Indri: A language model-based search engine for complex queries. CIIR tech. report #416, University of Massachusetts, Amherst (2005)
8. X. Tannier. From Natural Language to NEXI, an Interface for INEX 2005 Queries. In Proceedings of INEX, 373–387 (2005)
9. A. Trotman and M. Lalmas. Why structural hints in queries do not help XML retrieval. In Proceedings of SIGIR, 711–712 (2006)
10. A. Trotman and B. Sigurbjörnsson. Narrowed Extended XPath I (NEXI). Advances in XML Information Retrieval, 16–40 (2005)
11. A. Woodley and S. Geva. NLPX at INEX 2005. In Proceedings of INEX, 358–372 (2005)