Personalizing XML Search in PIMENTO

Sihem Amer-Yahia
Yahoo! Research
USA
sihem@yahoo-inc.com

Irini Fundulaki
University of Edinburgh
UK
Irini.Fundulaki@ed.ac.uk

Laks V.S. Lakshmanan
University of British Columbia
Canada
laks@cs.ubc.ca

Abstract

XML search is increasing in popularity as more and larger XML repositories are becoming available. The accuracy of XML search varies across different systems and a lot of effort is put into designing scoring functions tailored to specific users and datasets. We argue that there is no one scoring function that fits all and advocate incorporating user profiles into XML search to personalize query answers by accounting for user profiles.

First, we propose a framework for defining user profiles and for enforcing them during query processing. Second, we adapt the well-known topk pruning to account for user profiles. Finally, we present effectiveness and efficiency experiments which show that query personalization in XML search dramatically improves the accuracy of query results while incurring negligible processing overhead. This work is in the context of the PIMENTO *project which aims at improving the relevance of searching structured and unstructured content.*

1 Introduction

XML search is becoming widely popular due to the increasing amount of XML data and to a growing interest in designing appropriate scoring methods and ranking algorithms. The accuracy of XML search varies across systems and a lot of effort is put into designing scoring functions tailored to specific datasets. E.g., the INEX effort [11] aims at improving the search relevance of IEEE XML data collections. However, none of the existing XML search solutions leverages *user information* to determine relevant query answers. As an example, a painter who searches for "black paint" would receive the same results as a home builder. Similarly, a user looking for a used car would receive the same listing regardless from his car preferences (color, make, mileage). In this paper, we argue that there is no one scoring function that fits all and advocate the idea of incorporating *user profiles* into XML search in order to customize query answers and improve search quality. To the best of our knowledge, our work is the first attempt to apply query personalization to XML search. In this paper, we take the first necessary steps to achieve this goal by modeling user profiles and enforcing them efficiently and effectively in XML search.

Personalization is used in applications such as telecommunications to direct user calls based on the caller context (e.g., physical location, time of day). In Web search, the ranking of query answers may be modified based on the user's navigational behavior during a session. Relational query personalization has been studied extensively [9, 15] and shown to be effective in practice.

Query personalization through user profiles has different aspects that restrict or expand its applicability. Enforcing a user profile ranges from simply modifying the original ranking of query answers, to returning a substantially different set of answers. Consider the example document in Fig. 1. The document describes cars for sale. Information on each car may include the manufacturing date, the owner information, the price, horsepower, make, and color. Other information may be embedded in the car description.

A user interested in buying a car which is in a good condition and which costs less than $2000 can formulate the following XQuery Full-Text [21] query:
Q : //car[./description[*ftcontains*(. , "*good condition*") & *ftcontains*(. , "*low mileage*")] & ./price $<$ 2000]

Answers to such a query need to be ranked by their relevance to "good condition" and to "low mileage". Assuming that the user is located in New York City (NYC) and that he has a preference for red cars, it is natural to expect that regardless from the scoring function used by the underlying query engine, the user should receive red cars for sale in NYC ranked higher than other cars. On the other hand, a user may be willing to buy a car which is located in a different state provided it has a higher horsepower than cars for sale in NYC. Consequently, the process of query personalization may either expand or restrict the original set of query answers and some ranking preferences may be enforced when returning query results. In our formalization, a user profile is composed of two kinds of preference rules

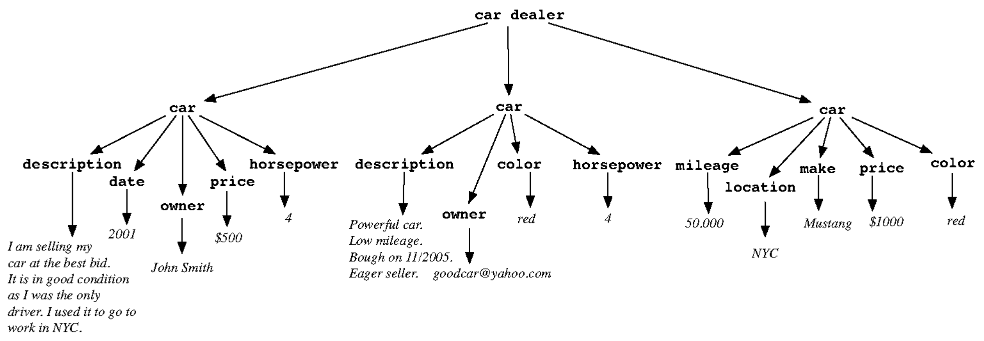


Figure 1. Example of a Car Sale Database

– *scoping rules* (SRs) and *ordering rules* (ORs). SRs are used to expand (e.g., I am willing to drop the low mileage requirement) or restrict (e.g., I only want to see cars for sale located in my area) the original result. ORs are used to enforce ranking preferences (e.g., I prefer red cars or cars with higher horsepower). SRs may be conflicting. For example, one SR may only be applicable to queries specifying a certain condition, while some other SR may remove this condition from the query. The order in which these two SRs are applied to a query may lead to different results. We propose to detect and resolve such conflicting SRs. Similarly, ORs may be ambiguous. For example, a user may prefer cars with a high horsepower and also prefer cars with a low mileage. The database of cars for sale may contain a Honda with a higher horsepower than a Mustang. However, the mustang may have a lower mileage than the Honda. The preference between them is then unclear. We propose to resolve *ambiguous* ORs before enforcing them in query personalization.

Enforcing SRs and ORs is not a straightforward task due to the number of rewritings of the user query that may need to be generated using SRs, and to the possibly inefficient answer re-ranking implied by the application of ORs. Therefore, we leverage existing query relaxation work [3, 19] to enforce SRs efficiently. Ultimately, the user is only interested in the top answers. Consequently, understanding how to combine user profiles with topk processing is a key aspect of efficient query personalization. A key point is, while it is necessary to be able to prune intermediate query results which will not make it to the top best answers [8, 10, 17], the introduction of ORs requires to revisit well-established topk pruning conditions such as the threshold algorithm defined in [10]. Even if their query score is low, user-preferred answers should not be pruned. Therefore, we formalize query processing in an algebra and define OR-aware topk operation that achieves effective pruning while guaranteeing soundness of our query evaluation, i.e., always returns the correct topk answers.

In this paper, we do not address the creation of user profiles. Our contributions are:

1. We formalize user profiles in terms of scoping rules (SRs) and ordering rules (ORs) and define query personalization as the process of rewriting a user query using SRs and ranking query answers using ORs.
2. We describe an algorithm to detect and resolve conflicting SRs and ambiguous ORs.
3. We define OR-aware topk pruning to guarantee efficient query personalization.
4. We run effectiveness experiments on INEX [11] datasets and queries which show that enforcing user profiles achieves good precision and recall, and efficiency experiments on XMark datasets which show that personalization induces negligible processing overhead.

Section 2 provides an overview of existing work in query personalization. Our data model and definition of user profiles are given in Section 3. Section 4 contains a summary of the problems studied in this paper. In Section 5 we discuss the static analysis of preference rules. Section 6 describes our algebra and OR-aware topk operator as well as query evaluation algorithms. Experiments are provided in Section 7. We conclude in Section 8.

2 Related Work

Two approaches have been proposed for the definition of user preferences in the relational world. In the *qualitative* approach [9, 12, 13], preferences are specified as *preference relations*, which are defined by means of either *logical formulas* [9] or preference constructors [12, 13]. Using this approach, a user can specify *relative* preferences of the form: 'I like X better than Y'. In the *quantitative* approach [1, 4, 5, 7, 10, 16, 17] preferences are mostly *absolute* and are specified in terms of *scoring functions* that associate a score with every answer. Koutrika et. al. in [15] extend this approach by associating degrees of interest with preferences, the latter expressed as conjunctions of predicates on relations of interest. In addition to these two approaches, authors in [14] propose a type of preferences in the form of *query rewriting rules* which are used to syntactically restrict the user query by conditionally adding new predicates.

Query personalization in the qualitative and quantitative approaches is achieved by re-ordering query answers. In the case of qualitative preferences, Chomicki [9] proposes the *winnow* operator that is incorporated in the relational algebra to select the most preferred tuples according to the preference formulas specified by the user. In [13] the enforcement of user preferences is taken into account outside the query engine. In the context of quantitative approaches, optimization techniques for *topk queries* [7, 8, 16, 17] and their successors have been developed. Authors in [15] follow a hybrid approach where they use quantitative user preferences to syntactically rewrite the user query, and then use the preferences' degree of interest to re-rank the results.

In our work, we adopt a hybrid approach for capturing user preferences in which we define a *user profile* to be a set of *scoping* and *ordering* rules. To the best of our knowledge, this is the first work to combine qualitative preferences with query rewriting rules that go beyond those proposed in [14]. Rules in [14] are defined in a logic-based language and are of the form $H \leftarrow B$ where H and B denote the head and the body of the rule resp. and the expression in the head of the rule must be replaced with the query expressed in the body of the rule. This language cannot capture the predicates that we support in our work (see Section 3). Furthermore, our scoping rules can *relax*, *tighten* or simply *modify* the query by dropping, adding and replacing respectively potentially complex query predicates. Our ordering rules fall into the category of qualitative preferences, and as in [9, 12] they define a partial order between XML nodes of the same type. In contrast to these works, we are the first to address ambiguity of ordering rules. Furthermore, none of these works proposes efficient topk processing that considers both answer scores and user ordering rules.

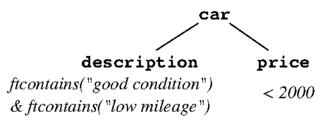
3 Class of Queries and User Profile

The XQuery Full-Text family of languages [6, 21] augment keyword search with two components: (i) full-text predicates such as proximity and order between keywords and (ii) path conditions which narrow the search scope. We abstract the core of such queries using *extended tree pattern queries*. A *tree pattern query* (TPQ) [2] is a pair $Q = (N, E)$, where (N, E) is a rooted tree, with nodes in N labeled by tags, and with $E = E_c \cup E_d$ consisting of *pc*-edges (E_c) and *ad*-edges (E_d). A distinguished node in N corresponds to the answer element. Each edge $e(x, y) \in E$ can be seen as a structural predicate saying y must be a child (resp., descendant) of x. We extend TPQs for querying XML data together with free text: we assume that *each leaf node x in a TPQ Q has a condition associated with it.* The condition may be:

• A conjunction of constraint predicates of the form $value\ relOp\ u$ where $value$ denotes the content of the leaf node x, $relOp$ is one of $=, \neq, >, \leq, <, \geq$, and u is a value from an appropriate domain.

• A conjunction of keyword predicates of the form $ftcontains(\text{“}k\text{”})$, where “$k$” is a keyword, which states that the node x contains an occurrence of the keyword “k” at any document depth.

The former condition expresses hard constraints such as ./price $<$ 2000 while the latter expresses keyword and phrase conditions such as $ftcontains(.\,,\ \text{“}NYC\text{”})$ & $ftcontains(.\,,\ \text{“}low\ mileage\text{”})$. A TPQ thus consists of a conjunction of structural, keyword, and constraint predicates. Fig. 2 shows the extended TPQ for query Q given in the introduction.



Scoping Rules (SRs)

ϱ1: **if** *pc(*`car, description`*) & ftcontains(*`description`,*"low mileage")*
then remove *ftcontains(*`car`, *"good condition")*

ϱ2: **if** *pc(*`car, description`*) & ftcontains(*`description`,*"good condition")*
then add *ftcontains(*`description`, *"american")*

ϱ3: **if** *pc(*`car, description`*) & ftcontains(*`description`,*"good condition")*
then remove *ftcontains(*`description`, *"low mileage")*

Ordering Rules (ORs)

π1: x.`tag`=*car* & y.`tag`=*car* & x.*color*=*red* & y.*color* <>*red* $\longrightarrow x \propto y$

π2: x.`tag`=*car* & y.`tag`=*car* & x.*mileage* < y.*mileage* $\longrightarrow x \propto y$

π3: x.`tag`=*car* & y.`tag`=*car* & x.*make* =y.*make* & x.*hp* >y.*hp* $\longrightarrow x \propto y$

π4: x.`tag`=*car* & y.`tag`=*car* & *ftcontains(x*, *"best bid")* $\longrightarrow x \propto y$

π5: x.`tag`=*car* & y.`tag`=*car* & *ftcontains(x*, *"NYC")* $\longrightarrow x \propto y$

Figure 2. TPQ, SRs and ORs for the running example

We model a user profile using two orthogonal and complementary components. First, we use *Scoping Rules* (SR) to let the user change the scope of her query by broadening/narrowing the search by *relaxing tightening* query predicates. E.g., a parent-child relationship may be relaxed to ancestor-descendant or a leaf node may be dropped or a new predicate added [3, 19]. Examples of SRs are given in Fig. 2.

Second, we use *Ordering Rules* (OR) to specify how to rank answers obtained from the system. This allows the user to override the system's default ranking criteria and express her preferences for answers. Examples of ORs are given in Fig. 2.

3.1 Scoping Rules

There are two kinds of SRs (see Fig. 2). Narrowing the search is accomplished by *add* rules which restrict the user query by adding predicates. Broadening the search is accomplished by either *delete* rules which remove existing query predicates or by *replace* rules which replace existing query predicates by weaker ones. An *add/delete* rule is of the form $\textbf{if}\ (condition)\ \textbf{then}\ (action, conclusion)$ where: *(i)* $condition$ is either a conjunction of structural and constraint predicates, or the value true; *(ii)* $action$ is one of *add* and *delete* and finally *(iii)* $conclusion$ is a conjunction of structural and constraint predicates. We assume the conjunction of structural and constraint predicates forms a connected graph so that after the rule is applied to a TPQ, what results is still a TPQ. A *replace* rule is of the form: $\textbf{if}\ (condition)\ \textbf{then}\ replace\ \mathsf{E}\ \textbf{with}\ \mathsf{E}'$ where as previously, $condition$, E and E' are conjunctions of predicates that form a connected graph.

The intuitive semantics of an *add/delete* rule are the following: if a user query Q subsumes the $condition$ of a rule, then apply the $conclusion$ of that rule to the query as specified by $action$. In the case of *replace* rule, the intuition is the following: if a user query Q subsumes the $condition$ of the rule, then replace E, if present in the query, with E'.

For the subsumption checks, we can use the well-known XPath containment algorithms (e.g., see [2, 18]). In general, the application order of SRs might be significant. E.g., consider rules ρ_1 and ρ_2 in Fig. 2. Both ρ_1 and ρ_2 are applicable to the TPQ in the same figure. Applying ρ_2 first will add the predicate $ftcontains(\mathsf{description}, \text{“}american\text{”})$. Applying ρ_1 to the result removes $ftcontains(\mathsf{description}, \text{“}good\ condition\text{”})$. However, applying ρ_1 first renders ρ_2 inapplicable. Thus, the orders ρ_1, ρ_2 and ρ_2, ρ_1 yield different results. The user profile can associate a priority with SRs, thus forcing a fixed order of rule application, making the semantics of a set of SRs well defined.

3.2 Ordering Rules

ORs are of two kinds: value-based and keyword-based. A *value-based OR (VOR)* specifies that a user might prefer those answers satisfying a specific property to other answers, where the property is the value of an attribute/element. E.g., in doing a car search, the user may prefer red cars to others (illustrated by π_1 in Fig. 2). More generally, the user may have her own *preference relation* on colors, in general, a partial order. π_2 is another example where the user indicates a preference for cars with a lower mileage.

A value-based OR can have one of the following forms:

$$\mathcal{C}\ \&\ x.attr = c\ \&\ y.attr \neq c\ \rightarrow\ x \prec y \quad (1)$$

$$\mathcal{C}\ \&\ x.attr\ relOp\ y.attr\ \rightarrow\ x \prec y \quad (2)$$

$$\mathcal{C}\ \&\ prefRel(x.attr, y.attr)\ \rightarrow\ x \prec y \quad (3)$$

where:
1. $\mathcal{C}$ is a conjunction of conditions on x and y that equate their common properties (we call $\mathcal{C}$ the common conditions) and c is a constant. An example of common condition on x and y in rule π_3 is that x and y are both cars with the same make.
2. $relOp$ is one of the relops $\{<, >\}$ (e.g., $x.mileage < y.mileage$) and
3. $prefRel$ is a binary relation on the domain of $x.attr$ ($y.attr$) which is a strict partial order, e.g., a partial ordering on colors.

We require $relOp$ to be one of $<, >$ since we want $\prec$ to be a strict partial order. E.g., the rule π_3 in Fig. 2 says between cars of the same make, those with higher horsepower are preferred.

The general form of *keyword-based ORs (KOR)* is $\mathcal{C}\ \&\ ftcontains(x, \text{“}k\text{”})\ \rightarrow\ x \prec y$, where $\mathcal{C}$ are the common conditions as before. It says between answers x and y, x is preferred to y provided it contains an occurrence of the keyword k. E.g., the rule π_4 in Fig. 2 says that among all cars, the user prefers those that contain an occurrence of “best bid” while rule π_5 says that among all cars, the user prefers those that contain an occurrence of “NYC”.

3.3 Answer Ranking

Each query answer acquires a score based on its query match. It also gets a KOR score based on any KORs in the user profile. The VORs in the user profile may impose an ordering on the query answers independently of the above two. How are we then to order answers? We consider two possibilities. The order $\mathcal{K}, \mathcal{V}, \mathcal{S}$ indicates that we order answers by their KOR scores first and then based on the VOR preferences. When two answers tie on their KOR score and their VOR properties, we order them by query score. The other order we consider is $\mathcal{V}, \mathcal{K}, \mathcal{S}$.

4 Problems Studied

Our goal is to assist the user in enhancing her query answering experience in searching XML documents. We have proposed two complementary components for configuring an effective user profile: (i) the scoping rules (SRs) and (ii) the value-based and keyword-based ordering rules (ORs).

Given a set of SRs Σ and a query Q, the intended effect of SRs is that rules in Σ should be used to rewrite Q before it is evaluated. At the same time, the user should not be penalized for having configured a profile. If there are few or no answers satisfying the rewritten query, we should still consider answers satisfying the original query. Thus, query answering w.r.t. a set of SRs really entails evaluating a *flock* of related queries. The first problem we study is how to pin down this query flock exactly. This is complicated by the fact that sometimes rules may conflict with each other: applying a rule may render another rule inapplicable. We would like to define the exact semantics of answering a query in the presence of SRs, with an emphasis on *topk answers*.

Second, the value-based ORs may sometimes result in *ambiguity*. Intuitively, this happens whenever there is a pair of answers x, y such that x is preferable to y according to some ORs and y is preferable to x according to some others. We would like to have an efficient algorithm for detecting ambiguity of value-based ORs and a means of resolving it.

Suppose we have a user profile $\Pi = (\Sigma, \mathcal{O}_v, \mathcal{O}_k)$, with scoping rules Σ, value-based ORs $\mathcal{O}_v$ and keyword-based ORs $\mathcal{O}_k$. Let Q be a TPQ. The third problem we study is to develop efficient query evaluation strategies for Q in the presence of Π, with an emphasis on *topk answers*.

5 Static Analysis

The first two problems from the previous section are studied in the next two subsections.

5.1 Scoping Rules and Query Flocks

The main issue that arises in rewriting a query w.r.t. a set of SRs is that one rule's application may render another rule inapplicable. We say that a rule ρ is applicable to a query Q if the condition in ρ is subsumed by Q. In Section 3, we have seen that different order of applying SRs to a query can result in different rewritten queries. A second issue that can arise is that a rule may "conflict" with another. For an SR ρ and a query Q, we denote by $\rho(Q)$ the result of applying ρ to Q. Given a set of SRs and a query Q, we say a rule ρ_1 *conflicts* with ρ_2 w.r.t. Q provided: (i) both ρ_1, ρ_2 are applicable to Q, and (ii) ρ_2 is not applicable to $\rho_1(Q)$. E.g., consider the query and SRs in Fig. 2. Both ρ_1 and ρ_2 are applicable to the query as it subsumes their conditions. ρ_1 conflicts with ρ_2 since ρ_2 is not applicable to the result of applying ρ_1 to the query.

Conflict among SRs can be captured using a directed graph where each node is an SR. There is an arc (ρ_i, ρ_j) iff ρ_i conflicts with ρ_j. If this conflict graph is acyclic, then we can topologically sort the nodes and apply the SRs to a query in the topological sort order.

However, there may be cycles in the conflict graph. E.g., ρ_1 and ρ_3 in Fig. 2 conflict with each other. To mitigate this problem, we require the user to assign priorities to rules. Given that different order of rule application may result in different rewritten queries, we believe it's important for the user to have a say in which order is used. Rule priorities resolve the problem of conflict cycles by forcing a specific order of rule application. We henceforth assume either the set of SRs is conflict-free or that there is a user assigned priority forcing a specific order of rule application.

Given a query Q and a set of SRs Σ, possibly together with rule priorities, the *query flock* associated with Q and Σ consists of the family of queries $Q, \rho_1(Q), \rho_2(\rho_1(Q)), ..., \rho_n(\rho_{n-1}(\cdots(\rho_1(Q)))$, where we assume that the order of rule application imposed by the priorities is $\rho_1, ..., \rho_n$. The idea is that *all* the queries in the query flock must be evaluated and answers ranked according to the ORs.

5.2 Ambiguity of Value-based ORs

Consider the value-based ORs π_1 and π_2 in Fig. 2 expressing preferences among cars. The rules appear quite reasonable. However, consider a pair of cars c, d such that c has color red, but a higher mileage than d. Then according to π_1, $c \prec d$, while according to π_2, $d \prec c$. Thus, there are database instances where the intended preference among elements is not clear. We consider such ORs "ambiguous". More precisely, a set of value-based ORs $\mathcal{O}_v$ is *ambiguous* provided there is a database instance D and a pair of elements $e, f \in D$ such that $D \cup \mathcal{O}_v \models$ is inconsistent. Notice that if $\mathcal{O}_v$ is ambiguous, it does *not* mean $\mathcal{O}_v$ itself is inconsistent. E.g., if we have a database D without any red cars, or a database D where all red cars have a lower mileage than other cars, then $\mathcal{O}_v \cup D$ is indeed consistent. We say $\mathcal{O}_v$ is *unambiguous* if it is not ambiguous. Intuitively, it means no matter which database D we consider, $\mathcal{O}_v \cup D$ is consistent, i.e., the preference among elements in D is precisely defined by $\mathcal{O}_v$.

It is important to detect whether a set of value-based ORs is ambiguous and if so have the user assign priorities to ORs to make them unambiguous. We discuss an algorithm for dealing with ambiguity next. Denote a value-based OR as $\mathsf{local}(x) \,\&\, \mathsf{local}(y) \,\&\, \mathsf{comp}(x, y) \rightarrow x \prec y$, where $\mathsf{local}(x)$ denotes constraints involving only variable x (e.g., $x.\mathsf{tag} = car$, $x.color = red$, etc.) and $\mathsf{comp}(x, y)$ denotes constraints involving both x and y (e.g., $x.mileage < y.mileage$). We rename variables if necessary so different ORs use disjoint sets of variables, to avoid confusion.

Let $\mathsf{local}^\star(x)$ (resp., $\mathsf{comp}^\star(x, y)$) denote the set of constraints involving only x (resp., both x and y) that are implied by $\mathsf{local}(x) \,\&\, \mathsf{local}(y) \,\&\, \mathsf{comp}(x, y)$. E.g., from $x.color = red \,\&\, y.color \neq red \,\&\, y.hp = 200 \,\&\, x.hp < y.hp$, we can infer $\mathsf{local}^\star(x) = x.color = red \,\&\, x.hp < 200$ and $\mathsf{comp}^\star(x, y) = x.hp < y.hp \,\&\, x.color \neq y.color$. Let $\pi_i \equiv \mathsf{local}(x_i) \,\&\, \mathsf{local}(y_i) \,\&\, \mathsf{comp}(x_i, y_i) \rightarrow x_i \prec y_i$, $i = 1, 2$ be a pair of rules. We say that variables y_1 and x_2 are *compatible* provided the conjunction of constraints $\mathsf{local}^\star(y_1) \,\&\, \mathsf{local}^\star(x_2) \,\&\, x_2 = y_1$ is logically consistent. E.g., revisit π_1: $x.tag = car \,\&\, y.tag = car \,\&\, x.color = red \,\&\, y.color \neq red \rightarrow x \prec y$ and π_2: $u.\mathsf{tag} = car \,\&\, v.\mathsf{tag} = car \,\&\, u.mileage < v.mileage \rightarrow u \prec v$. The variables y and u are compatible since $\mathsf{local}^\star(y) \equiv \mathsf{local}(y)$ is $y.\mathsf{tag} = car \,\&\, y.color \neq red$ and $\mathsf{local}^\star(u) \equiv \mathsf{local}(u)$ is $u.\mathsf{tag} = car$. The conjunction $y.\mathsf{tag} = car \,\&\, y.color \neq red \,\&\, u.\mathsf{tag} = car \,\&\, y = u$ is obviously consistent. Define a *constraint graph* $G(\mathcal{O}_v)$ for a set of value-based ORs $\mathcal{O}_v$ as follows. Its nodes are the variables occurring in the rules $\mathcal{O}_v$. Whenever $x \prec y$ appears on the right-hand-side of a rule, G has an arc (x, y) labeled $\prec$. Whenever x, y are variables appearing in different rules and are compatible, G has an undirected edge $\{x, y\}$ labeled $=$. By an *alternating cycle* we mean a cycle of the form $(v_1, ..., v_k, v_1)$, $k \geq 2$, k an even number, such that: (i) the edges (v_i, v_{i+1}) are directed arcs labeled $\prec$ for odd i and (ii) the edges $\{v_i, v_{i+1}\}$ are undirected edges labeled $=$ for even i, and (iii) the edge (v_k, v_1) is undirected and is labeled $=$. We have the following result.

Lemma 5.1 (Ambiguity) : *Let $\mathcal{O}_v$ be a set of value-based ORs and $G(\mathcal{O}_v)$ the associated constraint graph. Then $\mathcal{O}_v$ is ambiguous iff G contains an alternating cycle.*

Using a straightforward adaptation of depth-first search, we can readily detect ambiguity in time $O(\#\text{edges})$. Suppose a set of value-based ORs defined by a user is ambiguous. Then by assigning a priority to the rules, alternating cycles can be broken. E.g., the rules $\{\pi_1, \pi_2\}$ form an ambiguous set. The reader can easily check that y, u are compatible (shown above) and that v, x are compatible as well. This creates an alternating cycle in the constraint graph. It results in ambiguity since if we have a red car with high mileage and a non-red car with low mileage, the intended order is not clear. The alternating cycle and hence the ambiguity can be broken if we assign, e.g., priority 1 to π_2 and 2 to π_1. Intuitively, this means low mileage cars preferred to high mileage cars. All else being equal, red cars are preferred to non-red ones.

6 Query Plan Generation and Evaluation

In this section, we give an overview of query semantics, develop a technique for "OR-aware" topk pruning, and give a summary of query evaluation.

6.1 Overview of Query Semantics

Recall that given a query Q and a set of SRs and ORs, we want to enforce relevant SRs in Q and return query answers ranked by $\mathcal{V}$ (score from value-based ORs), $\mathcal{K}$ (score from keyword-based ORs), $\mathcal{S}$ (query score) or $\mathcal{K}$, $\mathcal{V}$, $\mathcal{S}$. We will focus on one ranking order $\mathcal{K}$, $\mathcal{V}$, $\mathcal{S}$ without loss of generality.

SRs are enforced by conceptually rewriting the query using the SRs. Recall, that SR rules may correspond to relaxation or tightening. A key contribution is that we show SRs can be enforced by encoding the query flock into a single query plan, without requiring actual rewriting. ORs are enforced using new operations defined in the next section. The result of a query is a list of answers ranked by $\mathcal{K}$, $\mathcal{V}$, $\mathcal{S}$.

6.2 Algebra

For ease of exposition, we consider two SRs, ρ_2 and ρ_3, one value-based OR, π_1 and two keyword-based ORs, π_4 and π_5 for our running example query from Fig. 2. For query plans, since the distinguished node (car in our example) is the one being topk-pruned, we wanted to choose plans which facilitate this pruning and allow the distinguished node bindings to be pipelined throughout. Two such (equivalent) plans are given in Fig. 4. We first overview *Plan 1* and defer the discussion on *Plan 2* for later. For both plans, notice that joins with keywords are score contributors (e.g., *ftcontains*("*good condition*")) while other joins aren't (e.g., the structural semijoin between cars and price).

Plan 1 starts by evaluating the user query (cars in good condition and costing less than \$2000), then it enforces the SRs ρ_2 and ρ_3 by using an outer-join which makes "american" and "low mileage" optional. The outer-join ensures american cars with low mileage as well as other cars are captured, and assigns a higher score to american, low mileage cars.

In order to enforce ORs in the plan, we introduce two new operators: `vor` for value-based ORs and `kor` for keyword-based ones. Answers are then sorted by $\mathcal{K}$, $\mathcal{V}$, $\mathcal{S}$. Finally, the last operator, `topkPrune` selects the top best answers and prunes the remaining ones. Additionally, the `sort` operator needs to sort an input list parametrically, i.e., by $\mathcal{K}$, $\mathcal{V}$, $\mathcal{S}$ or by some other order. Fig. 3 summarizes the new operations in the algebra.

A typical optimization is to prune intermediate answers that will not make it to the topk result as early as possible in the plan. In the case where no user profile is being enforced, query answers need to be returned ranked by their query score $\mathcal{S}$. Early pruning can be achieved by enhancing the `topkPrune` operation as given in Algorithm 1. This is in the same spirit as traditional topk pruning [8, 10, 17]. This algorithm is very simple and is based on two key points: (i) maintaining a list of current top k answers, *topkList*, and, (ii) computing a score bound *query-scorebound*, which represents the maximum score that could be acquired by an intermediate answer a in the remaining parts of the query plan. If the score of an answer a, augmented with the maximum score bound, does not exceed the score of the current kth answer, that answer can be safely pruned since it will never make it to the topk final answers. Otherwise, if the score of a exceeds the score of kth, the topk list is updated with a.

Algorithm 1 does not account for OR-contributed scores, i.e., $\mathcal{K}$, nor for value-based ORs, i.e., $\mathcal{V}$. As such it is applicable only when the (query) plan contains no `kor`'s or `vor`'s. Intuitively, in our running example, red cars are preferred to non-red cars (π_1). Moreover, if a car is offered to the best bidder (π_4) and it is located in NYC (π_5), then it should be ranked higher than another car which is in a good condition and has low mileage but does not satisfy the location and bidding preferences. In Algorithm 1, `topkPrune` is

Operator	Description
$\texttt{vor}_{\pi_i}(R)$	applies a value-based OR by augmenting current answers with their OR value
$\texttt{kor}_{\pi_j}(R)$	applies a keyword-based OR by modifying the keyword-based OR score value of current answers
$\texttt{topkPrune}(R)$	applied on sorted input, returns a *topkList* containing the topk answers based on current sorting condition
$\texttt{sort}_{\text{param}}(R)$	sorts input list R based on parameters *param*.

Figure 3. The Algebra

Algorithm 1 `topkPrune` using $\mathcal{S}$ only

```
Require: topkList, intermediate answer a, query-scorebound
Ensure: top k answers ranked by query score $\mathcal{S}$
 1: if (a.$\mathcal{S}$ + query-scorebound $\leq$ kth.$\mathcal{S}$) then
 2:    prune a;
 3: else
 4:    if (a.$\mathcal{S} \geq$ kth.$\mathcal{S}$) then
 5:       insert a in topkList (at the right place) ;
 6:       kth answer is no longer in topkList
 7:       keep kth answer in the flow
 8:    else
 9:       keep a in the flow
10:    end if
11: end if
```

not aware of the two ORs π_4 and π_5 as it only uses the query score $\mathcal{S}$ to decide which answers to prune. Thus, it may prune cars satisfying ORs. Therefore, in order to enable the application of `topkPrune` anywhere in the query plan, its definition needs to be modified to become *OR-aware*. We are now ready to describe our pruning algorithms, in particular, the non-trivial interaction between enforcing ORs and topk pruning.

6.3 Pruning Algorithms

In order to make `topkPrune` OR-aware, we need to modify it in two ways. Algorithm 2, shows how value-based ORs such as π_1, are taken into account in pushing `topkPrune` down a query plan when the query does not contain keyword-based ORs. We use the notation $a \prec_{\mathcal{V}} kth$ to indicate that answer a has an OR value that is preferable to that of kth answer (e.g., a is a red car while kth is not). Similarly, $a ==_{\mathcal{V}}, kth$ means their OR value is equally preferable (e.g., both are red cars). We say a and kth are *incomparable* w.r.t. $\prec_{\mathcal{V}}$ if they are incomparable w.r.t. the partial order defined by the value-based OR.

Given an intermediate answer a that is being considered for pruning and the current kth answer kth, Algorithm 2 augments Algorithm 1 by enabling its pruning routine when a and kth share the same VOR value, or when kth has an OR value that is incomparable to that of a. Finally, in the case where a or kth has an OR value that is preferable, the appropriate pruning takes place.

We are now ready to describe the complete version of `topkPrune`. This version accounts for the presence of keyword-based and value-based ORs. Algorithm 3 shows this case. The key intuition here is to use the same pruning condition on $\mathcal{K}$ as the one used for $\mathcal{S}$ and account for the precedence of $\mathcal{K}$ over $\mathcal{S}$ in ranking query answers.

Algorithm 2 `topkPrune` using $\mathcal{V}$ and $\mathcal{S}$

```
Require: topkList, intermediate answer a
Ensure: top k answers ranked by $\mathcal{V}$, $\mathcal{S}$
 1: if ($a ==_{\mathcal{V}} kth$) then
 2:    apply Algorithm 1
 3: else
 4:    if ($kth \prec_{\mathcal{V}} a$) then
 5:       prune a;
 6:    else
 7:       if ($a \prec_{\mathcal{V}} kth$) then
 8:          insert a in topkList (at the right place)
 9:          kth answer is no longer in topkList
10:          keep kth answer in the flow
11:       else
12:          if ($kth$ and $a$ are incomparable w.r.t. $\prec_{\mathcal{V}}$) then
13:             apply Algorithm 1
14:          end if
15:       end if
16:    end if
17: end if
```

Algorithm 3 uses the same pruning principle as Algorithm 1 and augments Algorithm 2 to make sure that keyword-based ORs take precedence over value-based ORs and over query scores. The principle used in Algorithm 3 is to compute a maximum score bound as a combination of the maximum scores that might be contributed by the remaining keyword-based ORs. More precisely, the operation $\texttt{topkPrune}_1$ applied right before $\texttt{kor}_{\pi_4}$ in *Plan 2* in Fig. 4 uses a *kor-scorebound* which is the sum of the highest scores contributed by both π_4 and π_5 while the $\texttt{topkPrune}_2$ operation applied right before $\texttt{kor}_{\pi_5}$ uses a *kor-scorebound* equal to the highest score contributed by $\texttt{kor}_{\pi_5}$ only. Note that in order to generate *Plan 2*, other rewritings are needed such as pushing selections and commuting `topkPrune` and join operations.

6.4 Query Evaluation Algorithm

A core of our contribution is the pruning algorithms presented in the previous section. We summarize here how all our algorithms fit together and some key points that will be explored in Section 7. We rely on inverted indices on keywords and on an index per distinct tag. We im-

Algorithm 3 topkPrune using $\mathcal{K}, \mathcal{V}, \mathcal{S}$

```
Require: topkList, intermediate answer a
Ensure: top k answers ranked by V, K, S
 1: if (kor-scorebound == 0) then
 2:   if (a.K == kth.K) then
 3:     apply Algorithm 2
 4:   end if
 5: else
 6:   if (a.K + kor-scorebound ≤ kth.K) then
 7:     prune a;
 8:   else
 9:     replace kth answer with a in topkList
10:     kth answer is no longer in topkList
11:     keep kth answer in the flow
12:   end if
13: end if
```

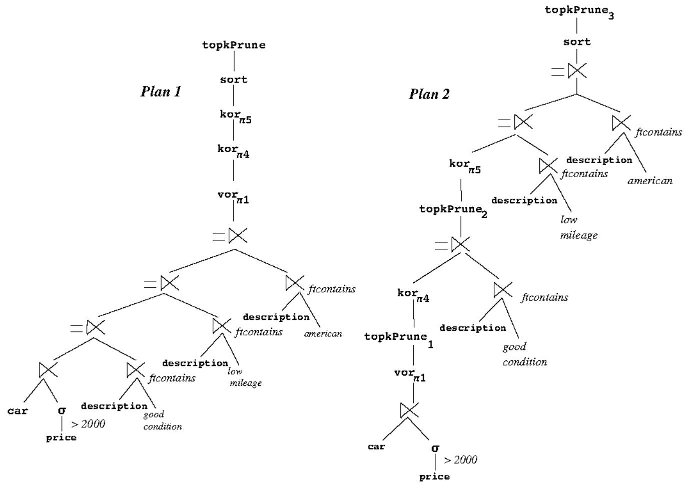


Figure 4. Equivalent Query Plans

plemented indexed nested loop joins. Joins are pipelined. The vor and kor operations are implemented as joins and outer-joins respectively. Joins involving query keywords or keyword-based ORs contribute to the score of their output. The topk list is maintained by the topkPrune operation. When the input to topkPrune is sorted, additional pruning can be achieved. More precisely, given a sorted input to topkPrune, if an intermediate answer is to be pruned then there is no need to produce the remaining answers, as they will be probably inferior. This bulk pruning may result in important savings which make up for the overhead of enforcing SRs and ORs to achieve query personalization.

7 Experiments

We implemented PIMENTO as a collection of Java classes. We performed experiments on a 1.6GHz Pentium PC with 512MB of RAM under Fedora Core 3. We used the INEX document collection for an empirical evaluation of our approach, and the XMark data to show performance results for the different topkPrune algorithms.

7.1 Effectiveness on INEX Collection

The objective of this experiment was to verify the effectiveness of our approach in a real world scenario, for which we chose the INEX collection. The INEX collection consists of full articles from IEEE Computer Society journals and conference proceedings covering a range of topics in the field of computer science. INEX is an initiative whose aim is to provide means, in the form of a large XML test collection and appropriate scoring methods, for the evaluation of XML retrieval systems. The participating organizations create topics (i.e. queries), perform retrieval runs and provide relevance assessments.

The INEX topics consider either content only (i.e., keywords) or content and structure. A content and structure topic consists of a NEXI [20] query, an explanation of the requested information in plain English, and finally a narrative that describes the criteria used by the INEX assessors to determine whether an answer to the query is relevant. Consider for example the INEX topic below:

```
< inex_topic topic_id = "131" query_type = "CAS" >
  < title > //article[about(.//au, "JiaweiHan")]
            //abs[about(]., "data mining")] < /title >
  < description > We are looking for the abstracts of the
    documents about data mining and written by Jiawei Han.
  < /description >
  < narrative > To be relevant, the component has to be
    the abstracts written by Jiawei Han about"data mining"
    Any topics of data mining(e.g. association rules,
    data cube etc.) should beconsidered as relevant.
  < /narrative >< /inex_topic >
```

According to the narrative of the topic, an abstract is relevant to the query if it contains keywords related to data mining (such as association rules, knowledge discovery, data cube, etc.). From this, one can derive the following keyword-based OR:

if $(x.\mathsf{tag} = \mathsf{abs}\ \&\ y.\mathsf{tag} = \mathsf{abs}\ \&\ (\mathit{ftcontains}(x, \text{"}\mathit{data\ cube}\text{"})$
$\&\ \mathit{ftcontains}(x, \text{"}\mathit{association\ rule}\text{"})$
$\&\ \mathit{ftcontains}(x, \text{"}\mathit{data\ mining}\text{"})))$ **then** $x \prec y$

This is just a shorthand for three ORs containing one of the three *ftcontains* predicates. An INEX assessment records for a given topic and a given IEEE article (document), the degree of relevance of the document component to the INEX topic. A component is judged on two dimensions: *relevance* and *coverage*. Relevance judges whether the component contains information relevant to the query subject and coverage describes how much of the document component is relevant to the query subject.

We experimented with 8 INEX topics to examine whether we could capture the narrative of the topic in terms of our scoping and keyword-based ORs. We did not consider thesauri or ontologies to expand the set of keywords included in the query. We considered stemming and uppercase/lower-case options for the query keywords. We focused on topics that request XML nodes that are descendants of article sections. A component can be deemed irrelevant, marginally relevant or highly irrelevant. We consider the components of the document whose relevance is not explicitly reported as irrelevant. In the majority of the topics the nodes that are deemed as relevant by the assessor are not only the ones requested by the query (for example, a topic requests paragraphs containing a certain keyword, but in the assessment article figures are also returned as relevant). For this, we included distinguished nodes other than the ones requested by the query.

We compared the answers obtained by applying the scoping rules and the keyword-based ORs, to the answers deemed relevant by an INEX assessment. For a given topic and assessment, we measured how many of the XML nodes that were deemed relevant in the assessment we missed (*precision*) and how many more XML nodes we retrieved (*recall*). We considered the best 5 answers for each XML element type that was requested by the query.

Table 1 shows the precision and the recall results for the INEX topics that we experimented with.

	Precision		Recall	
Topic	**Missed**	**Out of**	**Retrieved**	**Instead Of**
130	0	7	16	7
131	1	6	13	6
132	3	12	16	12
140	6	20	18	20
141	0	5	17	5
142	1	8	14	8
145	0	6	15	6
151	0	6	11	6

Table 1. INEX results

One can observe that we achieve rather good precision results. On the other hand, we retrieve more document components than the ones included in the INEX assessment, leading to poor recall results. In average, it was sufficient to get the 5 best results for each XML node type that we considered in order to get the relevant nodes and have a relatively good recall ratio.

Nevertheless, we observed that the components retrieved by applying the SRs and ORs which are not included in the INEX assessment, have a non-negligible score for the keywords of the query and the narrative. As a consequence, we are not sure whether these retrieved elements were considered as irrelevant by the author of the assessment, or were simply ignored. Another interesting observation that we made was that when we applied some form of relaxation (like stemming, or upper/lower case), then the precision decreased in the cases in which a node that was marginally relevant w.r.t the query and keywords in SRs, became highly relevant because it was containing relaxed forms of those keywords. In this case, it made the topk answers by removing some other node that was containing exactly the query keywords. After this empirical evaluation, one can conclude that we need to consider weights for our SRs and incorporate those weights when the query score is computed.

7.2 Performance results

This section studies the performance of early pruning in the presence of keyword-based ORs. In particular, we observed that pushing `topkPrune` in the plan does not always pay off due to the distribution of scores contributed by each keyword-based OR.

Query Q: *ad(*`person,business`*) & ftcontains(*`business,` *"Yes")*

π1: *x*.**`tag`**=*person & y*.**`tag`**=*person & ftcontains(x,"male")* $\longrightarrow x \propto y$
π2: *x*.**`tag`**=*person & y*.**`tag`**=*person & ftcontains(x,"United States")* $\longrightarrow x \propto y$
π3: *x*.**`tag`**=*person & y*.**`tag`**=*person & ftcontains(x,"College")* $\longrightarrow x \propto y$
π4: *x*.**`tag`**=*person & y*.**`tag`**=*person & ftcontains(x,"Phoenix")* $\longrightarrow x \propto y$
π5: *x*.**`tag`**=*person & y*.**`tag`**=*person & x.age =33 & y.age <> 33* $\longrightarrow x \propto y$

Figure 5. XMark Query Q

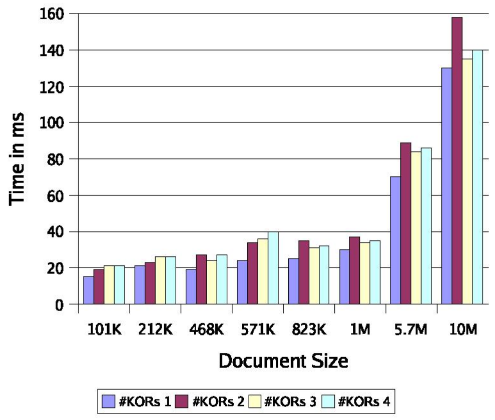


Figure 6. Query time for PushtopKPrune, for increasing number of KORs and increasing document size

We measure the performance of three plans: **Naivetop-kPrune** *(Ntkp)* where `topkPrune` is applied at the end of the plan, **InterleavetopkPrune** *(ILtpk)* where `topkPrune` is applied after each application of a KOR and, **Pushtop-KPrune** *(Ptkp)* where `topkPrune` is pushed all the way the plan. The query and ORs we are using are reported in Fig. 5.

The first experiment is reported in Fig. 6. We used XMark documents of increasing size. The experiment shows that **PushtopKPrune** scales with an increasing document size and an increasing number of KORs (1 to 4). In particular, the difference in query response time between a 1MB document and a 5.7MB document is sub-linear.

Fig. 7 reports the result of running four equivalent plans for the query in Fig. 5 on a 10MB document. The main observation is that pushing `topkPrune` all the way down the plan (**PushtopKPrune**) saves computation while applying it after each keyword-based OR (**InterleavetopkPrune**) induces too much overhead. In particular, in this case, most of the pruning is done by the first `topkPrune` operation and the additional `topkPrune` operations do not pay off. Note also that **InterleavetopkPrune** with sorting (*S-ItpkP*) outperforms the non-sorted version (*NS-ItpkP*) as it enables batch pruning.

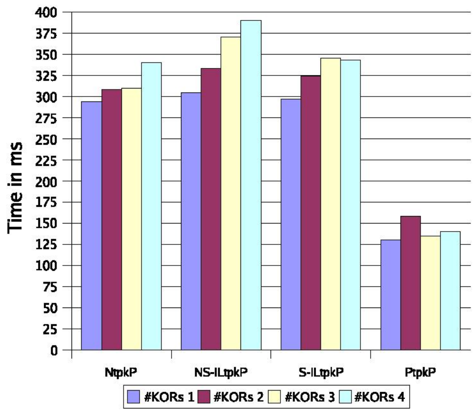


Figure 7. Comparison of query evaluation times, for increasing number of KORs on a 10MB document

We tried these four plans on two other queries and observed that **PushtopKPrune** never does worse than **Naive**. In general, the performance of the plans depends on two key points: i) the order of application of the KORs, and ii) the value of the maximum score bound contributed by the remaining KORs in the plan. We observed that applying the KOR which contributes the highest score first is beneficial as it increases the pruning threshold. In addition, given that the maximum score bound for KORs is a conservative estimate, the higher the number of answers to which it applies, the more effective the pruning is. This suggests that pruning pays the most when the scores contributed by the KORs are uniformly distributed across answers.

8 Conclusion

We presented a novel approach to XML search that leverages user information to return more relevant query answers. This approach is based on a formalization of user profiles in terms of scoping rules which are used to rewrite an input query, and of ordering rules which are combined with query scoring to customize the ranking of query answers to specific users. We showed that enforcing user profiles yields higher precision and recall on INEX queries and we showed on XMark that personalization does not compromise query efficiency. We intend to explore the idea of using weights to perform a fine-tunning of the application of the SRs as described in the experiments.

References

[1] R. Agrawal and et. al. A framework for Expressing and Combining Preferences. In *SIGMOD*, 2000.
[2] S. Amer-Yahia and et. al. Minimization of Tree Pattern Queries. In *SIGMOD*, 2001.
[3] S. Amer-Yahia and et. al. FleXPath: Flexible Structure and Full-Text Querying for XML. In *SIGMOD*, 2004.
[4] W.-T. Balke and et. al. On Real-time Top-k Querying for Mobile Devices. In *CoopIS*, 2002.
[5] S. Borzsonyi and et. al. The Skyline Operator. In *ICDE*, 2001.
[6] C. Botev and et. al. Expressiveness and Performance of Full-Text Search Languages. In *EDBT*, 2006.
[7] N. Bruno and et. al. Evaluating Top-k Queries over Web-Accessible Databases. In *ICDE*, 2002.
[8] S. Chaudhuri and et. al. Evaluating Top-k Selection Queries. In *VLDB*, 1999.
[9] J. Chomicki. Preference formulas in relational queries. *ACM TODS*, 28(4):427–466, 2003.
[10] R. Fagin and et. al. Optimal Aggregation Algorithms for Middleware. In *PODS*, 2001.
[11] Initiative for the Evaluation of XML Retrieval. http://inex.is.informatik.uni-duisburg.de:2004/.
[12] W. Kiessling. Foundations of Preferences in Database Systems. In *VLDB*, 2002.
[13] W. Kiessling and et. al. Preference XPATH: A Query Language for E-Commerce. In *Konferenz fur Wirtschaftsinformatik*, 2001.
[14] G. Koutrika and et. al. Rule-based query personalization in digital libraries. *IJDL*, 4(1):60–63, 2004.
[15] G. Koutrika and et. al. Personalized Queries under a Generalized Preference Model. In *ICDE*, 2005.
[16] C. Li and et. al. RankSQL: query algebra and optimization for relational top-k queries. In *SIGMOD*, 2005.
[17] A. Marian and et. al. Adaptive Processing of Top-k Queries in XML. In *VLDB*, 2005.
[18] G. Miklau and D. Suciu. Containment and Equivalence for an XPath Fragment. In *PODS*, 2002.
[19] T. Schlieder. Schema-Driven Evaluation of Approximate Tree-Pattern Queries. In *EDBT*, 2002.
[20] A. Trotman and et. al. NEXI, Now and Next. In *INEX*, 2004.
[21] XQuery 1.0 and XPath 2.0 Full-Text. http://www.w3.org/TR/2006/WD-xquery-full-text-20060501/, 2006. W3C Working Draft.