AN APPROXIMATION-AWARE ALGEBRA FOR XML FULL-TEXT QUERIES*

Giacomo Buratti
Department of Mathematics and Informatics, University of Camerino, Via Madonna delle Carceri 9, Camerino, Italy

Danilo Montesi
Department of Computer Science, University of Bologna, Mura Anteo Zamboni 7, Bologna, Italy

Keywords: XML, Algebra, Full-Text, Approximate Query.

Abstract: XQuery Full-Text is the proposed standard language for querying XML documents using either standard or full-text conditions; while full-text conditions can have a boolean or a ranked semantics, standard conditions must be satisfied for an element to be returned. This paper proposes a more general formal model that considers structural, value-based and full-text conditions as *desiderata* rather than *mandatory* constraints. The goal is achieved defining a set of relaxation operators that, given a path expression or a selection condition, return a set of relaxed path expressions or selection conditions. Algebraic approximated operators are defined for representing typical queries and returns either elements that perfectly respect the conditions and elements that answer to a relaxed version of the original query. A score reflecting the level of satisfaction of the original query is assigned to each result of the relaxed query.

1 INTRODUCTION

The study of semi-structured data and XML have received in the last years a further boost from a new trend: the integration of structured, semi-structured and unstructured data into a more general framework (INEX, 2006). A convergence between these diverging models is made necessary by the consideration that many today's applications have to cope with data covering the entire spectrum. This requirement led to the definition of many query languages for XML with full-text capabilities; lastly, W3C has published a Working Draft for XQuery Full-Text (W3C, 2006), a extension of XQuery with full-text operators.

Our contribution to this convergence effort has been the definition of AFTX (Buratti, 2007), an algebra for semi-structured and full-text queries over XML repositories. Our algebra includes either basic or full-text operators. Basic operators are used to restructure and filter the input trees. Full-text operators perform full-text searches, either using a *boolean* model or a *ranked* retrieval. In the first case a binary judgement (relevant / not relevant) is assigned to each input tree; in the second case a *score* value, reflecting how relevant the tree is with respect to the user query, is calculated.

1.1 The Need for Approximation

One of the main differences between structured and semi-structured paradigm is the *flexibility* of the schema. In fact the schema specifications for an XML document can define some elements as optional, or the cardinality of an element can differ from documents to documents; moreover, it is perfectly legal for an XML document not to have an associated schema at all. This flexibility poses interesting questions for what concerns answering to a query that imposes some constraints on the structure of XML fragments to retrieve; it could be the case that such constraints are satisfied by a very small part of input documents. Nevertheless, there could be documents that are relevant to users, even if they do not closely respect some structure constraints.

XQuery Full-Text treats basic conditions (i.e. navigational expressions and constraints on value of elements or attributes) and full-text conditions in a non-uniform way. In fact, when writing full-text con-

*This work has been supported by CIPE 4/2004

Buratti G. and Montesi D. (2007).
AN APPROXIMATION-AWARE ALGEBRA FOR XML FULL-TEXT QUERIES.
In *Proceedings of the Second International Conference on Software and Data Technologies - Volume ISDM/WsEHST/DC*, pages 62-69
DOI: 10.5220/0001327700620069

ditions, it is possible to specify whether to use a boolean semantics or to perform a ranked retrieval. On the contrary, basic conditions are always treated as *mandatory*: in order to be retrieved, an element *must* be reachable by exactly following the specified path expression, and all the conditions on values *must* be satisfied.

In the effort of providing a uniform treatment of basic and full-text conditions, the key idea is to consider the searched path expression and the specified conditions on values as *desirable* properties to enjoy for an element to be returned, instead that considering them as mandatory constraints. Therefore, an element should be returned even if it does not perfectly respect basic conditions, and a *score* value should indicate how well such conditions are satisfied.

1.2 A Motivating Example

Consider the XML document shown graphically in Figure 1. Suppose a user writes an XQuery expression containing the `for` clause `for $a in doc(` "bib.xml")`/bib/book/author`. The user need is probably to find all book authors, including those who just co-authored a book. The `for` clause, however, will find only those authors that are the single authors of at least one book. If the `for` clause has an *approximated* behavior, it could also return a subtree reachable by following an *relaxed* version of the original path expression, for example `/bib/book//author`, therefore including co-authors in the result.

This relaxed query would find all the book authors, but not the paper authors. It could be the case that such authors are also of interest for the user. The user need could be satifisfied by further relaxing the query, i.e. the path expression `/bib/book//author` could be transformed into the path expression `/bib//author`.

Suppose now the user writes an expression containing the clause `for $a in doc(` "bib.xml")`/bib/book/title`. Such a query finds all the book titles, but ignores paper titles, which could also be interesting for the user. In fact some semantic relationship exists between the words *book* and *paper*: using some lexical database (e.g. (Princeton University, 2007)) we can find that both these words are hyponyms of the word *publication*. Considering such a relationship, a different kind of relaxation could treat the path `/bib/paper/title` as a approximated version of `/bib/book/title` and therefore include in the result also the paper titles.

Let us now consider a `for` clause including a filter predicate based on the full-text operator `ftcontains`, like the following:

```
for $a in doc("bib.xml")//paper
    [//section/title ftcontains "INEX"]
```

We are looking for papers that include in a section title the word *INEX*. The paper shown in Figure 1 is not returned, because the titles of the various sections do not include the searched word. However, the word is included in the content of the first section of the paper, therefore the paper is probably of interest for the user. A possible relaxation could transform the previous query by removing the last step in the path expression, thus obtaining:

```
for $a in doc("bib.xml")//paper
    [//section ftcontains "INEX"]
```

As a final example, consider the partial query

```
for $b in doc("bib.xml")/bib/book
where $b /price < 39
```

The user wants to find books with a price lower than 39. However, it could be the case that very few books satisfy such a constraint (in the document of Figure 1, no book satisfies the constraint); consequently, the user could also be interested in books having a price of 39, or even in books having a price not much greater than 39. A relaxed version of the `where` clause could return such books, by substituting the condition `$b/price < 39` with an approximated version of it, obtained by changing the comparison operator (`$b/price` $\leq$ `39`) or even increasing the threshold price (`$b/price < 45`).

1.3 Our Contribution

The purpose of this paper is to formally define the notion of query relaxation that has been informally presented. Section 2 represents the core of the paper; here we introduce the various relaxation operators, that perform one of the following tasks: 1) given a path expression, define a set of relaxed path expressions; 2) given a predicate on a element value, define a set of relaxed predicates.

With respect to (Amer-Yahia et al., 2004), the paper that mainly influenced us, our work has the advantage of considering a wider spectrum of relaxations. Moreover, we incorporate the notion of approximation into a general algebraic framework suitable for representing queries over XML.

The relaxation operators are then used in Section 3 to define a set of approximated algebraic operators. These operators are a variant of some of those previously defined for AFTX; AFTX, which is briefly reviewed in the same section, is an algebra working on forests of trees, i.e. ordered lists of trees. For a deeper treatment of AFTX data model, algebraic op-

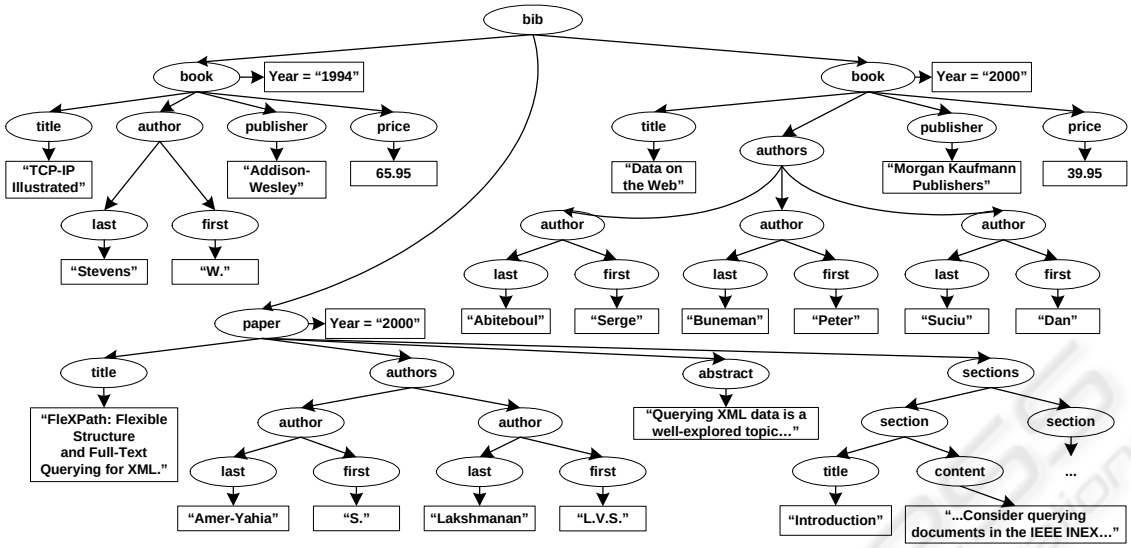

Figure 1: Graphical representation of an XML document.

erators, XQuery Full-Text translation, and algebraic optimization see (Buratti, 2007).

Approximated algebraic operators are based on the concept of *score*: a *relaxed answer* (i.e. an answer to a relaxed query) has a score which reflects how exact is the query that returns such an answer. In a certain way exact and relaxed queries play the same role of boolean and ranked retrieval in classical Information Retrieval (and in XQuery Full-Text): while exact queries classify each document fragment as either relevant (i.e. fulfilling constraints imposed on document structure and elements / attributes value) or not relevant, relaxed queries establish how relevant a fragment is. The definition of an approximation-aware algebra for XML standard and full-text search can serve as the theoretical foundation for the definition of an approximation-aware version of XQuery Full-Text.

The main contribution of the presented work is the formal definition of the concept of approximation in terms of approximation operators. Therefore it should be regarded to as a valuable starting point for future research in the area; in Section 4, besides drawing some conclusions, we outline some possible future research directions.

2 RELAXATION OPERATORS

In this section we formally define a set of relaxation operators. We propose two kinds of relaxation: *path relaxations* and *comparison relaxations*.

2.1 Path Relaxations

We define four path relaxation functions: axis relaxation, internal step deletion, final step deletion, and name relaxation. Their goal is to obtain a set of relaxed path expressions, starting from an input path expression.

Axis Relaxation P_A approximates a path expression λ_1/λ_2 by substituting a child axes with a descendant axes, thus obtaining the relaxed expression $\lambda_1//\lambda_2$. Its goal has already been depicted in an example in Section 1, where the path expression `/bib/book/author` were transformed into `/bib/book//author`.

It is clear that multiple relaxed expression can be obtained from the original expression, possibly recursively applying the relaxation function; in this example, other possible relaxed expressions are `/bib//book/author`, `/bib//book//author`, `//bib/book//author` etc. Formally, the set of obtainable relaxed expressions has cardinality $\sum_{i=1}^{n}\binom{n}{i}$.

Internal Step Deletion P_S approximates a path expression $\lambda_1\alpha\beta//\lambda_2$ (where α is an axes and β is an element name) by eliminating the internal step $\alpha\beta$, thus obtaining the relaxed expression $\lambda_1//\lambda_2$. As an example, consider the XML document in Figure 1 and suppose to start with the path expression `/bib/book/authors/author`. Using P_A we could transform it into the relaxed version `/bib/book/authors//author`, but this transformation is not yet sufficient to capture authors of a book having just one author; now we can apply P_S, obtaining `/bib/book//author`, which captures either au-

thors and co-authors.

Final Step Deletion P_F removes from the original path expression $\lambda\alpha\beta$ the final step $\alpha\beta$, thus obtaining the relaxed expression λ; for example, using P_F we can transform the expression `/bib/book/content` into the expression `/bib/book`. The behaviour of the final step deletion function is radically different to that of P_A and P_S; while an application of P_A or P_S results in a relaxed path expression that reaches all the elements that can be reached using the original path expression (plus some extra elements), an application of P_F results in a set of path expressions that reach a completely different set of elements. As one could expect, its usage is therefore different from that of the two previous relaxations, as we will see in Section 3.

Name Relaxation P_N substitutes an element name β in a path expression $\lambda_1\alpha\beta\lambda_2$ with another name β', thus obtaining the relaxed expression $\lambda_1\alpha\beta'\lambda_2$. For example, using P_N we could transform the expression`//book/author` into the expression `//publication/author`.

In the spirit of *data integration*, this operation should be intended as a way to manage heterogenous data sources disregarding possible name conflicts due to the usage, for example, of synonyms in the schema definition. Generally speaking, the Name Relaxation function should be thought of as a way to substitute a name with another one that has a certain degree of *similarity* with it; such a similarity could be calculated, for example, using an ontology.

2.2 Comparison Relaxations

Typical operations on XML documents also involve checking the satisfaction of a comparison predicate. For example, having selected a set of `book` elements, we could filter those element on the basis of the book price, using the predicate `/price < 50`. There are two possible relaxations we can perform on a predicate like this: operator relaxation and value relaxation.

Operator Relaxation C_O substitutes an operator θ in a comparison expression $x\theta y$ with another operator θ', thus obtaining $x\theta' y$. For example, using C_O we could transform $/\texttt{price} < 50$ into the relaxed comparison $/\texttt{price} \leq 50$.

Clearly, such a transformation makes sense only if the new operator guarantees a larger (or at least equal) number of successful comparison than the original one. A partial ordering relation $\preceq$ can be defined between available operators, stating that $\theta \preceq \theta'$ if, for each pair of operands (x,y), $x\theta y$ implies $x\theta' y$. An example of ordering relationships between operands on numeric values is shown in Figure 2, where an arrow from operand θ to operand θ' means that $\theta \preceq \theta'$; similarly, an ordering relationship should be defined between operators on strings, dates, etc.

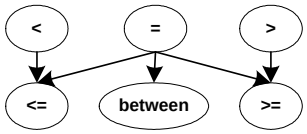

Figure 2: Ordering between numeric operators.

Value Relaxation C_V substitutes an operand y in a comparison expression $x\theta y$ with another operand y', thus obtaining $x\theta y'$. For example, using C_V we could transform $/\texttt{price} < 50$ into the relaxed comparison $/\texttt{price} < 55$.

This relaxation, along with C_O, permits to include in the list of satisfactory elements also those whose value slightly differs from the user request. As usual, its goal is therefore to expand the space of possible results. The choice of the new value y' should depend on the comparison operator. For example, given the comparison $x < y$, y' should be a value such that $y' > y$.

3 APPROXIMATED ALGEBRAIC OPERATORS

In this section we define a set of approximated algebraic operators. For each of them, we briefly recall the semantics of the basic AFTX operator and define the new operator on the basis of the relaxation operators that can be used.

3.1 Approximated Projection

AFTX projection π is a unary operator that takes is a forest and operates a *vertical* decomposition on it: every input tree contributes, with the subtrees of interest, to the projection output. The subtrees of interest are specified in the projection predicate through a *path expression* λ, a concept almost identical to that used in XPath, except for the fact that it can not contain selection conditions. For example, the AFTX expression $\pi_{/\texttt{bib/book}}(\text{“}\texttt{bib.xml}\text{”})$, where *bib.xml* is the XML document shown in Figure 1, returns a forest containing two trees, that correspond to the two subtrees rooted at `book` that can be found in the input tree.

The **approximated projection** $\pi^\star$ operator has the following behavior:

- calculate all possible relaxation of the path expression λ in the predicate, using Axis Relaxation P_A, Internal Step Deletion P_S, Name Relaxation P_N;
- for each relaxed path expression, execute the projection using that path expression;
- calculate the union of the projection results (including the result of the projection that uses the original path expression), eliminating duplicate trees (i.e. subtrees reachable by following two different relaxed path expressions).

As one could expect, the Final Step Deletion relaxation function P_F is not used here. In fact, such a relaxation would lead to results completely unrelated to those expected. For example, suppose we want to do a projection using the predicate `/bib/book/author`; by eliminating the final step we would obtain the predicate `/bib/book`; trees resulting from such a projection would represent books, a completely different concept to that of authors.

Example 1 *Consider the XML document in Figure 1 and suppose to write the algebraic expression* $\pi^*_{\texttt{/bib/book/author}}(\text{“}\texttt{bib.xml}\text{”})$. *Using the basic AFTX projection this expression would return only the author* W. Stevens. *However, by applying P_A to the projection path expression we can obtain the relaxed path expression* `/bib/book//author`, *thus including in the result also* Serge Abiteboul, Peter Buneman, *and* Dan Suciu. *Moreover, by applying P_N we can substitute* `book` *with* `paper`, *thus obtaining the path expression* `/bib/paper//author` *and adding to the result also* S. Amer-Yahia *and* L.V.S. Lakshmanan.

3.2 Approximated Selection

AFTX selection σ is a unary operator that takes is a forest and operates a *horizontal* decomposition on it: only the trees of interest contribute, with their entire content, to the selection output. The selection predicate is composed by a path expression λ and a *selection condition* γ. The path expression is used to operate a temporary projection on the input tree. Each subtree T' belonging to the temporary projection result is then checked: if at least one of them satisfies the selection condition, the original input tree is added to the selection output.

The selection condition is a conjunction of base selection conditions. The evaluation of each base condition γ_i depends on its form:

- if γ_i is the form λ', it is satisfied if exists at least one subtree T'_1 that can be reached from $root(T')$ by following λ';
- if γ_i is the form $\lambda' \equiv \lambda''$, it is satisfied if exists at least a pair of subtrees (T'_1, T'_2) that can be reached from $root(T')$ by following, respectively, λ' and λ'', and such that T'_1 is strictly equal to T'_2 (informally, strict equality between trees means that the two trees are two copies of *the same* tree);
- if γ_i is the form $\lambda' p \theta x$, where x is a constant and p is an element property (for example its value, the value of one of its attributes, etc.), it is satisfied if exists at least one subtree T'_1 that can be reached from $root(T')$ by following λ' such that $root(T'_1)p$ is in relation θ with x;
- if γ_i is the form $\lambda' p' \theta \lambda'' p''$, it is satisfied if exists at least a pair of subtrees (T'_1, T'_2) that can be reached from $root(T')$ by following, respectively, λ' and λ'', and such that $root(T'_1)p'$ is in relation θ with $root(T'_2)p''$.

For example, the AFTX expression $\sigma_{\texttt{/book[/author AND /price.v<40]}}(\pi_{\texttt{/bib/book}}(\text{“}\texttt{bib.xml}\text{”}))$ returns a forest containing the subtrees rooted at `book` that have at least one `author` sub-element and such that the value of the `price` sub-element is less than 40.

The **approximated selection** $\sigma^\star$ operator does the following:

- calculate all possible relaxation of the path expression λ in the predicate, using Axis Relaxation P_A, Internal Step Deletion P_S, Name Relaxation P_N;
- calculate all possible relaxations of the selection condition γ, applying some relaxation function to each base selection condition depending on the kind of the base selection condition;
- for each relaxed path expression and selection condition, calculate the result of the selection;
- calculate the union of the selection results (including the result of the selection that uses the original selection predicate), eliminating duplicate trees.

If a base condition γ_i is the form λ', we can relax λ' as usual. For example, the selection predicate `/book[/authors/author]` can be relaxed into `/book[//author]`. Moreover, P_F can also be applied. For example `/book[/authors/author]` can be relaxed into `/book[/authors]`, or even into `/book[]`; in practice, we relax (or even eliminate) the constraint on the presence of a subtree.

If γ_i is the form $\lambda' \equiv \lambda''$, we can apply to λ' and λ'' P_A, P_S and P_N. For example the selection predicate `/books[/csbook/authors /author` $\equiv$ `/mathbook/authors/author]` can be relaxed into `/books[/csbook//author` $\equiv$ `/mathbook//author]`.

If γ_i is the form $\lambda' p \theta x$, we can apply P_A, P_S and P_N on λ'; for example `/book[/authors/author.count > 1]` ("find all the books with more than one author") can be relaxed into `/book[//author.count > 1]`. Moreover, we can apply C_O and C_V on $p\theta x$; for example the predicate `book[/price.v < 50]` can be relaxed into `book[/price.v` $\leq$ `55]`.

Finally, if γ_i is the form $\lambda' p' \theta \lambda'' p''$, we can use all the relaxations seen for the previous case; moreover λ'' can also be relaxed using P_A, P_S and P_N. For example, the predicate `/books[/csbook/totalprice < /mathbook /totalprice]` could be relaxed into `/books[ /csbook/price` $\leq$ `/mathbook/price]` using: 1) Name Relaxation on `/csbook/totalprice`, 2) Name Relaxation on `/mathbook/totalprice`, and 3) Operator Relaxation on <.

Example 2 *Consider the XML document in Figure 1 and suppose to write the following algebraic expression:*
$\sigma^*_{\texttt{/book[/author/last.v="Amer-Yahia" OR /price.v<60]}}(\pi^*_{\texttt{/bib/book}}(\texttt{"bib.xml"}))$.

Using non-approximated operators, projection returns a forest containing the two books, and the subsequent selection retains the book Data on the Web. *The paper is not returned, even if, having* Amer-Yahia *among its authors, it is probably of interest for the user. However, using the approximated selection and projection operators:*

- *projection returns also the paper, because using P_N the path expression* `/bib/book` *can be transformed into* `/bib/paper`*;*
- *selection retains the paper in the result; using P_N and P_A the selection predicate is relaxed into* `/paper[//author/last.v = ''Amer-Yahia'' OR /price.v < 60]`*;*
- *selection retains the book* TCP-IP Illustrated *also; in fact the selection base condition* `/price.v < 60` *can be transformed into* `/price.v < 70`.

3.3 Approximated Full-text Selection

AFTX full-text selection ς behaves in a way similar to that of basic selection operator: it performs a horizontal decomposition of the input forest, retaining only those trees having at least one subtree satisfying the full-text selection predicate. The full-text selection predicate allows to search one or more words or phrases (specified by the parameter γ, which is a list of word or phrases connected with boolean operators) into the full-text value of an element (i.e. the value of the element concatenated with the value of its sub-elements) or into the value of an attribute a. Moreover, it supports *proximity search*. For example, the expression $\varsigma_{\texttt{/*/title["XML" OR "Web"]}}(\pi_{\texttt{/bib/*}}(\texttt{"bib.xml"}))$ returns all the books or papers that contain the word *XML* or the word *Web* in the title (note that * is a wildcard that means "*any name*").

The **approximated full-text selection** operator $\varsigma^\star$, as usual, transforms the predicate using some relaxation operator and returns the union of the results of the relaxed full-text selections.

First of all, the path expression λ can be subject to P_A, P_S and P_N; for example, `/book/chapter/section["XML"]` can be relaxed into `/publication//section["XML"]`, thus obtaining as result also those publications (like papers) which are not divided into chapters.

Another relaxation function that is worthwhile applying is P_F, thus *broadening* the search scope. For example, relaxing `/book/title["XML" AND "Algebra"]` into `/book["XML" AND "algebra"]` we obtain as result all the books that contain the searched words everywhere, instead that just in the title: we have broadened the search scope from the full-text value of `/book/title` to the full-text value of `/book` (that includes the full-text value of `/book/title`).

Example 3 *Consider the XML document in Figure 1. Suppose we look for papers that include, in their title, the words* XML *and* INEX. *Then we write the following algebraic expression:*
$\varsigma_{\texttt{/paper/title["XML" AND "INEX"]}}(\pi_{\texttt{/bib/paper}}(\texttt{"bib.xml"}))$.
This expression would return an empty answer; in fact the paper title contains the word XML*, while the word* INEX *is included only in the content of the first section. However, using P_F we can remove the* `title` *step in the path expression of the full-text selection predicate, thus obtaining the relaxed predicate* `/paper["XML" AND "INEX"]`. *Therefore the paper will be returned, because both searched words are found in its full-text value.*

3.4 Generalized Top-K and Threshold Selection

The three approximated operators have the goal to broaden the result space, by adding some trees that would have been discarded by applying the corresponding exact operator. Their usage is therefore valuable, because a strict interpretation of conditions imposed by the user query could discard trees which could be of interest for the user, even if they do not perfectly respect some conditions.

However, the usage of such relaxations could lead to the opposite problem: the user who poses the query

could be overwhelmed by a huge amount of answers. What is needed is therefore a way to filter such results, retaining only those that best match the user needs.

A similar problem has already been tackled in basic AFTX regarding the full-text score assignment operator ξ. This operator does not perform a selection: each input tree is returned, without filtering. What it does is to assign to each tree a *score* value that represents the relevance of that tree to the full-text search condition; this value is stored in the *score* property of the root element. The full-text condition is specified in the score assignment predicate, in the same way as in the full-text selection predicate. However, a *weight* can be assigned to each word or phrase (within the parameter γ) in order to specify which words (or phrases) should highly influence score calculation. Moreover, an extra parameter f, which can be thought of as a *function pointer*, specifies the way the score is calculated.

After using full-text score assignment, a user typically wants to receive results in score order, disregarding those with a lower score. The solution has been found in the introduction of two derived operators: top-K full-text selection $\top$ and threshold full-text selection ω. They assign a score to each input tree and return, respectively, the k trees with highest scores and those trees whose score is higher than a defined threshold τ; in both cases trees are returned in descending score order.

Having introduced relaxed operators into our algebra, now such a process of filtering and ordering could be based on two kinds of score:

- the full-text score, which represents the level of satisfaction of full-text conditions; such a score is calculated by full-text score assignment;
- a new *structural* score, which represents the level of satisfaction of non-full-text conditions.

Informally, the structural score should be the answer to the question *"how much have you relaxed my query in order to include this tree in the result?"*. The best way to calculate such a score is not trivial to find; similarly, the way to combine the structural score with the full-text score is a quite interesting problem. Both these issues are beyond the scope of this paper and are candidate targets for future research ((Amer-Yahia et al., 2005) and (Marian et al., 2005) both deal with this issue). However, supposing to have a set of *structural score calculation functions* and a set of *combined score calculation functions*, we can define a generalized version of top-K $\top^*_{f_1,f_2,k}(F)$ and threshold $\omega^*_{f_1,f_2,\tau}(F)$ operators, which operates as follows:

- calculate the structural score using the function f_1;
- combine the structural score just calculated and the full-text score (previously calculated by some score operator) using the function f_2;
- retain in the output, respectively, the k trees with highest score or the trees with a score higher than τ.

Example 4 *Consider the XML document shown in Figure 1 and suppose to write the following algebraic expression:*

$$\top^*_{f_1,f_2,1}(\xi_{/*/\texttt{title}[0.2\ \text{“}\texttt{Web}\text{”}\ \texttt{OR}\ 0.8\ \text{“}\texttt{XML}\text{”}]\texttt{f}}(\pi^*_{/\texttt{bib}/\texttt{book}}(\text{“}\texttt{bib.xml}\text{”})))\ .$$

The projection find all books, then the score assignment calculates a score, using the scoring function f and considering the word XML *more important than* Web*; finally the generalized top-K returns the book with the highest combined score.*

By applying P_N to the path expression, the relaxed projection returns a forest containing three trees, corresponding to the two books and to the paper. Suppose the full-text scoring function f calculates a simple sum of the weights of the found words; then the full-text score of the two books are, respectively, 0 *and* 0.2*, while the paper has a full-text score of* 0.8*.*

Suppose now the structural score function f_1, when P_N has been used, assigns a structural score corresponding to the degree of similarity between the original word and the substitute, and suppose that the similarity between book *and* paper *is* 0.7*. Therefore, the two books has a structural score of* 1 *(because no relaxation has been done for them), while the paper has a structural score of* 0.7*.*

Finally, suppose the combined score calculation function f_2 returns a weighted sum of the structural score (with weight 0.2*) and the full-text score (with weight* 0.8*). Then: the book* TCP-IP Illustrated *has a combined score of* $1*0.2+0=0.2$*; the book* Data on the Web *has a combined score of* $1*0.2+0.2*0.8=0.36$*; the paper has a combined score of* $0.7*0.2+0.8*0.8=0.78$*. Therefore, the generalized top-k operator returns the paper, which is the publication with the highest combined score.*

4 CONCLUSION AND FUTURE WORK

In this paper we have presented an approximation-aware theoretical framework for full-text search over XML repositories. Some relaxation operators have been presented and used for defining approximated algebraic operators. The algebra is intended as a formal basis for the definition of an approximated query lan-

guage for XML, which should be an extension of the actual W3C candidate standard, XQuery Full-Text.

As already mentioned, a first future research direction is the development of valid scoring methods for calculating the structural score of a tree, i.e. a score reflecting how precisely a tree satisfies the structural conditions imposed by a user query. Moreover, a set of combined score calculation functions should be developed; their goal is to combine the structural score and the full-text score, returning a global score reflecting how precisely a tree satisfies structural and full-text conditions. The combined score calculation function should enjoy some properties, e.g. the value of the result should be a continuous function of the weights assigned to structural and full-text score (Fagin and Wimmers, 2000).

Considering structural constraints of a query as a *desiderata* instead that a requirement also poses interesting performance issues. In fact, dealing with relaxation means transforming a query into a set of similar queries, each of which must be executed in order to calculate the final result. It is therefore needed a way to efficiently compute such answers. This problem is closely related to that of score calculation; again, the usage of scoring functions enjoying some properties (like monotonicity) should allow the definition of impacting optimization strategies, for example allowing to *prune* some part of the tree of the possible relaxed queries, which means avoiding to execute part of the relaxed queries.

REFERENCES

Amer-Yahia, S., Koudas, N., Marian, A., Srivastava, D., and Toman, D. (2005). Structure and Content Scoring for XML. In *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB 2005)*, pages 361–372, Trondheim, Norway.

Amer-Yahia, S., Lakshmanan, L. V. S., and Pandit, S. (2004). FleXPath: Flexible Structure and Full-Text Querying for XML. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 83–94, Paris, France.

Buratti, G. (2007). A Model and an Algebra for Semi-Structured and Full-Text Queries (Ph.D. Thesis). Technical Report UBLCS-2007-03, University of Bologna.

Fagin, R. and Wimmers, E. L. (2000). A Formula for Incorporating Weights into Scoring Rules. *Theoretical Computer Science*, 239(2):309–338.

INEX (2006). INitiative for the Evaluation of XML Retrieval. `http://inex.is.informatik.uni-duisburg.de/2006/`.

Marian, A., Amer-Yahia, S., Koudas, N., and Srivastava, D. (2005). Adaptive Processing of Top-K Queries in XML. In *Proceedings of the 21st International Conference on Data Engineering (ICDE 2005)*, pages 162–173, Tokyo, Japan.

Princeton University, C. S. L. (2007). Wordnet. http://wordnet.princeton.edu/.

W3C (2006). XQuery 1.0 and XPath 2.0 Full-Text, W3C Working Draft. `http://www.w3.org/TR/xquery-full-text/`.