An Approach to Define Flexible Structural Constraints in XQuery

Emanuele Panzeri and Gabriella Pasi

University of Milano-Bicocca
Viale Sarca 336, 20126 Milano, Italy
{panzeri,pasi}@disco.unimib.it

Abstract. This paper presents a formal definition of an extension of the XQuery Full-Text language: the proposed extension consists in adding two new flexible axes, named `below` and `near`, which express structural constraints that can be specified by the user. Both constraints are evaluated in an approximate way with respect to a considered path, and their evaluation produces a path relevance score for each retrieved element. The formal syntax and the semantics of the two new axis are presented and discussed.

1 Introduction

The increasing number of huge collections of highly structured XML documents has stimulated in recent years a wealth of research aimed at improving XML querying to both increase query languages expressiveness, and to provide an approximate matching of queries with the consequent ranking of the retrieved elements [3,18]. The first XML query languages were designed based on a data-centric view of XML repositories to allow an efficient access to complex data structures; these languages were finalized to the specification of structural constraints as well as content-related constraints (specification of exact values for XML element contents) in a Data Base style: the results produced by such constraints evaluation is a set of relevant elements. Later, several proposals appeared based on a document-centric view of XML repositories; such approaches have been classified by the information retrieval (IR) community as content-only search (CO) and content and structure search (CAS) [14]. CO approaches were mainly aimed at allowing the specification of keyword based queries in an IR style, where query evaluation produces a ranking of the retrieved XML elements [5]. CAS approaches were defined to allow the formulation of constraints on both documents content and structure [14]. CAS approaches that were based on the syntax of XPath [15], constituted a first attempt to merge the IR and DB search paradigms. Since then, the importance of merging the IR and the DB search perspectives has been widely recognized, and it has recently culminated in the W3C standard XQuery Full-Text (XQ-FT) [17] extension. The evaluation of XQ-FT queries produces a set of weighted elements, where scoring is based on a keyword based matching in textual elements. The problem of providing a ranking of XML

R. Huang et al. (Eds.): AMT 2012, LNCS 7669, pp. 307–317, 2012.

elements retrieved by a query based on both content and structural constraints has been addressed in [3], where a query relaxation technique that provides an approximate structural matching was introduced.

None of the above approaches allows users to directly specify the structural relaxations via ad-hoc predicates with a score computation. More recently, in [6], an approach to structural relaxation in XPath via new user specified constraints is proposed; a RDBMS is extended to evaluate relaxed structure matching. However the query evaluation does not provide any ranking of the retrieved fragments. As outlined in [18] and [6], querying highly structured databases or document repositories via structured query models (as XQuery is) forces the users to be well aware of the underlying structure, which is not trivial. In the above cases, users could benefit of a query language that allows a direct specification of flexible structural constraints that easily allow to require the relative position of important nodes, independently of an exact knowledge of the underlying structure(s). To achieve this aim, in this paper, we propose a formal extension of XQuery Full-Text, where two new flexible structural axes, specified by the predicates `below` and `near` are defined. The work reported in this paper was originated by a previous research where a flexible extension of the XQuery language was advocated and informally sketched in [9,7]. This is the first work that proposes the full syntax and semantics of the formal extension.

The proposed extension allows to obtain: (1) a ranking based on content predicates evaluation only (as in the original XQ-FT), (2) a ranking based on the flexible structural constraints evaluation (based on our proposal), or (3) a ranking based on a linear combination of the two above scores, which the user may also specify via the `order-by` clause, as it will be explained in the paper.

In summary, the main contributions of this paper are: (1) to define a formal extension of XQ-FT with two new flexible axes, thus allowing users to explicitly specify their tolerance to an approximate structural matching, while not forcing them to be aware of all the possible structural variations of the data/document structure; (2) to define an ad-hoc approximate matching of the flexible structural constraints thus allowing both a ranking only based on approximate structure matching, and a ranking based on a combination of content predicates and the new flexible structural predicates (while preserving a ranking based only on content predicates).

The paper is organized as follows: Section 2 reviews the research work related to introducing flexibility in XML query evaluation. Section 3 presents the proposed extension of the XQuery Full-Text language with the new flexible structural constraints: both the syntax and the semantics of the new constraints are formalized as well as some usage examples. Section 4 concludes the paper.

2 Related Work

As outlined in Section 1, several approaches to introduce some flexibility in XML retrieval have been proposed in last years, by both the database and the IR communities [8,10,12,16] In IR, the approaches to inquiry XML documents

have been classified as content-only search (CO), and both content and structure search (CAS). CAS approaches (proposed in the IR research context) consider both document content and structure in query formulation and evaluation. CAS approaches include: TopX [13], NEXI [15], TeXQuery [1], and FleXPath [4].

Most CAS approaches, like TopX [13], do not consider the content-related and the structural constraints equally important; in fact, they employ a two stage evaluation strategy by which the evaluation of content predicates is first performed (as done with CO queries), and then the obtained results are analysed, and eventually filtered out from the final result set, based on the structural constraints satisfaction. Amer-Yahia et al. [2] define some relaxations in XML structure and content querying such as the introduction of generalized data-types, the adoption of edit distances on paths, and some operations to modify the structure such as delete node, insert intermediate nodes or rename nodes. The aim of these approaches is to modify the structure specified in the query in order to relax the selection of a candidate fragment during a query evaluation. The language NEXI [15] was defined to propose a common language for CAS approaches: it is a *reduced* version of XPath where the only supported axis are the *descendant* and *self* axis. Another important XML query language is TeXQuery [1] which provides a set of full-text search features, such as Boolean connectives, phrase matching, proximity distance. TexQuery is the precursor of the XQuery Full-Text[17] language extension of single path pattern queries or more complex twig pattern queries.

FleXPath [4] is the first approach proposing an approximate matching of structural query constraints by a formalization of relaxations in the evaluation of the structure specified in the queries; it constitutes the first algebraic framework for spanning relaxations. This approach has been further developed in [3], where path scoring is formalized as an approximation of twig scoring by loosing correlations between query nodes in score computation. After the two above seminal contributions, subsequent research has addressed the problem of approximate structural matching of XML data [18]. It should be noted that all previous approaches introduce flexibility in the evaluation process of conventional queries, and therefore these approaches do not allow users to specify flexible constraints that explicitly require the application of an approximate structural matching providing fragment scores distinct from scores produced by the keyword-based evaluation. This means that the user has no way to distinguish between structural constraints in the query the evaluation of which has to produce a set of fragments, and flexible structural constraints the evaluation of which has to produce weighted fragments, with structural fragment scores distinct from content related fragment scores (usually computed by CAS approaches, XQ-FT included).

The novelty of our approach is to rely on users' specification of flexible structural constraints which require an approximate matching of XML nodes. It is worth noticing that also in the DB community a formalization of flexible structural queries was proposed in [16] and [6]; the authors defined in fact two new axes to introduce flexibility in XML structural matching. However, the above

approaches neither compute a relevance score for each matched fragment, nor offer users the possibility to combine content and structural scores in a user-defined fragment ranking as proposed in this work.

3 The Proposed Extension

In this section, we introduce the proposed extension that integrates the new predicates `below` and `near` with the XQ-FT syntax. Like the `descendant` axis, also the constraint `below` is specified as a flexible axis of a path expression: its evaluation is aimed at identifying elements (called *target nodes*) that are direct descendants of a node (called the *context node*). However, differently from the `descendant` evaluation, the `below` constraint computes a numeric score for each retrieved node. The constraint `near` is specified as a flexible axis of a path expression; it allows to identify XML elements (*target nodes*) connected to the *context node* through *any path*. Also by this axis, for each retrieved fragment a score is computed.

3.1 Flexible Structural Constraints

The main innovative characteristics of the `below` and `near` constraints are: (i) they are user-specified, and (ii) their evaluation produces a weighted set of nodes. A node weight is computed based on the node closeness to the ideal paths identified by the flexible constraints. We outline that ranking based on user specified structural constraints is a new approach to XML querying, and that, to our knowledge, no XML query engine provides this feature. The proposed approach allows to obtain a node ranking either based on structural constraints evaluation only, or based on content constraints evaluation only, or on a combination of them, which may also be decided by a user. It is important to notice that the integration of the `below` and `near` axes in the XQuery syntax allows to specify them in *any* XQuery predicate, as explained in Section 3.2.

The flexible axes can also be used in conjunction with positional predicates: as the matched elements are returned in decreasing order of relevance, the positional predicates are referred to the rank of the fragments. The following definitions, as well as the provided examples, make use of the *unabbreviated* language form where each axis is explicitly specified. For example, the query $//book/description$ is written as $/descendant :: book/child :: description$.

The Constraint "`Below`": The evaluation of the constraint `below` extends the XQuery `descendant` axis evaluation by computing, for each retrieved node, a *path relevance score* that is inversely proportional to the path distance from the ideal path identified by the `below` constraint. The ideal path is the one where the target node is a direct descendant (direct child) of the context node.

An example of the `below` evaluation is graphically sketched in Fig. 1(a): the query "`/descendant:: person/below::name`" is evaluated against the XML

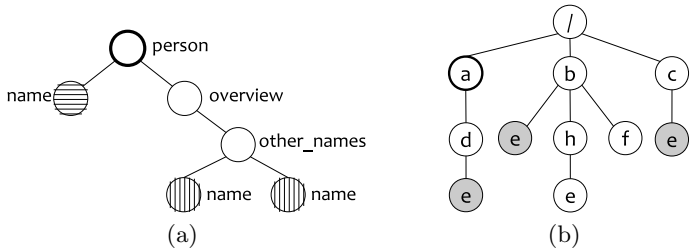

Fig. 1. (a) Graphical representation of the `below` constraint evaluation for the query: `/descendant::person/below::name` (b) The `near` constraint evaluation for the query: "`a/near(3)::e`"

document[1] fragment shown in the figure. The node labeled `person` is the *context* node for the `below` axis evaluation, while all nodes labeled `name` are the *target* nodes. The different filling of the `name` nodes indicates the ranking produced by the below constraint evaluation: the `name` element filled with horizontal lines represents the ideal element, due to the fact that it is the direct descendant of the `person` element. The nodes filled with vertical lines, instead, do not have a direct child relationship with the *context* node, and thus their score is proportional to their distance from the `person` element. Based on the example in Fig. 1(a) the elements retrieved in decreasing order of relevance estimate, are: (i) the node `person/name`; and (ii) the nodes `person/name/other_names/name` filled with vertical lines. An important observation is that, although the XQ-FT standard constraints could allow to formulate complex queries with a behavior similar to the one associated with `below`, the use of explicit flexible constraints is clearly more user-oriented and better complies with the XQ-FT scoring mechanism.

The Constraint "`Near`": The `near` constraint requires to find target nodes that are "in the neighborhoods" of the context node, in all directions (not only on the descendant axis, but also with respect to siblings and ancestors). A parameter can be specified with the `near` constraint to indicate the maximum distance between the context node and the target node: nodes reachable with more than n arcs form the context node will be excluded from the retrieved elements. The parameter allows users to control the `near` evaluation by avoiding to search in the whole XML graph for matching target nodes. The syntax of the `near` constraint is: `/near(n)::label` (based on the considered application, it could make sense to specify a default value for n). The `near` constraint evaluation computes a relevance score for each matching node: this score is inversely proportional to the distance of the target node from the context node (by considering that the ideal node is directly connected to the context node). The evaluation function of the `near` constraint is formally defined in Section 3.5.

[1] In the example the document is taken from the INEX DataCentric collection created from the IMDD movie database.

As an example, in Fig. 1(b) the evaluation of the query "`a/near(3)::e`" is shown: the node labeled `a` with bold border is the *context* node, while the `e` nodes with the filled background are the nodes matched by the example query. Note that the `e` node with path `/b/h/e` will not be retrieved because its distance from the context node is more than 3 arcs. To summarize the example in Fig. 1(b) the nodes matched by the query "`a/near(3)::e`", in decreasing order of relevance estimate, are: (i) node `/a/d/e`; and (ii) nodes `/b/e` and `/c/e`.

3.2 Syntax

The extended grammar (based on the Core XPath grammar as defined in [11]) that includes the `below` and `near` constraints is expressed in EBNF (Extended Backus–Naur Form) as follows:

```
locpath ::= '/' locpath | locpath '/' locpath | locpath '|' locpath|locstep
locstep ::= axis '::' t | locstep '[' bexpr ']'
axis     ::= xpathAxis | axisNear | axisBelow
xpathAxis::= 'self' | 'child' | 'parent' | ...
axisNear ::= 'near' | 'near(' number ')'
axisBelow::= 'below'
```

where `locpath` is the start symbol, named the *location path*; `bexpr` represents a boolean *filter expression* used as a filter for location paths; `t` denotes tag labels of document nodes; `axis` denotes the axis relations in the XPath language (`xpathAxis` represents all the standard axes, not fully listed for sake of readability) as well as the new flexible axis `near` and `below`. As outlined in Section 3.1 the `below` constraint produces the same *node-set* of the XPath axis `descendant`; however the evaluation of the `below` constraint associates a score with each node in the retrieved *node-set*. To manage such numeric scores, an optional Score Variable named `score-structure` has been introduced, in addition to the XQ-FT `score` variable, in the `for` FLWOR[2] clause. We extend the XQuery `for` clause, defined in [17], as follows:

```
ForClause     ::= "for" "$" VarName TypeDeclaration? PositionalVar?
  FTScoreVar? StructScoreVar? "in" ExprSingle ("," "$" VarName
  TypeDeclaration? PositionalVar? FTScoreVar? "in" ExprSingle)*
FTScoreVar    ::= "score" "$" VarName
StructScoreVar::= "score-structure" "$" VarName
```

where `Varname` is a variable name; `TypeDeclaration` is a variable type declaration; and `ExprSingle` is the actual query for node selection as defined in the XQuery language. An example of XQuery expression including both the Full-Text and `below` axis evaluation is the following:

[2] The FLWOR acronym stands for `For-Let-Where- OrderBy-Return` that represents the ability of the XQuery language to support selection and iterations over XML elements.

```
for $item score $scoreFT score-structure $scoreS in
  person/below:name[text() contains text "brad"] order by $scoreS
  return <i scS='{$scoreS}' scFT='{$scoreFT}'>$item</i>
```

By this query the user declares her/his interest in all nodes labeled `name` that contain the text *brad*; such nodes must have a `person` node as ancestor. The resulting `name` nodes are then ranked based on their distance from the context node `person` (as obtained by the `below` axis evaluation), and stored in the `$scoreS` variable. The results are then returned from the XQuery expression evaluation in an XML form where both structural and Full-Text scores are stored as well as the `name` node textual contents. An example of obtained results is the following:

```
1)   <i scS="1" scFT="1">Brad</i>
2)   <i scS="0.3" scFT="0.62">Brad Winsley</i>
```

A linear combination of the two scores can be provided by the user to obtain an overall ranking score. Further details on the `below` axis evaluation and scoring computation will be given in Section 3.5.

3.3 Semantics

In this section, we present the formal definition of the semantics of the proposed flexible structural constraints. An important observation here is that a key feature of the new structural constraints is the computation of a relevance score, that should be formally defined in the constraint semantics. However, in this paper, we comply to the formal definition of the XQ-FT scores semantics, where to justify the computation of scores in the evaluation of XQ-FT expressions, we introduce second-order functions. The produced scores are then used in `FLWOR` clauses to assign scoring-variables values. For further details, refer to [17, Chapter 4.4]. We extend then the semantic function S of Core XPath (as defined in [11]) by adding the `below` and the `near` semantics to define the *node-sets* retrieved by the axis constraint evaluation. In Equation (1), the semantics of the axis evaluation is provided as a reference:

$$S[\![\chi :: t]\!](N_0) := \chi(N_0) \cap T(t) \tag{1}$$

where the axis relation function $\chi : 2^{dom} \rightarrow 2^{dom}$ is defined as $\chi(N_0) = \{x \mid \exists x_0 \in X_0 : x_0 \chi x\}$ (thus overloading the χ relation name), N_0 is a set of context nodes, and a query π evaluates as $S[\![\pi]\!](N_0)$. It should be noticed that in the presented EBNF, the flexible constraints `below` and `near` can be nested without any limit in any query.

The "`Below`" Semantics: As previously outlined the definitions provided in this section are finalized to identify the set of nodes retrieved by the new axes evaluation. In this sense, the formal semantics of the constraint `below` is the same formal semantics of the `descendant` constraint, as both of them identify the same set of nodes, i.e., those having node t as a descendant of context node

N_0. In other words, both of them allow to match all the descending nodes of the context node n_0 with a given label t. The only and important difference between the `below` and the `descendant` constraints is the computation of *path relevance scores* that are produced by the `below` constraint evaluation: for each fragment matching the descendant constraint (and thus returned in the set of retrieved fragments) a score is computed, as it will be described at an operational level in section 3.5. Based on this assumption, we may then assert that:

$$S[\![below :: t]\!](N_0) = S[\![descendant :: t]\!](N_0) \tag{2}$$

As defined in Section 3.2, the `below` constraint can be inserted in any query with an unlimited nesting. The relevance score of a retrieved fragment is computed as an aggregation of all the nested `below` axes evaluation. To achieve this aim, we use the function *min* as an aggregation operator as we require that **all** constraints be satisfied. We will further address in a future work the important issue of the selection of alternative aggregation operators.

The "`Near`" Semantics: The `near` axis allows to define a maximum distance x that acts as a threshold on the number of arcs between the context node and the target node; nodes the distance of which is more then x arcs are filtered out from the possible results. Following the CoreXPath semantics introduced before, we define here below the `near` constraint semantics; also in this case (as already outlined for the `below` constraint) the score computation is formally defined as in [17]. The semantics is:

$$S[\![near(x) :: t]\!](N_0) := Near(N_0, x) \cap T(t) \tag{3}$$

where $Near(N_0, x)$ is the function that returns all nodes with a maximum distance of x from the set of nodes in N_0. As an example we present the $near(3)$ semantics. In this example the constraint `near(3)` specifies a maximum distance of 3 nodes between the context node and the target node. Here below all the matching paths of `/near(3)::l` are listed.

$$\begin{aligned} S[\![/near(3) :: l]\!](n_0) := S[\![/child :: l]\!](n_0) \quad (4)\\ \cup S[\![/child :: */child :: l]\!](n_0) \cup S[\![/child :: */child :: */child :: l]\!](n_0)\\ \cup S[\![/parent :: l]\!](n_0) \cup S[\![/parent :: */parent :: l]\!](n_0)\\ \cup S[\![/parent :: */parent :: */parent :: l]\!](n_0) \cup S[\![/parent :: */child :: l]\!](n_0)\\ \cup S[\![/parent :: */child :: */child :: l]\!](n_0) \cup S[\![/parent :: */parent :: */child :: l]\!](n_0) \end{aligned}$$

For each matching path represented in (4) a *path relevance score* is computed.

3.4 Flexible Constraints and Query Branching

An important issue related to the flexible axis evaluation concerns the aggregation of queries involving multiple flexible constraints and branching in fragment selection. The flexible constraints, as described in Section 3.1, allow to associate with each node involved in the flexible part of the query a relevance score in the range $[0, 1]$, which is used to compute a ranking of the selected fragments. While in flat queries this can be done without any difficulties, a particular observation

should be made for branching queries where the selection node that needs to be ranked appears in a different branch than the one/ones that use the flexible constraints, thus obtaining a path relevance score.

Let us consider the complex query `person[descendant::act/near(4)::title[contains(.,'gran torino')]]/child::name` and the XML document fragment shown in Fig. 2(b). Let us suppose that the user is interested in finding the names of people involved in the movie entitled "Gran Torino." The user interest is mainly in, but not limited to, people who acted in such movie: by using the constraint `near` the user requires also to find people who worked as director, producer, etc. (even if with a lower structural relevance). In Fig. 2(a), the tree representation of the query is shown: the underlined *name* element identifies the selection node; the edges between two nodes identify the *axes*-constraints (the label identifies the specified axis, i.e., `child` and `near`). Dotted lines represent filtering functions, in this example the `contains` function. The element `person` is also called *branching point*.

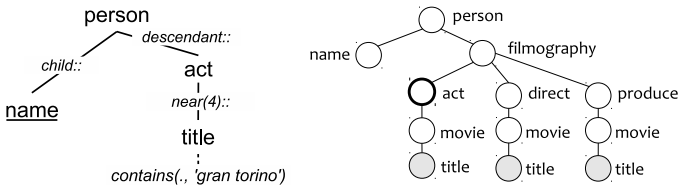

Fig. 2. (a)The tree-representation of the query `person[descendant::act/near(4)::title[contains(.,'gran torino')]]/child::name` (b) the XML fragment graph

From the above example is clear how the evaluation of the right branch of Fig. 2(a) can produce a set of elements for a single person (i.e., *Clint Eastwood* was involved in the movie as the main actor, the director and the producer). In this case, each retrieved fragment has a score associated, and it is not clear how the final score should be computed and assigned to the branching point to allow a ranking of the elements matched in the left branch of the example.

As a first reasonable solution to address this situation, we assign to the branching point element (in our example the `person` element), a score which is the maximum value among those obtained by evaluating the right branches. Although the choice of applying the `max()` aggregation is quite natural to obtain an optimistic aggregation, other aggregation schemes will be investigated.

3.5 The Proposed Approximate Evaluation

In this section, the evaluation functions of the new axes `near` and `below` are defined; each function computes the *path relevance score* of a document path with respect to the query path. Each score is in the interval $[0, 1]$ where 1 represents a full satisfaction of the axis constraint evaluation, while values less than 1 are assigned to target nodes *far* from the context node. Nodes with a path relevance score of zero will not be retrieved as they are not relevant to the user query.

As previously outlined the notion of path *closeness* is related to the concept of node distance (intended as the number of arcs connecting two nodes). Both the `near` and the `below` evaluation functions are based on a count of the of arcs between the context node and the target node to compute the path relevance score; however alternative evaluation functions can be easily implemented.

The "`Below`" Constraint Evaluation Function: As previously stated, the `below` axis produces the same *node-set* result as the XPath `descendant` axis; however, for each node, a score is computed based on the *distance* between the context node and the target node. The path relevance score for the `below` axis evaluation, with a context node c and a target node t, can be computed as :

$$w_{below}(c,t) = \frac{1}{|desc\_arcs(c,t)|}. \tag{5}$$

Where $desc\_arcs(c,t)$ is a function that, given two XML nodes c and t (where t must be a descendant of node c), returns the set of descending arcs from c to t. The score computed for the `below` axis is inversely proportional to the distance of the nodes c and t, thus giving to nodes *closer* to the context node a higher score than the one given to nodes *far* from the context node.

The "`Near`" Constraint Evaluation Function: As explained in Section 3.3, the `near` axis evaluation allows to retrieve nodes that are *close* to the given context node in every path direction; in the `near` axis evaluation the maximum allowed distance that can occur between the two nodes is taken into account.

In Equation (6), the evaluation function used to compute the *path relevance score* based on the `near` axis is defined, where c is the context node, t is the current target node, l is the maximum allowed distance and $arcs(c,e)$ is a function that returns the set of arcs in the shortest path between c and t.

$$w_{near}(c,t,l) = \begin{cases} \frac{1}{|arcs(c,t)|} & \text{if } |arcs(c,t)| \leq l \\ 0 & \text{else.} \end{cases} \tag{6}$$

Like the `below` scoring, score is inversely proportional to the distance of the context node from the target node. The function assign higher values if the target node is *closer* to the context node, while the relevance score decreases to zero as the distance increases.

4 Conclusions and Future Work

In this work, a new approach to XML querying has been presented: two new flexible axes, named `below` and `near`, have been introduced into the XQuery Full-Text language, to give users the possibility of specifying flexible structural constraints. For each axis evaluation, a *path relevance score* is computed to produce a ranked list of elements. Each retrieved XML element is evaluated based on the notion of *path closeness* between the considered element and the ideal element specified in the query. In this work, the syntax, the semantics and the axes evaluation functions have been defined, as well as an initial analysis

of the branching issue for the proposed axis evaluation. Further research will address the branching issue for the `below` and the `near` axes evaluation, as well as the definition of meaningful aggregation strategies for such cases.

Ongoing works are being conducted related to the implementation of the two new axes inside a XML Query engine, able to handle both the XQuery Full-Text specification and the flexible structural constraints `below` and `near`.

References

1. Amer-Yahia, S., Botev, C., Shanmugasundaram, J.: TeXQuery: A Full-Text Search Extension to XQuery. In: WWW 2004, pp. 583–594. ACM (2004)
2. Amer-Yahia, S., Cho, S., Srivastava, D.: Tree Pattern Relaxation. In: Jensen, C.S., Jeffery, K., Pokorný, J., Šaltenis, S., Bertino, E., Böhm, K., Jarke, M. (eds.) EDBT 2002. LNCS, vol. 2287, pp. 496–513. Springer, Heidelberg (2002)
3. Amer-Yahia, S., Koudas, N., Marian, A., Srivastava, D., Toman, D.: Structure and Content Scoring for XML. In: VLDB 2005, pp. 361–372 (2005)
4. Amer-Yahia, S., Lakshmanan, L.V.S., Pandit, S.: FleXPath: flexible structure and full-text querying for XML. In: SIGMOD 2004, pp. 83–94 (2004)
5. Amer-Yahia, S., Lalmas, M.: XML search: languages, INEX and scoring. In: SIGMOD 2006, pp. 16–23 (2006)
6. Bhowmick, S.S., Dyreson, C., Leonardi, E., Ng, Z.: Towards non-directional Xpath evaluation in a RDBMS. In: CIKM 2009, pp. 1501–1504 (2009)
7. Campi, A., Damiani, E., Guinea, S., Marrara, S., Pasi, G., Spoletini, P.: A fuzzy extension of the xpath query language. J. Intell. Inf. Syst., 285–305 (2009)
8. Chinenyanga, T.T., Kushmerick, N.: An expressive and efficient language for XML Information Retrieval. JASIST 53, 438–453 (2002)
9. Damiani, E., Marrara, S., Pasi, G.: A flexible extension of XPath to improve XML querying. In: SIGIR 2008, pp. 849–850 (2008)
10. Fuhr, N., Großjohann, K.: XIRQL: A Query Language for Information Retrieval in XML Documents. In: SIGIR, pp. 172–180 (2001)
11. Gottlob, G., Koch, C., Pichler, R.: Efficient algorithms for processing xpath queries. ACM Trans. Database Syst. 30, 444–491 (2005)
12. Theobald, A., Weikum, G.: Adding Relevance to XML. In: Suciu, D., Vossen, G. (eds.) WebDB 2000. LNCS, vol. 1997, pp. 105–124. Springer, Heidelberg (2001)
13. Theobald, M., Schenkel, R., Weikum, G.: TopX and XXL at INEX 2005. In: Fuhr, N., Lalmas, M., Malik, S., Kazai, G. (eds.) INEX 2005. LNCS, vol. 3977, pp. 282–295. Springer, Heidelberg (2006)
14. Trotman, A., Lalmas, M.: The Interpretation of CAS. In: Fuhr, N., Lalmas, M., Malik, S., Kazai, G. (eds.) INEX 2005. LNCS, vol. 3977, pp. 58–71. Springer, Heidelberg (2006)
15. Trotman, A., Sigurbjörnsson, B.: Narrowed Extended XPath I (NEXI). In: Fuhr, N., Lalmas, M., Malik, S., Szlávik, Z. (eds.) INEX 2004. LNCS, vol. 3493, pp. 16–40. Springer, Heidelberg (2005)
16. Truong, B.Q., Bhowmick, S.S., Dyreson, C.: SINBAD: Towards Structure-Independent Querying of Common Neighbors in XML Databases. In: Lee, S.-g., Peng, Z., Zhou, X., Moon, Y.-S., Unland, R., Yoo, J. (eds.) DASFAA 2012, Part I. LNCS, vol. 7238, pp. 156–171. Springer, Heidelberg (2012)
17. W3C. XQuery/XPath FullText (March 2011), `http://www.w3.org/TR/xpath-full-text-10`
18. Yu, C., Jagadish, H.V.: Querying Complex Structured Databases. In: VLDB 2007, pp. 1010–1021 (2007)