Dynamic Filtering of Ranked Answers when Evaluating Fuzzy XPath Queries

Jesús M. Almendros-Jiménez[1], Alejandro Luna[2], and Ginés Moreno[2]

[1] Dep. of Languages and Computation, University of Almería, Spain
Email: jalmen@ual.es
[2] Dep. of Computing Systems, University of Castilla-La Mancha, Spain
Emails: Gines.Moreno@uclm.es, Alejandro.Luna@alu.uclm.es

Abstract. We have recently designed an extension of the XPath language which provides ranked answers to flexible queries taking profit of fuzzy variants of *and*, *or* and *avg* operators for XPath conditions, as well as two structural constraints, called *down* and *deep*, for which a certain degree of relevance is associated. In practice, this degree is very low for some answers weakly accomplishing with the original query, and hence, they should not be computed in order to alleviate the computational complexity of the information retrieval process. In this work we focus on the scalability of our interpreter for dealing with massive XML files by making use of its ability for prematurely disregarding those computations leading to non significant solutions (i.e., with a poor degree of relevance according the preferences expressed by users when using the new command FILTER). Since our proposal has been implemented with a fuzzy logic language, here we exploit the high expressive resources of this declarative paradigm for performing "dynamic thresholding" in a very natural and efficient way, thus connecting with the so-called *top-k answering problem*, which is very well-known in the fuzzy logic and soft computing arena.

Key words: Information Retrieval Systems; Fuzzy XPath; Information Filtering Systems; Fuzzy Filtering & Thresholding; Fuzzy Logic Programming

1 Introduction

The *XPath* language [7] has been proposed as a standard for XML querying and it is based on the description of the path in the XML tree to be retrieved. XPath allows to specify the name of nodes (i.e., tags) and attributes to be present in the XML tree together with boolean conditions about the content of nodes and attributes. XPath querying mechanism is based on a boolean logic: the nodes retrieved from an XPath expression are those matching the path of the XML tree. Therefore, the user should know the *XML schema* in order to specify queries. However, even when the XML schema exists, it can not be available for users. Moreover, XML documents with the same XML schema can be very different in structure. Let us suppose the case of XML documents containing the curriculum

vitae of a certain group of persons. Although they can share the same schema, each one can decide to include studies, jobs, training, etc. organized in several ways: by year, by relevance, and with different nesting degree.

Therefore, in the context of semi-structured databases, *flexible query languages* arise for allowing the formulation of queries without taking into account a rigid database schema, usually including too mechanisms for obtaining a certain *ranked list* of answers. The ranking of answers can provide *satisfaction degree* depending on several factors. In a structural XPath-based query, the main criteria to provide a certain degree of satisfaction depends on the *hierarchical deepness* and *document order*. Therefore the query language should provide mechanisms for giving *priority* to answers when they occur in different parts of the document. In this sense, the need for providing flexibility to XPath has recently motivated the investigation of extensions of the XPath language. We can distinguish those in which the main goal is the introduction of fuzzy information in data (similarity, proximity, vagueness, etc) [9, 16, 30, 26] and the proposals in which the main goal is the handling of crisp information by fuzzy concepts [10, 13, 15, 14, 20]. Our work focuses on the second line of research.

In [10, 13] authors introduce in XPath flexible matching by means of fuzzy constraints called *close* and *similar* for node content, together with *below* and *near* for path structure. In addition, they have studied *deep-similar* notion for tree matching. In order to provide ranked answers they adopt a *Fuzzy set theory*-based approach in which each answer has an associated numeric value (the membership degree). The numeric value represents the *Retrieval Status Value (RSV)* of the associated item. In the work of [15], they propose a satisfaction degree for XPath expressions based on associating a degree of importance to XPath nodes, and they study how to compute the best k answers. In both cases, authors allow the user to specify in the query the degree in which the answers will be penalized. On the other hand, in [14], they have studied how to relax XPath queries by means of rewriting in order to improve information retrieval in the presence of heterogeneous data resources. Our proposal also connects with the recent approaches of [27, 28] but, as we are going to see, it is important to note that many of our fuzzy commands are directly inspired by the powerful expressive resources of the underlying fuzzy logic language used for implementing our tool.

As we will resume in Section 2, in [3–6] we have presented both an interpreter and a debugger coping with an extension of the XPath query language for managing flexible queries in a very natural way (the tool can be tested on-line via `http://dectau.uclm.es/fuzzyXPath/`). Our approach proposes two structural constraints called *down* and *deep* for which a certain degree of relevance can be associated. In such a way that *down* provides a ranked set of answers depending on the path is found from "top to down" in the XML document, and *deep* provides a set of answers depending on the path is found from "left to right" in the XML document. Both structural constraints can be combined. In addition, we provide fuzzy operators *and*, *or* and *avg* for XPath conditions. In this way, users can express the priority they give to answers. Such fuzzy operators can be combined to provide ranked answers. Our approach has been implemented with the so-called

«*Multi-Adjoint Logic Programming*» language (MALP in brief) [22] by using our «*Fuzzy LOgic Programming Environment for Research*» $\mathcal{FLOPER}$ [23–25], which can be freely downloaded from `http://dectau.uclm.es/floper/`.

We wish to remark now that our proposal is an extension of previous works about the implementation of XPath by means of logic programming [2], which has been extended to XQuery in [1]. The new extension follows the same encoding proposed in [1] in which a predicate called *xpath* is defined by means of PROLOG rules, which basically traverse the PROLOG representation of the XML tree by means of a PROLOG list. In order to implement *Fuzzy-XPath* by means of $\mathcal{FLOPER}$ we proceed similarly to the PROLOG implementation of XPath, but proposing a new (fuzzy) predicate called *fuzzyXPath* implemented in MALP. The new query language returns a set of ranked answers each one with an associated RSV. Such RSV is computed by easily using MALP rules (thus exploiting the correspondences between the languages *for-being* and *to-be* implemented), where the notion of *RSV* is modeled inside a multi-adjoint lattice, and usual fuzzy connectives of the MALP language act as ideal resources to represent new flexible XPath operators.

As we will see in Section 3, the main goal of this paper consists in the introduction of a new fuzzy command inside *Fuzzy-XPath* which comfortably relies on our implementation based on fuzzy logic programming. So, when «[FILTER=r]» precedes a fuzzy query, the interpreter *lazily* explores an input XML document for dynamically disregarding as soon as possible those branchs of the XML tree leading to irrelevant solutions with an RSV degraded below r, thus allowing the possibility of efficiently managing large files without reducing the set of answers for which users are mainly interested in.

2 A Fuzzy Extension of XPath

Following [3–6], our flexible XPath is defined by means of the following rules:

```
    xpath :=  [deepdown]path
     path :=  literal | text() | node | @att |
              node/path | node//path
     node :=  QName | QName[cond]
     cond :=  path op path
 deepdown :=  DEEP=degree, DOWN=degree
       op :=  > | = | < | and | or | avg
```

Basically, our fuzzy proposal extends XPath as follows:

- A given XPath expression can be adorned with «[DEEP = r_1, DOWN = r_2]» which means that the *deepness* of elements is penalized by r_1 and that the *order* of elements is penalized by r_2, and such penalization is proportional to the distance. In particular, «[DEEP = 1, DOWN = r_2]» can be used for penalizing only w.r.t. document order. $DEEP$ works for //, and $DOWN$ works for / and //.

```
<bib>
   <book year="2001" price="45.95">
      <title>Don Quijote de la Mancha</title>
      <author>Miguel de Cervantes Saavedra</author>
      <publications> <book year="1997" price="35.99">
                     <title>La Galatea</title>
                     <author>Miguel de Cervantes Saavedra</author>
                     <publications>
                              <book year="1994" price="25.99">
                              <title>Los trabajos de Persiles y Segismunda</title>
                              <author>Miguel de Cervantes Saavedra</author></book>
                     </publications></book>
      </publications></book>
   <book year="1999" price="25.65">
      <title>La Celestina</title>
      <author>Fernando de Rojas</author></book>
   <book year="2005" price="29.95">
      <title>Hamlet</title>
      <author>William Shakespeare</author>
      <publications>
         <book year="2000" price="22.5">
            <title>Romeo y Julieta</title>
            <author>William Shakespeare</author></book>
      </publications></book>
   <book year="2007" price="22.95">
      <title>Las ferias de Madrid</title>
      <author>Felix Lope de Vega y Carpio</author>
      <publications>
         <book year="1996" price="27.5">
            <title>El remedio en la desdicha</title>
            <author>Felix Lope de Vega y Carpio</author> </book>
         <book year="1998" price="12.5">
            <title>La Dragontea</title>
            <author>Felix Lope de Vega y Carpio</author></book>
      </publications></book>
</bib>
```

Fig. 1. Input XML document in our examples

- Moreover, the classical *and* and *or* connectives admit here a fuzzy behavior based on fuzzy logic, i.e., assuming two given *RSV*'s r_1 and r_2, operator *and* is defined as $r_3 = r_1 * r_2$ and operator *or* returns $r_3 = r_1 + r_2 - (r_1 * r_2)$. In addition, the *avg* operator is defined as $r_3 = (r_1 + r_2)/2$.

In general, an extended XPath expression defines, w.r.t. a XML document, a sequence of subtrees of the XML document where each subtree has an associated RSV. XPath conditions, which are defined as fuzzy operators applied to XPath expressions, compute a new RSV from the RSVs of the involved XPath expressions, which at the same time, provides a RSV to the node. In order to illustrate these explanations, let us see some examples of our proposed fuzzy version of XPath according to the XML document shown in Figure 1 whose *skeleton* is depicted in Figure 2.

Example 1. Suppose the XPath query: « [DEEP=0.9,DOWN=0.8]//title », that requests *title*'s penalizing the occurrences from the document root by a proportion of 0.9 and 0.8 by nesting and ordering, respectively, and for which we obtain

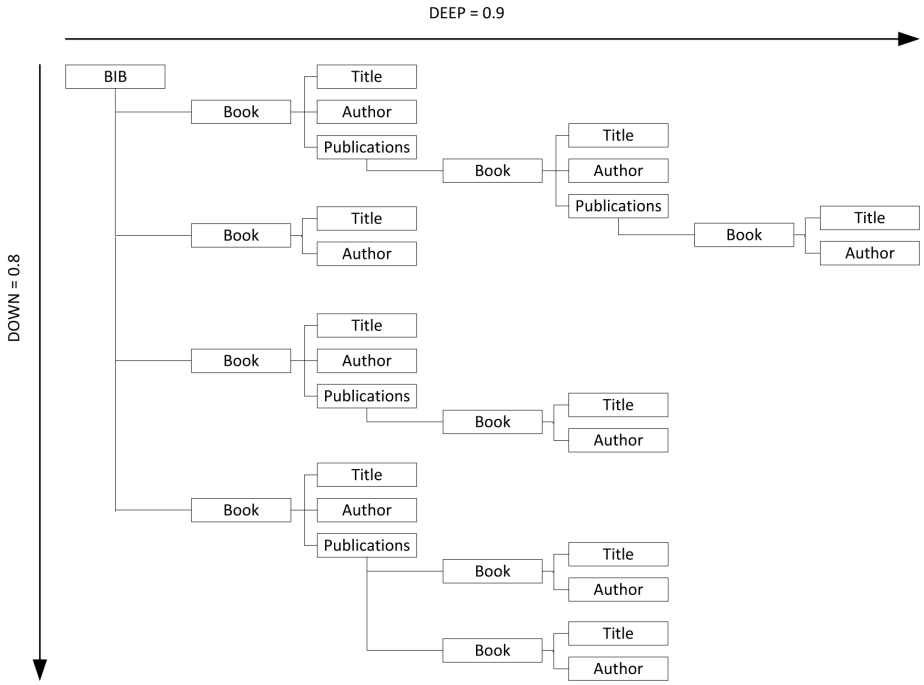

Fig. 2. XML skeleton represented as a tree

Document	**RSV computation**
<result>	
<title **rsv**="0.81">Don Quijote de la Mancha</title>	$0.81 = 0.9^2$
<title **rsv**="0.6561">La Galatea</title>	$0.6561 = 0.9^4$
<title **rsv**="0.531441">Los trabajos de Persiles y ...</title>	$0.531441 = 0.9^6$
<title **rsv**="0.648">La Celestina</title>	$0.648 = 0.9^2 * 0.8$
<title **rsv**="0.5184">Hamlet</title>	$0.5184 = 0.9^2 * 0.8^2$
<title **rsv**="0.419904">Romeo y Julieta</title>	$0.419904 = 0.9^4 * 0.8^2$
<title **rsv**="0.41472">Las ferias de Madrid</title>	$0.41472 = 0.9^2 * 0.8^3$
<title **rsv**="0.3359232">El remedio en la desdicha</title>	$0.3359232 = 0.9^4 * 0.8^3$
<title **rsv**="0.26873856">La Dragontea</title>	$0.26873856 = 0.9^4 * 0.8^4$
</result>	

Fig. 3. Output of a query using *DEEP/DOWN*

Document	**RSV computation**
<result>	
<book **rsv**="0.5" ...> <title>Don Quijote ...</title> ...</book>	$0.5 = 0 + 1/2$
<book **rsv**="1.0"...><title>La Celestina</title> ...</book>	$1 = 1 + 1/2$
<book **rsv**="1.0" ...><title>Hamlet</title> ...</book>	$1 = 1 + 1/2$
<book **rsv**="0.5" ...><title>Las ferias de Madrid</title> ...</book>	$0.5 = 1 + 0/2$
</result>	

Fig. 4. Output of a query using *AVG*

the file listed in Figure 3. In such document we have included as attribute of each subtree, its corresponding RSV. The highest RSVs correspond the main

Document	RSV computation
<result>	
<title **rsv**="0.3645">La Galatea</title>	$0.3645 = 0.9^3 * 1/2$
<title **rsv**="0.295245">Los trabajos de Persiles y... </title>	$0.295245 = 0.9^5 * 1/2$
<title **rsv**="0.72">La Celestina</title>	$0.72 = 0.9 * 0.8 * 1$
<title **rsv**="0.288">Hamlet</title>	$0.288 = 0.9 * 0.8^2 * 1/2$
<title **rsv**="0.2304">Las ferias de Madrid</title>	$0.2304 = 0.9 * 0.8^3 * 1/2$
<title **rsv**="0.2985984">El remedio en la desdicha</title>	$0.2985984 = 0.9^3 * 0.8^4 * 1$
<title **rsv**="0.11943936">La Dragontea</title>	$0.11943936 = 0.9^3 * 0.8^5 * 1/2$
</result>	

Fig. 5. Output of a query using all operators

Records	FILTER								
	0.1	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**
1000	1.766	1.696	1.734	0.842	0.469	0.268	0.221	0.087	0.056
2000	6.628	6.432	6.998	3.242	1.439	0.677	0.599	0.168	0.122
3000	14.532	14.023	14.059	6.306	2.831	1.257	1.101	0.253	0.179
4000	25.535	24.684	24.722	10.883	4.827	1.918	1.794	0.345	0.242
5000	41.522	37.782	37.166	16.201	7.242	2.993	2.516	0.427	0.281
6000	58.905	55.354	55.596	24.411	10.993	4.207	3.554	0.554	0.373
7000	85.167	85.652	82.733	37.748	14.436	5.083	4.653	0.649	0.460
8000	137.737	102.816	102.763	69.401	26.680	8.273	5.894	0.690	0.481
9000	175.272	131.828	131.021	56.937	22.601	7.869	7.329	0.824	0.549
10000	195.613	185.201	167.676	95.286	26.649	9.516	9.595	0.973	0.742

Fig. 6. Performance of *Fuzzy-XPath* by using FILTER on XML files with growing sizes

book's of the document, and the lowest RSVs represent the *book*'s occurring in nested positions (those annotated as related *publication*'s).

Example 2. Figure 4 shows the answer associated to the XPath expression: « /bib/book[@price<30 avg @year<2006] ». Here we show that books satisfying a *price* under 30 and a *year* before 2006 have the highest RSV.

Example 3. Finally, combining all operators «[DEEP=0.9,DOWN=0.8] //book [(@price>25 and @price<30) avg (@year<2000 or @year>2006)]/title», the RSV values are more scattered, as shown in Figure 5.

3 Using Filters for the Dynamic Thresholding of Queries

In [19, 18] we have reported some *thresholding* techniques specially tailored for the MALP language, where the main idea consists in to dynamically create and evaluate filters for prematurely disregarding those superfluous computations leading to non-significant solutions. Somehow inspired by the same guidelines, we have recently equipped our fuzzyXPath interpreter with a new command with syntax «[FILTER=r]» (being r a real number between 0 and 1) which can be used just at the beginning of a query for indicating that only those answers with RSV greater of equal than r must be generated and reported.

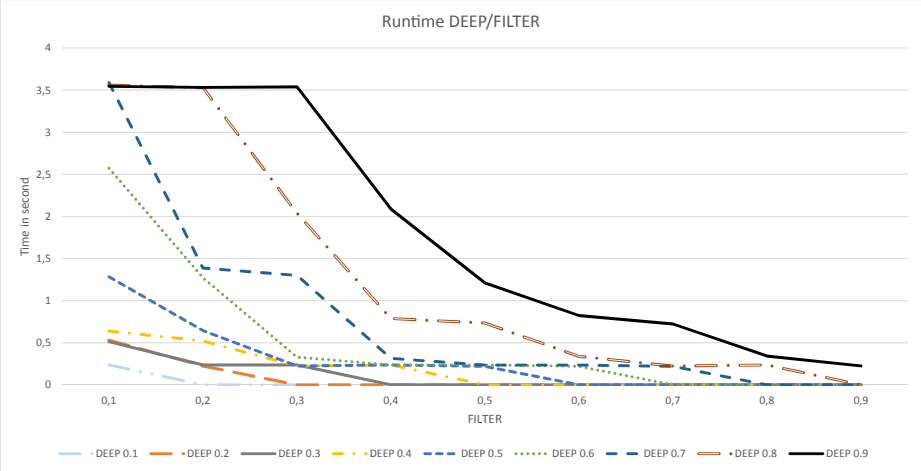

Fig. 7. Runtime for several *Fuzzy-XPath* queries varying DEEP and FILTER

So, if we consider a *Fuzzy-XPath* query with the following form «[FILTER=0.4]//book[@year<2000 avg @price<50]/title», we obtain nine answers, but only five if we fix «[FILTER=0.8]». Obviously, we would hope that the runtime of the second case should be lower than the first one since, as our approach does, there is no need for computing all solutions and then filtering the best ones. This desired dynamic behaviour when avoiding useless computations is reflected in Figure 6 which considers the effort needed for executing (excluding parsing/compiling time) a query like «[FILTER=r]//book[(@price>25 and @price<30) avg (@year<2000 or @year>2006)]»where each row represents the size of several XML files accomplishing with the same structure of our running example (but considering different nesting levels of tags `book`, `title`, `author` and `publications`), and each column refers to a different degree of the FILTER command. Here, the runtime is measured in seconds (the benchmarks have been performed using a computer with processor Intel Core Duo, with 2 GB RAM and Windows Vista) and each record in the input file refers to a different `book` (that is, the number of records coincides with the number of occurrences of tag `book`) which might contain other books inside its `publications` tag.

Moreover, in Figure 7 we continue with a similar query to the previous one, but also considering the DEEP command[3]. Here, for a large XML document with a fixed size, we express the number of seconds needed for executing such query when varying FILTER and DEEP, where it is easy to see that the behaviour is more and more improved whenever FILTER grows and DEEP decreases, as wanted. Note that the previous query makes use of the *avg* command

[3] This kind of statistics can be produced on-line for several XML files and *Fuzzy-XPath* queries via the following URL that we have just prepared for the interested reader: `http://dectau.uclm.es/fuzzyXPath/fuzzyXPathEstatistic2.php#testing`.

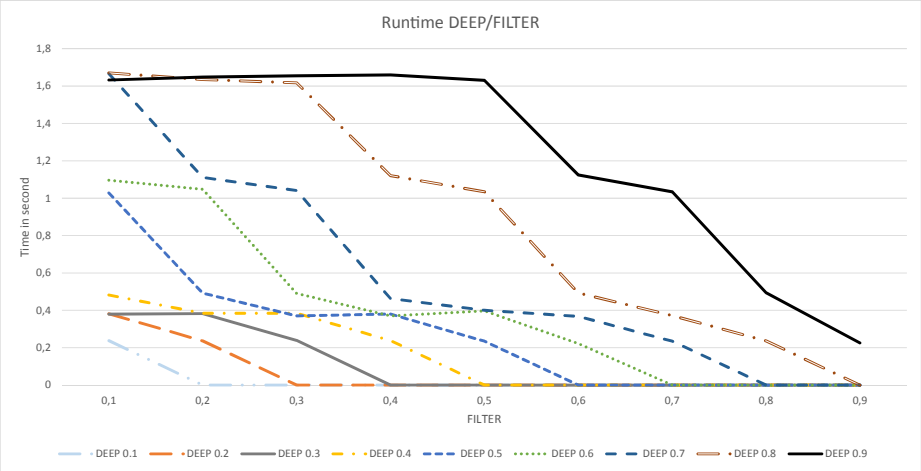

Fig. 8. Varying DEEP and FILTER in a query using $avg\{30, 1\}$

and remember that its behaviour is defined, for two given *RSV*'s r_1 and r_2, as $r_3 = (r_1 + r_2)/2$. We have recently conceived a *priorized version* of such operator which let us to give different degrees of importance to its arguments. In general, $avg\{p_1, p_2\}$ is computed by $r_3 = (r_1 * p_1 + r_2 * p_2)/(p_1 + p_2)$ and hence, if in the previous query we use $avg\{30, 1\}$ instead of standard avg, we indicate that the first sub-condition (i.e., @price>25 and @price<30) is 30 times more important than the second one (i.e., @year<2000 or @year>2006), whereas $avg\{1, 30\}$ represent the inverse criterium. In Figures 8 and 9 we provide statistics in the same way than in Figure 7, but using now *average with priorities* 30-1 and 1-30, respectively.

Although the core of our application is written with (fuzzy) MALP rules, our implementation is based on the following items:

(1) We have reused/adapted several modules of our PROLOG-based implementation of (crisp) XPath described in [1, 2].
(2) We have used the SWI-PROLOG library for loading XML files, in order to represent a XML document by means of a PROLOG term[4].
(3) The parser of XPath has been extended to recognize the new keywords `FILTER`, `DEEP`, `DOWN`, `avg`, etc... with their proper arguments.
(4) Each tag is represented as a data-term of the form: `element(Tag, Attributes, Subelements)`, where `Tag` is the name of the XML tag, `Attributes` is a PROLOG list containing the attributes, and `Subelements` is a PROLOG list containing the sub-elements (i.e. sub-trees) of the tag. For instance, the document of Figure 1 is represented in SWI-PROLOG like in Figure 10. Loading of documents is achieved by predicate `load_xml(+File,-Term)` and writing by predicate `write_xml(+File,+Term)`.

[4] The notion of *term* (i.e., data structure) is just the same in MALP and PROLOG.

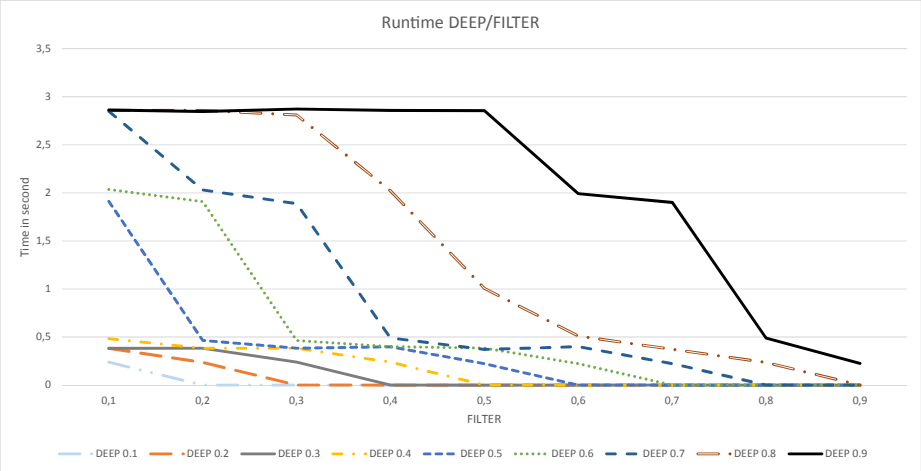

Fig. 9. Varying DEEP and FILTER in a query using $avg\{1, 30\}$

```
[element(bib,[],
    [element(book,[year=2001,price=45.95],
             [element(title,[],[Don Quijote de la Mancha]),
             element(author,[],[Miguel de Cervantes Saavedra]),
             element(publications,[],
                          [element(book,[year=1997,price=35.99],
                                   [element(title,[],[La Galatea]),
                                   element(author,[],[Miguel de Cervantes Saavedra]),
                                element(publications,[],...])...]),])])
```

Fig. 10. A data-term representing a XML document

(5) Predicate `fuzzyXPath(+ListXPath,+Tree,+Deep,+Down,+Filter,+Accum)` receives six arguments: (1) `ListXPath` is the PROLOG representation of a XPath expression; (2) `Tree` is the term representing an input XML document; (3) `Deep/Down/Filter` have the obvious meaning, and finally (4) the last argument `Accum` (which is appropriately updated -maybe decreased- when going deeper in the exploration of the file) accumulates the sequence of penalties produced till reaching a concrete node, and it is very useful for deciding when performing a recursive call to the children of such node whenever the value of `Accum` is better than the one fixed by `Filter`.

(6) The evaluation of the query generates a *truth value* which has the form of a tree, called *tv tree*. For instance, the query shown in Example 1, generates the one illustrated in Figure 11. The main power of a fuzzy logic programming language like MALP w.r.t. PROLOG, is that instead of answering questions with a simple *true/false* value, solutions are reported in a much more tinged, documented way. Basically, the `fuzzyXPath` predicate traverses the PROLOG tree representing a XML document annotating into the *tv tree* the corresponding *deep/down* values according to the movements performed

in the horizontal and vertical axis, respectively. In addition, the *tv tree* is annotated with the values of *and*, *or* and *avg* operators in each node.

(7) Finally, the *tv tree* is used for computing the output of the query, by multiplying the recorded values. A predicate called `tv_to_elem` has been implemented to output the answer in a pretty way.

```
tv(1, [[],
   tv(0.9,[[],
      tv(0.9,[element(title,[],[Don Quijote de la Mancha]),[],
      tv(1,[[],[],
      tv(1,[[],
          tv(0.9,[[],
              tv(0.9,[element(title,[],[La Galatea]),[],
              tv(1,[[],[],
              tv(1,[[],
                  tv(0.9,[[],
                      tv(0.9,[element(title,[],[Los trabajos de Persiles..]),...]),
      tv(0.8,[[],
         tv(0.9,[element(title,[],[La Celestina]),[],[]]),...
```

Fig. 11. Example of a MALP output

4 Conclusions and Future Work

In [3–6] we have recently enriched XPath with new constructs (both structural -*deep* and *down*- and constraints -*avg* and fuzzy versions of classical *or*/*and* operators-) in order to flexibly query XML documents. This paper has highlighted the benefits of using a new fuzzy command for filtering the set of ranked answers in a dynamic way, in order to reduce the runtime and complexity of computations when dealing with large files. Our approach represents the first real-world application developed with the fuzzy logic language MALP, for which we have recently developed some thresholding tabulation techniques[5] [19, 18]. All these actions will be very useful for addressing in our framework the well-known "top-k ranking problem" (i.e. determining the top k answers to a query without computing the -usually wider, possibly infinite- whole set of solutions, which is strongly related with the FILTER command reported along this paper) inspired by [8, 11, 12, 21, 29, 17].

Acknowledgements

We are grateful to anonymous reviewers for providing us valuable suggestions which have been used for improving the material compiled so far. This work was supported by the EU (FEDER), and the Spanish MINECO Ministry (*Ministerio de Economía y Competitividad*) under grant TIN2013-45732-C4-2-P, as well as by the Andalusian Regional Government (Spain) under Project P10-TIC-6114.

[5] We are now implementing into $\mathcal{FLOPER}$ this highly efficient procedural mechanism.

References

1. J. M. Almendros-Jiménez. An Encoding of XQuery in Prolog. In *Proceedings of the Sixth International XML Database Symposium XSym'09*, pages 145–155, Heildelberg,Germany, 2009. Springer, LNCS 5679.
2. J. M. Almendros-Jiménez, A. Becerra-Terón, and Francisco J. Enciso-Baños. Querying XML documents in logic programming. *Theory and Practice of Logic Programming*, 8(3):323–361, 2008.
3. J.M. Almendros-Jiménez, A. Luna, and G. Moreno. A Flexible XPath-based Query Language Implemented with Fuzzy Logic Programming. In *Proc. of 5th International Symposium on Rules: Research Based, Industry Focused, RuleML'11*, pages 186–193. Springer Verlag, LNCS 6826, 2011.
4. J.M. Almendros-Jiménez, A. Luna, and G. Moreno. Fuzzy Logic Programming for Implementing a Flexible XPath-based Query Language. *Electronic Notes on Theoretical Computer Science, ENTCS*, 282:3–18, 2012.
5. J.M. Almendros-Jiménez, A. Luna, and G. Moreno. Annotating Fuzzy Chance Degrees when Debugging Xpath Queries. In *Proc. of the 12th International Work-Conference on Artificial Neural Networks, IWANN'13*, pages 300–311. Springer Verlag, LNCS 7903, Part II, 2013.
6. J.M. Almendros-Jiménez, A. Luna, G. Moreno, and C. Vázquez. Analyzing Fuzzy Logic Computations with Fuzzy XPath. In *Proc. of PROLE'13*, pages 136–150 ("work in progress" track, extended version submitted to ECEASST). Universidad Complutense de Madrid (ISBN: 978-84-695-8331-9), 2013.
7. A. Berglund, S. Boag, D. Chamberlin, M.F. Fernandez, M. Kay, J. Robie, and J. Siméon. XML path language (XPath) 2.0. *W3C*, 2007.
8. Nicolas Bruno, Surajit Chaudhuri, and Luis Gravano. Top-k selection queries over relational databases: Mapping strategies and performance evaluation. *ACM Trans. Database Syst.*, 27(2):153–187, 2002.
9. P. Buche, J. Dibie-Barthélemy, O. Haemmerlé, and G. Hignette. Fuzzy semantic tagging and flexible querying of XML documents extracted from the Web. *Journal of Intelligent Information Systems*, 26(1):25–40, 2006.
10. A. Campi, E. Damiani, S. Guinea, S. Marrara, G. Pasi, and P. Spoletini. A fuzzy extension of the XPath query language. *Journal of Intelligent Information Systems*, 33(3):285–305, 2009.
11. Kevin Chen-Chuan Chang and Seung won Hwang. Minimal probing: supporting expensive predicates for top-k queries. In Michael J. Franklin, Bongki Moon, and Anastassia Ailamaki, editors, *SIGMOD Conference*, pages 346–357. ACM, 2002.
12. Surajit Chaudhuri, Luis Gravano, and Amélie Marian. Optimizing top-k selection queries over multimedia repositories. *IEEE Trans. Knowl. Data Eng.*, 16(8):992–1009, 2004.
13. E. Damiani, S. Marrara, and G. Pasi. FuzzyXPath: Using fuzzy logic and IR features to approximately query XML documents. *Foundations of Fuzzy Logic and Soft Computing*, pages 199–208, 2007.
14. B. Fazzinga, S. Flesca, and F. Furfaro. On the expressiveness of generalization rules for XPath query relaxation. In *Proceedings of the Fourteenth International Database Engineering & Applications Symposium*, pages 157–168. ACM, 2010.
15. B. Fazzinga, S. Flesca, and A. Pugliese. Top-k Answers to Fuzzy XPath Queries. In *Database and Expert Systems Applications*, pages 822–829. Springer, 2009.
16. A. Gaurav and R. Alhajj. Incorporating fuzziness in XML and mapping fuzzy relational data into fuzzy XML. In *Proceedings of the 2006 ACM symposium on Applied computing*, pages 456–460. ACM, 2006.

17. Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4), 2008.
18. P. Julián, J. Medina, P.J. Morcillo, G. Moreno, and M. Ojeda-Aciego. An unfolding-based preprocess for reinforcing thresholds in fuzzy tabulation. In *Proc. of the 12th International Work-Conference on Artificial Neural Networks, IWANN'13*, pages 647–655. Springer Verlag, LNCS 7902, Part I, 2013.
19. P. Julián, J. Medina, G. Moreno, and M. Ojeda. Efficient thresholded tabulation for fuzzy query answering. *Studies in Fuzziness and Soft Computing (Foundations of Reasoning under Uncertainty)*, 249:125–141, 2010.
20. H.G. Li, S.A. Aghili, D. Agrawal, and A. El Abbadi. FLUX: fuzzy content and structure matching of XML range queries. In *Proceedings of the 15th international conference on World Wide Web*, pages 1081–1082. ACM, 2006.
21. Amélie Marian, Nicolas Bruno, and Luis Gravano. Evaluating top-k queries over web-accessible databases. *ACM Trans. Database Syst.*, 29(2):319–362, 2004.
22. J. Medina, M. Ojeda-Aciego, and P. Vojtáš. Similarity-based Unification: a multi-adjoint approach. *Fuzzy Sets and Systems*, 146:43–62, 2004.
23. P.J. Morcillo and G. Moreno. Programming with Fuzzy Logic Rules by using the FLOPER Tool. In Nick Bassiliades et al., editor, *Proc of the 2nd. Rule Representation, Interchange and Reasoning on the Web, International Symposium, RuleML'08*, pages 119–126. Springer Verlag, LNCS 3521, 2008.
24. P.J. Morcillo, G. Moreno, J. Penabad, and C. Vázquez. A Practical Management of Fuzzy Truth Degrees using FLOPER. In M. Dean et al., editor, *Proc. of 4nd Intl Symposium on Rule Interchange and Applications, RuleML'10*, pages 20–34. Springer Verlag, LNCS 6403, 2010.
25. G. Moreno and C. Vázquez. Fuzzy logic programming in action with floper. *Journal of Software Engineering and Applications*, 7:237–298, 2014.
26. B. Oliboni and G. Pozzani. An XML Schema for Managing Fuzzy Documents. *Soft Computing in XML Data Management*, pages 3–34, 2010.
27. Emanuele Panzeri and Gabriella Pasi. An approach to define flexible structural constraints in xquery. In *Proc. of 8th International Conference on Active Media Technology, AMT'12*, pages 307–317. Springer Verlag, LNCS 7669, 2013.
28. Emanuele Panzeri and Gabriella Pasi. Flex-basex: an xml engine with a flexible extension of xquery full-text. In *Proc. of the 36th International ACM SIGIR conference on research and development in Information Retrieval, SIGIR'13, (http://doi.acm.org/10.1145/2484028.248421)*, pages 1038–1084. ACM, 2013.
29. Christopher Re, Nilesh N. Dalvi, and Dan Suciu. Efficient top-k query evaluation on probabilistic data. In Rada Chirkova, Asuman Dogac, M. Tamer Özsu, and Timos K. Sellis, editors, *ICDE*, pages 886–895. IEEE, 2007.
30. L. Yan, ZM Ma, and J. Liu. Fuzzy data modeling based on XML schema. In *Proceedings of the 2009 ACM symposium on Applied Computing*, pages 1563–1567. ACM, 2009.