© 2012 **DSRC**

A COMPLETE SEARCH ENGINE FOR EFFICIENT AND DATA INTEGRATION USING FUZZY SEARCH

S Maheshwara Prasad 1*, M.Rajasekhar 2*

1. *Research scholar, Avr and Svr Engineering College, Kurnool*
2. *Asst Prof, Avr and Svr Engineering College, Kurnool.*

Abstract: *As the next generation of the Web language, XML is straightforwardly usable, which has been the de-facto standard of information representation and exchange over the Web. XML employs a tree-structured data model, and XML queries specify patterns of selection predicates on multiple elements related by a tree structure. Due to increase in web-based applications, searching for all occurrences of a twig pattern in XML documents becomes an important operation in XML query processing. A number of algorithms have been proposed to solve these twig pattern queries. However, less fulfill the requirements of managing fuzzy XML twig pattern queries. In this paper, we present an algorithm called FTwig, which adopts the region encoding scheme of structure relations to process fuzzy twig pattern queries, for matching an XML query twig pattern. Based on the fuzzy set and possibility distribution theory, we also introduce a set of primitive fuzzy XML algebraic operations.*

1. INTRODUCTION

With the prompt development of the Internet, the requirement of managing information based on the Web becomes more and more important. XML is rapidly emerging and has been the de-facto standard for exchanging data on the Web. XML data are often represented as tree models, and common XML query languages, such as XPath and XQuery, issue structural queries over the XML data. An XML query is typically formed as a twig pattern with predicates additionally imposed on the contents or attribute values of the tree nodes. Due to their significance to many practical applications, efficient processing of such structural queries [2, 3, 9, 11] has received significant attentions from both academic and industrial communities. At the same time, information is often vague or ambiguous in the real world applications. Some data are inherently fuzzy since their values are subjective in the real applications. For example, consider values representing the satisfaction degree for a film, different person may have different satisfaction degree. Consequently, finding all occurrences of a fuzzy twig pattern in XML database is a core operation in fuzzy XML query processing

The basic strategy is to first develop a fuzzy labeling scheme to capture the structural information of fuzzy XML documents, and then perform twig pattern matching based on the labels. Traditional methods for designing appropriate these labeling schemes, the structural relationships between two elements in XML documents can be determined from their corresponding labels. Unfortunately, due to lack of effective fuzzy labeling scheme on the web, XML cannot be used in processing fuzzy data query although querying with probabilistic and incomplete information [6, 7, 8, 9, 13] have been discussed extensively.

**** S Maheshwara Prasad***
Research scholar, Avr and Svr Engineering College, Kurnool.

Efficient matching of twig pattern queries over XML data is one of the most fundamental challenges for processing XML queries. Previous researches build various structural indexes for XML documents and expect to use them to accelerate the query processing time. However, less fulfill the requirements of managing fuzzy XML twig pattern queries. If we wanted to manage the data in fuzzy XML databases, it is clear that we would need methods to process fuzzy twig queries in XML.

In this paper, we propose a novel fuzzy labeling scheme to capture the structural information of fuzzy XML documents. Based on the fuzzy set [16] and possibility distribution theory, we also introduce a set of primitive fuzzy XML algebraic operations. Finally, we propose an algorithm "FTwig" for fuzzy twig pattern query. A central contribution of this paper is the framework for fuzzy twig pattern queries that is believed to be one of the foundations of implementing web-based intelligent information management.

The rest of the paper is organized as follows. After a discussion of fuzzy XML in Section 2, we present the encoding scheme in the fuzzy XML documents in Section 3. Section 4 introduces a set of primitive XML algebraic operations, and gives the details of our FTwig algorithm. We describe related work in Section 5 and Section 6 concludes the paper.

II. FUZZINESS IN XML

We have two kinds of fuzziness in XML: the first is the fuzziness in elements and we use membership degrees associated with such elements; the second is the fuzziness in attribute values of elements and we use possibility distribution to represent such values. Note that, for the latter, there exist two types of possibility distribution (i.e., disjunctive and conjunctive possibility distributions) and they may occur in

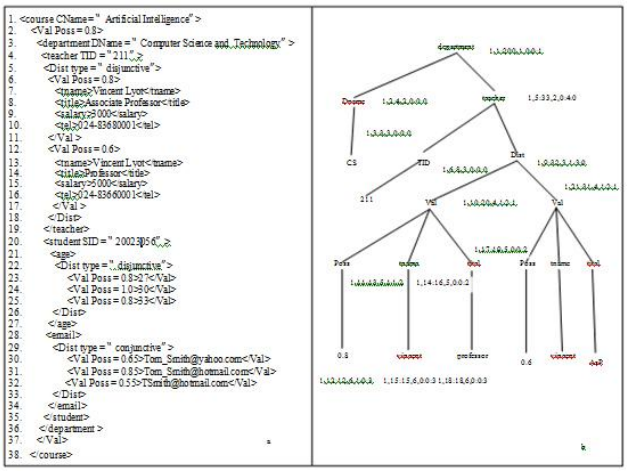

Figure 1. A fragment of an XML document with fuzzy data and a fuzzy encoding scheme tree

child elements with or without further child elements in the ancestor-descendant chain.

We first introduce a possibility attribute (More material can be found in our previous work [12]), denoted Poss, which takes a value between 0 and 1. This possibility attribute is applied together with a fuzzy construct called Val to specify the possibility of a given element existing in the XML document. Consider line 3 of Figure 1(a). <Val Poss = 0.8> states that the possibility of the given element department being Computer Science and Technology is equal to 0.8. For an element with possibility 1.0, pair <Val Poss = 1.0> and </Val> is omitted from the XML document. Based on pair <Val Poss> and </Val>, possibility distribution for an element can be expressed. Also possibility distribution can be used to express fuzzy element values. For this purpose, we introduce another fuzzy construct called Dist to specify a possibility distribution. Typically a Dist element has multiple Val elements as children, each with an associated possibility. Since we have two types of possibility distribution, the Dist construct should indicate the type of a possibility distribution, being disjunctive or conjunctive.

Again consider Figure 1(a). Lines 22-26 are the disjunctive Dist construct for the age of student Tom Smith. Lines 29-33 are the conjunctive Dist construct for the email of student Tom Smith. It should be pointed out that, however, the possibility distributions in line 22-26 and line29-33 are all for leaf nodes in the ancestor-descendant chain. In fact, we can also have possibility distributions and values over non-leaf nodes. Observe the disjunctive Dist construct in lines 5-18, which express the two possible statuses for the employee with ID 211. In these two teacher values, lines 6-11 are with possibility 0.8 and lines 12-17 are with possibility 0.6.

III. TWIG PATTERN QUERY

A. Fuzzy Encoding Scheme

In XML queries, we often have values and structure query. The former means the selection from XML context, we can often get the values of elements or attributes by matching the given information. The latter needs to match the given structure through the path expressions. Structure relations among elements contain: ancestor/descendant relation, parent/child relation and etc. In order to support XML queries effectively, we need encode the elements and attributes in the XML trees.

We use a 4-tuple (DocId, LeftPos: RightPos, LevelNum, Fuzzy: DNum: UNum) to express the nodes' position in the XML database. Where (i) DocId is the identifier of the document. (ii) LeftPos and RightPos can be generated by counting word numbers from the beginning of the document DocId until the start and the end of the element, respectively. Here, we use preorder traversal. When LeftPos is equal to RightPos, the node is a leaf node. (iii) LevelNum is the nesting depth of the element (or string value) in the document. The root node is 1, and the each following level adds 1. (iv) "Fuzzy" expresses whether a node is fuzzy, and its value is a Boolean one. When "Fuzzy" is equal to 1, it means the node is a fuzzy one. "Fuzzy" is equal to 0, it means the node is a crisp one. DNum and Unum indicate the fuzzy level numbers of descendant and ancestor, respectively.

Figure 1(b) gives the nodes encoding scheme. Here, due to space limitation, we just take part sub-tree in Figure 1(a) as an example.

Definition 1. We have the following positions information in XML database, Node1 (D1, L1: R1, LN1, F1: DN1: UN1), Nodei (Di, Li: Ri, LNi, Fi: DNi: UNi), Noden (Dn, Ln: Rn, LNn,Fn: DNn: UNn),1≤i≤n. Structural relationships between tree nodes are:

- ancestor-descentdant: when D1= Di, L1<Li, Ri<R1, node Nodei is a descendant of Node1.
- real-parent-child: In fuzzy XML documents, parent-child structural relationship is different from the crisp one,
  1. When D1=Di, L1<Li, Ri<R1, F1∩ Fi=1, LN1+1=LNi,node Node1 is the real parent of Nodei.
  2. When D1＝Dn, L1<Ln, Rn<R1, F1∩ Fn=1, LN1+i＝LNn, and Fi=0, node Node1 is the real parent of Noden.
- fuzzy-parent-child: When D1=Di, L1<Li, Ri<R1, F1 ∩ Fi=0, LN1+1=LNi, node Node1 is the fuzzy parent of Nodei.

In Figure 1(b), according to definition 1, we have: node "DName" is a crisp node. And node "Val" is a fuzzy one. The structural relationship between "TID" and "211" is real-parent-child, The structural relationship between "teacher" and "tname" is also real-parent-child. The structural relationship between "teacher" and "Dist" is fuzzy-parent-child.

B. Twig Query Matching

XML Query can be expressed using tree structure. The twig pattern node labels include element tags, attribute value comparisons, and string values. The query twig pattern edges are either parent-child edges (depicted using a single line) or ancestor-descendant edges (depicted using a double line). For a twig query matching, we have:

Definition 2. Given a query twig pattern Q and an XML database D, a match of Q in D is identified by a mapping from nodes in Q to nodes in D, such that: (i) query node predicates are satisfied by the corresponding database nodes. (ii) the structural (parent-child and ancestor-descendant) relationships between query nodes are satisfied by the corresponding database

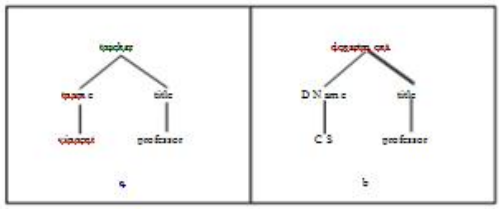

Figure 2. Query twig pattern

Figure 2 gives two twig pattern queries with different structural relationships. A is the parent-child query, and B is a

complex query. B contains both parent-child and ancestor-descendant queries.

IV. FUZZY TWIG PATTERN MATCHING

A. The Operators

All operators in our model take collections of data trees as input, and produce a collection of data trees as output.

Definition 3. For the model definition of the XML data set, we have X = ((E, Eroot), A, Rule, Ref, EAstr), including: (i) E = {e1, e2, …, en} is a finite element set of the root element Eroot, ei(1 ≤ i ≤ n) is the element of the XML data set. (ii) A = {a1, a2, …, am} is the attributes set, aj(1≤j ≤m) is the attribute of the XML data set. (iii) Rule (E) is the set of rules in the XML data set. (iv) Ref (E1/A1, E2/A2) is the reference relations in the XML data set. (v) EAstr is a mapping from E to the power set of A.

For ∀(e,{A1,A2,…,Ak}) ∈ EAstr, Ai(1≤i≤k) is the descendant attribute node of e, expressed as Ai<e. {A1, A2, …, Ak} is the set of descendant attribute nodes, expressed as EAext(e), then we can represent the descendant attribute set of finite set {E1, E2, …, Em} as ∪1≤j≤m EAext(Ej).

Definition 4. Suppose that S =((Es, Eroots), As, Rules, Refs, EAstrs) and T = ((Et, Eroott), At, Rulet, Reft, EAstrt) are two XML dataset models. S and T are isomorphism (abbr.T ≌S), When (i) ET= ES, AT=AS, and ErootT= ErootS (ii) It exists a one-to-one mapping, Frule: RuleS → RuleT , which makes ∀FRule(RuleS) = RuleT, lab(ς (RuleS)) = lab(ς (RuleT)),ς (RuleS)) = ς (RuleT), and for any element E in ς (RuleS), and meets the mapping rule RuleS(E) → RuleT(E) (iii) It exists a one-to-one mapping, FRef: RefS→RefT, which makes ∀FRule (e1S/a1S,e2S/a2S) = (e1T/a1T,e2T/a2T) meet e1S=e1T, a1S=a1T, e2S=e2T, a2S=a2T (iv) It exists a one-to-one mapping, FStr: EAstrS→ EAstrT , which makes ∀FStr(eS,aS)=(eT,aT) meet eS=eT,aS=aT

In fuzzy XML documents, every element appends a "Poss" attribute (0≤Poss≤1), Poss=1 as default. When Poss=1, it means the element value is a crisp one. If XML data are isomorphic, we have the following definition.

Definition 5. Supposed that T.R and T.S are the sub-elements of element T. T.R.X and T.S.Y are the output value of T.R and T.S respectively. μT.R(t) and μT.S(t) represent the possibility of T.R and T.S belonging to T. μ S(t) expresses the complement set of μS(t). Here, 0≤μT.R(t), μT.S(t) ≤1, μT.R(t), μT.S(t) and are equal to 1, as default.

If the XML documents are isomorphic, we have the fuzzy union, intersection, and difference definitions as follows.

⌘ Fuzzy union (∪):

$$T.R.X \cup T.S.Y = \begin{cases} T.R.X, \mu(t) = Max[\mu_T R(t), \mu_T S(t)] & when\ T.R.X = T.S.Y \\ T.(R.X \cup T.Y), \mu(t) = Max[\mu_T R(t), \mu_T S(t)] & when\ T.R = T.S, T.R.X \neq T.S.Y \end{cases}$$

⌘ Fuzzy intersection (∩):

$$T.R.X \cap T.S.Y = \begin{cases} T.R.X, \mu(t) = Min[\mu_T R(t), \mu_T S(t)] & when\ T.R.X = T.S.Y \\ \delta(t), \mu_T S(t)] & when\ T.R = T.S, T.R.X \neq T.S.Y \end{cases}$$

$$T.(R.X \cap T.Y), \mu(t) = Min[\mu_T$$

Fuzzy difference (−)

$$T.R.X - T.S.Y = T.R.X, \mu(t) = \mathrm{Min}[\mu_T R(t), \mu_T \overline{T.S}(t)]$$

$$\mu_T \overline{T.S}(t) = 1 - \mu_T S(t)$$

B. FTwig Algorithm

The existing XML twig query algorithm lacks of the ability of managing fuzzy information. For example, according to fuzzy encoding scheme in Figure 1(b), for the parent-child query "a" in Figure 2, we cannot get the right query answers, because node "teacher" and "tname" don't have the direct parent-child structure relation in the fuzzy XML tree. At the same time, for the ancestor/descendant relation query "b" in Figure 2, if we omit the existence of fuzzy elements, some answers may be deemed questionable because of losing the semantics. We need to modify the fuzzy degree between ancestor node "department" and descendant node "title" according to the fuzzy XML algebraic operations.

In order to solve the problems above, we propose the FTwig matching algorithm, which inspired by PathStack [3], to support fuzzy twig pattern queries.

For a given input twig pattern and stream Tqi, we have the fuzzy twig pattern query algorithm "FTwig" shown in Figure 3. Here, q is the root node. Algorithm FTwig construct stack encodings of partial and total answers to the query path pattern repeatedly, by iterating through the stream nodes in sorted order of their leftPos value. Line 1-10 output a list of root-leaf path solutions as intermediate path solutions. Line 11 merges all potential lists of path solutions which can contribute to produce the final answer to the whole query twig pattern. We may need to block some path solutions during output. More details about the blocking technique can be found in [3].

```
Algorithm Ftwig(q)
1:  While not end (q)
2:  q_min=getMin(q)
//return minimum left-position in the sub-tree node stream.
3:    for each q_i in subtreeNodes(q)
4:    while (not empty(Sq_i) ∧ topR(Sq_i) < leftPos(Tq_min))
5:    pop(Sq_i) // nodes are not in the same branch
6:    push(Sq_min, Tq_min, point to top(Sparent(q_min)))
//a node is pushed on the stack Sq_min, where q_min is the leaf
//node of the query path.
7:  advance(Tq_min)
8:  if (isLeaf(q_min))
9:  call solution(Sq_min, 1)
10: pop(Sq_min)
11: mergeAllSolutionsQ
//merge the trees and output the final answers

Function solution(SN, SP)
//SN is the current stack, SP is the position of node in the stack
1:  index[SN]=SP
2:  if(SN=1) then  // root node
3:  revise(index[n],…, index[1])
4:  else
5:  for i=1 to index[SN].pointer
6:  solution(SN-1, i)
```

Figure 3. FTwig algorithm

The Function solution(SN, SP) manifests the key difference between FTwig and the algorithm PathStack. In algorithm FTwig, we return all potential answers that include the fuzzy information rather than missing partial fuzzy matches which may contribute to the final answers.

When the queries are parent-child structure relation queries, we have

```
Function revise()
//get parent-child structure relation queries
1:  if (q_cnum.level+1=q_child.level)
// parent-child structure relation is crisp
2:  output(index[n],…,index[1])
3:  else
4:  for(k=0;k < q_cnum.dnum;k++)
5:  Q=getNextC
//get the next level nodes of parent node
6:  If(not isFuzzy(q_cnum)&not empty(SQ) ∧  topR(SQ)  < leftPos
(Tq_cnum))
//neither fuzzy nodes nor the child of q_cnum
7:  drop Q
//omit the current answers
8:  else
9:  output(reindex[n],… reindex[1])
//output the fuzzy answers
}
```

Figure 4. Parent-child pattern query algorithm

```
Function revise()
//get ancestor-descendant structure relation queries
1:  for (i=q_acnum.dnum;i>0;i--) {
2:  Q=getInputNext q_acnum
// get the children of node q_acnum
3:  if(is Fuzzy(Q))
4:  Push Q }
5:  for (k=q_dcnum.pnum;k>0;k--) {
6:  T=getInputNext q_dcnum
// get the father of node q_acnum
7:  if(is Fuzzy(T))
8:  Push T
}
9:  FMresult=T∩ Q
//merge the fuzzy nodes
10: Output(reindex[n],…reindex[1])
```

Figure 5. Ancestor-descendant pattern query algorithm

Figure 5 shows the ancestor-descendant pattern query algorithm. Line 1-4 identify whether the continuous descendants of node qancestor are fuzzy. If they are fuzzy, we keep the fuzzy nodes in the stack that can be extended to total answers, given knowledge of the next step to be processed. Line 5-8 preserve the fuzzy continuous ancestors of node qdescendant, which can be also extended to total answers, given knowledge of the next step to be processed. Line 9 is an important step. Here, we read the fuzzy elements in the stack, and merge the partial fuzzy elements by using the fuzzy algebraic operation. Note that this operation manifests the difference between FTwig and the algorithm PathStack. In this scenario, the PathStack returns crisp ancestor-descendant nodes instead of the ancestor-descendant nodes including the merging fuzzy nodes, which may result in losing many potential intermediate answers. Now we take the query "b" in Figure 2 as an example. Based on the encoding scheme, we can see that the ancestor node department's "DNum" and descendant node title's "Unum" are both above 0, we need to use the corresponding fuzzy XML algebraic operator to merge the fuzzy nodes according to the query predicate. Thus the final answer is as followed, {department → (1,5:33,2,0:3:0), DName → (1,2:4,2,0:0:0), CS → (1,3:3,3,0:0:0), Dist → (1,9:32,3,1:3:0), Val → (1,10:20,4,1:2:1), Pos → (1,11:13,5,1:1:2), title → (1,17:19,5,0:0:2), 0. → (1,12:12,6,1:0:3), professor→(1,18:18,6,0:0:3}.

V. RELATED WORK

With the increasing popularity of XML, query processing and optimization for XML databases have attracted a lot of research interest. In particular, twig query matching is identified as a core operation in querying tree-structured XML data. Therefore, there is a rich set of literatures on matching twig queries efficiently. Early work decomposes the tree pattern queries into a set of binary components, then the matches of each individual component are stitched together to get the final results [2]. The main drawback of the decomposition method is the large intermediate results. In [3], Bruno et al. proposed the holistic twig join algorithm to avoid producing a large intermediate result.

In the real world applications, information is often vague or ambiguous. Effects are mainly made in processing the incomplete and probabilistic XML data queries [1, 5, 6, 7, 8, 9, 14]. In [1], Abiteboul et al. introduce a framework for querying and updating probabilistic XML information over two specific models, simple probabilistic tree model and fuzzy tree model. The data model is based on trees where nodes are annotated with conjunctions of probabilistic event variables. A precise complexity analysis of queries and updates for probabilistic trees is introduced in [14]. The work [10] investigates the complete and incomplete twig matching. While this work studies the fuzzy XML twig matching under the distributional nodes with the assist of XML data model and algebraic operators. The works of Kimelfeld et al. [7, 9] appear to be similar to ours but in fact they are quite different. Three semantics twig queries over probabilistic XML has been investigated in their works. They proposed probabilistic twig queries with projections as primitives for matching twig pattern. However, our work focuses on the fuzzy XML twig query processing, and our algorithm FTwig is a generalization of the holistic twig join algorithm to match query twig pattern.

VI. CONCLUSION

The topic of fuzzy XML databases has been intensively studied, for instance, see the more recent works [4, 10, 12, 15]. Current efforts are mainly made in representing fuzzy information in XML databases, less fulfill the requirements of managing fuzzy XML queries, especially in fuzzy twig pattern queries. In order to process the queries, a novel fuzzy labeling scheme to capture the structural information of fuzzy XML databases, as well as some important fuzzy XML algebraic operations are proposed in this paper. On the basis, we present Algorithm “FTwig” for handing fuzzy twig pattern queries.

REFERENCES

[1] S. Abiteboul and P. Senellart, Querying and Updating probabilistic information in XML, In: Proc. EDBT, 2006, pp. 1059-1068.
[2] S. Al-Khalifa, et al. Structural Joins: A Primitive for Efficient XMLQuery Pattern Matching. In: Proc. ICDE, 2002, pp.141-152.
[3] N. Bruno, N.Koudas andD. Srivastava, Holistic Twig Joins: Optimal XML Pattern Matching, In Proc. SIGMOD, 2002, pp.310-321.
[4] A. Gaurav and R. Alhajj, Incorporating Fuzziness in XML and Mapping Fuzzy Relational Data into Fuzzy XML. In: Proc. of the 2006 ACM Symposium on Applied Computing, 2006, pp.456-460.
[5] E. Hung, L. Getoor and V.S. Subrahmanian, PXML: A Probabilistic Semistructured Data Model and Algebra, In: Proc. ICDE, 2003, pp. 467-478.
[6] Y. Kanza, W. Nutt and Y. Sagiv, Querying incomplete information in semistructured data, Journal of Computer and System Sciences 64(3), 2002, pp.655-693.
[7] B. Kimelfeld, Y. Kosharovshy and Y. Sagiv, Query Efficiency in Probabilistic XML Models, In: Proc. SIGMOD, 2008, pp.701-714.
[8] B. Kimelfeld and Y. Sagiv, Combining Incompleteness and Ranking in Tree Queries, In: Proc. ICDT, 2007, pp.329-343.
[9] B. Kimelfeld and Y. Sagiv, Matching Twigs in Probabilistic XML, In: Proc. VLDB, 2007, pp. 27-38.
[10] Liu J. Ma Z.M. and Yan L. Efficient Processing of Twig Pattern Matching in Fuzzy XML. In: Proc. CIKM, 2009, pp.117-126.
[11] J. Lu, T.W. Ling, C. Chan and T. Chen, From Region Encoding to Extended Dewey: On Efficient Processing of XML Twig Pattern Matching, In: Proc. VLDB, 2005, pp.193-204.
[12] Z.M. Ma and L. Yan, Fuzzy XML Data Modeling with the UML and Relational Data Models. Data & Knowledge Engineering. 63, 2007, pp.972-996.
[13] P. Smets, Imperfect Information: Imprecision-Uncertainty, Uncertainty Management in Information Systems: From Needs to Solutions. Kluwer Academic Publishers, 1997, pp.225-254.
[14] P. Senellart and S. Abiteboul, On the Complexity of Managing Probabilistic XML Data, In Proc. PODS, 2007, pp.283-292.
[15] K. Turowski and U. Weng, Representing and Processing Fuzzy Information-an XML-based Approach. Journal of Knowledge Based Systems, Vol 15, 2002, pp.67-75.