CIT. Journal of Computing and Information Technology, Vol. 24, No. 3, September 2016, 293–309
doi: 10.20532/cit.2016.1002861

An Object-Oriented Approach of Keyword Querying over Fuzzy XML

Ting Li

School of Computer Science and Engineering, Northeastern University, Shenyang 110819, China

As the fuzzy data management has become one of the main research topics and directions, the question of how to obtain the useful information by means of keyword query from fuzzy XML documents is becoming a subject of an increasing needed investigation. Considering the keyword query methods on crisp XML documents, smallest lowest common ancestor (SLCA) semantics is one of the most widely accepted semantics. When users propose the keyword query on fuzzy XML documents with the SLCA semantics, the query results are always incomplate, with low precision, and with no possibilities values returned. Most of keyword query semantics on XML documents only consider query results matching all keywords, yet users may also be interested in the query results matching partial keywords. To overcome these limitations, in this paper, we investigate how to obtain more comprehensive and meaningful results of keyword querying on fuzzy XML documents. We propose a semantics of object-oriented keyword querying on fuzzy XML documents. First, we introduce the concept of "object tree", analyze different types of matching result object trees and find the "minimum result object trees" which contain all keywords and "result object trees" which contain partial keywords. Then an object-oriented keyword query algorithm *ROstack* is proposed to obtain the root nodes of these matching result object trees, together with their possibilities. At last, experiments are conducted to verify the effectiveness and efficiency of our proposed algorithm.

ACM CCS (2012) Classification: Information systems → Data management systems → Query languages

Keywords: fuzzy XML, keyword, query, object-oriented, possibility

1. Introduction

Large quantities of fuzzy data appear in various real-world application domains, and how to manage the fuzzy data becomes more and more important. Extensible Markup Language (XML) is rapidly emerging and has been the de facto standard for representing and exchanging data on the Web. Also, how to manage the fuzzy data stored with XML becomes an important research topic. Keyword query is one of the most effective paradigms for information discovery, and it is a user-friendly query method. Users can obtain the corresponding query results only by proposing one keyword or several keywords, without understanding or mastering the complex structure query languages (such as *XQuery*) and the document's schema. Therefore, the study of keyword querying on fuzzy XML documents becomes an important research issue.

Recently, many researchers have devoted their efforts to the representations and query methods of uncertainty data in the forms of XML. For the probabilistic XML data, the data models [1], [2] and query methods [3], [4], [5] on probabilistic XML documents have been studied. And for the fuzzy XML data, the researchers have also proposed some models [6], [7] for the representation of fuzzy information and query methods [8], [9], [10] on fuzzy XML documents. Ma and Yan [7] propose a fuzzy XML data model by introducing the possibility distributions to represent two types of fuzziness. Panić et al. [6] combine indefiniteness in the values of XML and indefiniteness in the structure of XML into a single fuzzy XML extension. Liu et al. [8] propose a holistic twig matching algorithm *LTwig* to evaluate twig queries with AND, OR and NOT connectives in fuzzy XML. While for the research of keyword query methods on uncertainty XML documents, the existing achievements are mainly focused

on the keyword query methods on probabilistic XML documents [4], [5].

Many keyword query semantics and methods have been proposed for the crisp XML documents, and the existing keyword query methods are mainly based on the *Lowest Common Ancestor* (LCAs) semantics and their variants (e.g., SLCA, ELCA and VLCA). Xu and Papakonstantinou [11] propose the *Smallest Lowest Common Ancestor* (SLCA) semantics, and a SLCA of a set keywords is *a lowest node* whose subtree is the *smallest tree* containing all keywords. A smallest answer subtree of a set of keywords is an answer subtree such that none of its subtrees is an answer subtree. Also, two algorithms of Indexed Lookup Eager and Scan Eager are proposed for searching the SLCA semantic results. The *Exclusive Lowest Common Ancestor* (ELCA) semantics is proposed by Guo et al. [12], and an effective algorithm, Indexed Stack, for the keyword queries with ELCA semantics is proposed accordingly in [13]. Li et al. [14] introduce the notion of *Valuable Lowest Common Ancestor* (VLCA) to improve the accuracy and completeness of keyword query. In addition, some researchers focus their attentions on the query problem of returning paths from each LCA (or its variants) node to its descendant nodes as the keyword query results which is named "path return query" [15], [16].

Among these keyword query semantics, SLCA semantics is the most widely accepted one. Let us consider the keyword query over the fuzzy XML document with the SLCA semantics. Figure 1 shows a tree structure of a fuzzy XML document, node a represents the node which directly contains a. Node z_i is an attribute node and x_j is the value of the attribute (e.g., x_4 is the value of attribute z_5). When users propose keyword queries over fuzzy XML document with the traditional SLCA semantics, they face several problems.

(1) For example, when we propose the keyword query $\{x_1, x_2\}$ on this tree, the query result nodes will be the node Dist (conjunctive) and node c under the traditional SLCA semantics. However, the result node Dist is a fuzzy node and should not become the result (the information in the subtree which is rooted at the fuzzy node is incomplete).

(2) For the users, they may not only be interested with the results matching all keywords, but also the results matching partial keywords, such as node h and node g. Node h can be a result node matching keyword x_1, as node h has an attribute z_1 and z_1 has a value x_1. Node g can be a result node matching keyword x_2, as node g has an attribute z_2 and z_2 has a value x_2.

(3) As the fuzzy XML document contains fuzzy information which is represented by the membership degrees associated with elements and the possibility distributions among the values of attributes, a SLCA result should be given a possibility value with the consideration of the fuzzy information (membership degrees) on the path

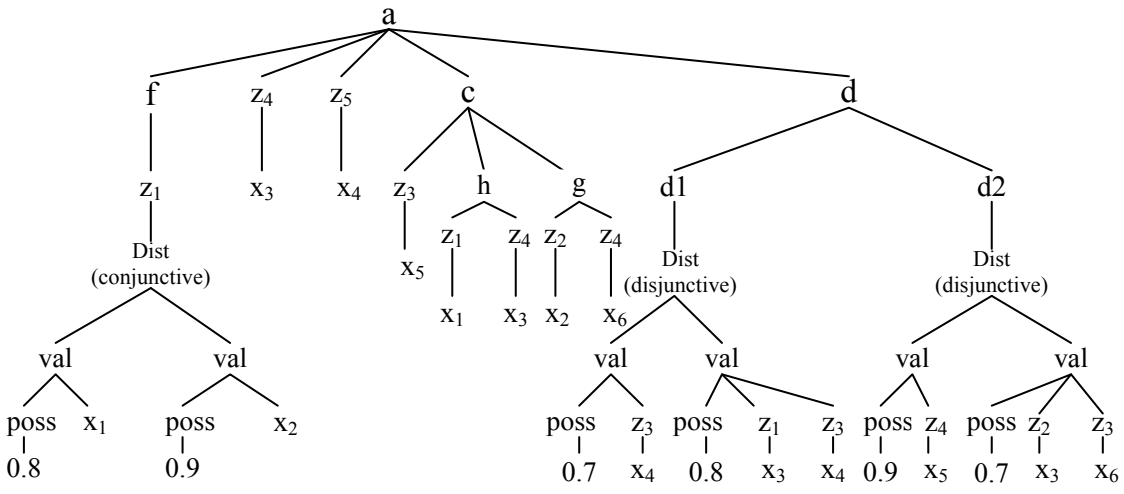


Figure 1. A tree structure of fuzzy XML document.

from the root node of the document to the keyword nodes which contain keywords in the subtree rooted at the SLCA node. Obviously, the traditional SLCA semantics and algorithms cannot compute and obtain the possibilities of result nodes.

It is shown from the descriptions above that it is necessary to obtain the complete and accurate results of keyword querying on fuzzy XML, which are the results with their possibilities matching all keywords and the results with their possibilities matching partial keywords. For this purpose, the object-oriented concept is adopted to capture the *smallest information objects* which contain all keywords in the objects and the *information objects* which contain partial keywords in the objects, and return more meaningful results at the object-level. Based on this idea, we propose the semantics of object-oriented keyword querying on fuzzy XML documents. In this paper, firstly, we introduce the concept of *object tree* into the fuzzy XML document, and a fuzzy XML tree can be divided into many *object trees*, which can be crisp object trees and fuzzy object trees. We analyze the types of matching result object trees which contain all keywords or partial keywords. Then the object-oriented keyword query semantics is proposed. The possibility computation methods for different types of matching result object nodes are given in the following. On these bases, we propose an effective algorithm *ROstack* to obtain the matching result object nodes and their possibilities.

We summarize the contributions of this paper as follows:

- We define the object-oriented keyword query semantics on fuzzy XML documents through introducing the concept of *object tree*. We analyze the types of matching result object trees and give the possibility computation methods for different types of matching result object nodes.
- We propose an algorithm *ROstack* to find the matching result object nodes together with their possibilities. It can also find the matching result object nodes and their possibilities by scanning the relevant keyword nodes only once.
- We conduct experiments to evaluate the performance of our algorithm.

The rest of the paper is organized as follows. We first introduce the preliminary knowledge on fuzzy sets, possibility distributions and the fuzzy XML data model in Section 2. In Section 3, we introduce the concept of "object tree", analyze the relationship between two connected object trees, propose the semantics of object-oriented keyword querying on fuzzy XML and give the methods for the possibility computation of matching result object nodes. The algorithm *ROstack* for generating the matching result object nodes and their possibilities is introduced in Section 4. The experimental results are reported in Section 5. Section 6 concludes the paper and outlooks the future work.

2. Preliminaries

2.1. Fuzzy Sets and Possibility Distributions

In real-world applications, the information is often imperfect (e.g., ambiguous, uncertain and imprecise). In order to reflect this characteristic, researchers have introduced different kinds of imperfect information [17] into the database system. Imprecision, inconsistency and uncertainty are three major kinds of imperfect information. To model the imperfect information in database, many approaches are proposed, and they can be grouped into two large categories: the symbolic and quantitative models [7]. Fuzzy sets [18] have been widely used for the quantification of imprecision and uncertainty.

Let H be universe of discourse and F a fuzzy set in H. A membership function μ_F: $H \rightarrow [0, 1]$ is defined for F, where $\mu_F(\eta_i)$, for each $\eta_i \in H$, denotes the membership degree of η_i in the fuzzy set F. Then, the fuzzy set F is described as follows:

$$F = \{\mu_F(\eta_1) / \eta_1, \mu_F(\eta_2) / \eta_2, \ldots, \mu_F(\eta_n) / \eta_n\} \quad (1)$$

When H is not a discrete set, the fuzzy set F can be represented by:

$$F = \int_{\eta_i \in H} \mu_F(\eta_i) / \eta_i \quad (2)$$

In the above two formulas, $\mu_F(\eta_i)$ is used to rep-

resent the membership degree that η_i belongs to fuzzy set F, and when the $\mu_F(\eta_i)$ is explained to be a measure of the possibility that a variable X has the value η_i, where X takes values in H, then a fuzzy value can be described by a possibility distribution ρ_X.

$$\rho_X = \{\rho_X(\eta_1)/\eta_1, \rho_X(\eta_2)/\eta_2, \ldots, \rho_X(\eta_n)/\eta_n\} \quad (3)$$

Here, $\rho_X(\eta_i)$, $\eta_i \in H$ denotes the possibility that η_i is true. Let ρ_X and F be the possibility distribution representation and the fuzzy set representation for a fuzzy value, respectively.

According to the descriptions above, a fuzzy value on H can be represented by a fuzzy set or a possibility distribution in H. Also, the information fuzziness can be described by means of similarity relations in domain elements, in which the fuzziness comes from the similarity relations between individual values in a universe of discourse [19]. There are three formal types of representations for fuzzy data: the fuzzy set representation, the possibility distribution representation, and the similarity relation representation. The usual data whose values are all crisp values can be regarded as crisp data, and the fuzzy data has the fuzzy value which can be represented by a fuzzy set, a possibility distribution or a similarity relation. The fuzzy set and possibility distribution theories have been used to extend various database models, and also be the basic theories in the fuzzy XML data model which will be introduced in the following.

2.2. Fuzzy XML

In order to represent fuzzy data in XML, two kinds of fuzziness are introduced in [7]: one is the fuzziness in elements, in which membership degrees associated with such elements are used; the other is the fuzziness in attribute values of elements, where possibility distributions are used to represent such values. There are two kinds of interpretation of possibility distributions: disjunctive possibility distributions and conjunctive possibility distributions. In the fuzzy XML tree structure, a possibility attribute is introduced, denoted as "Poss", which takes a value between 0 and 1 and is applied together with a fuzzy construct called "Val" to specify the possibility of a given element. Figure 2 shows a fragment of fuzzy XML document. Considering line 2, <Val Poss = "0.9"> denotes that the possibility of department's name being "Computer Science and Technology" is equal to 0.9. For a crisp element, its membership degree expression: <Val Poss = "1.0"> and </Val> is omitted. In order to express the possibility distributions of values of the attributes, a fuzzy construct "Dist" is introduced into the model. A Dist element has multiple Val elements as children, and each Val element is associated with a possibility for the value of attribute. The Dist element indicates the possibility distribution of values, which is disjunctive possibility distribution or conjunctive possibility distribution. Lines 5-18 in Figure 2 describe a Dist construct which makes the expression of two possible types of information of William James. One expresses (that) the possibility of the information that William James is an associate professor, and the salary is 6000 is equal to 0.8, the other expresses (that) the possibility of the information that William James is a professor, and the salary of 8000 is equal to 0.6. Although the possibility distribution in lines 5-18 is for leaf nodes in the ancestor-descendant chain, we can also have the possibility distributions over non-leaf nodes.

There are two kinds of structures to represent an XML document, the graph structure and tree structure. An XML document with ID/IDREF can be modeled with the graph structure [20], however, many designers may duplicate the information instead of using ID/IDREF links so that an XML document can be simply represented as a tree structure. As an XML document can be represented as an ordered and directed tree structure, a fuzzy XML document can also be represented as a tree structure, and the basic structure of fuzzy XML model is the "data tree". When a fuzzy XML document is represented by an ordered and directed tree T, where $T = (V, E)$, and V are sets of nodes, E are sets of edges. For each $v \in V$, it can be denoted by *label* (v). For two nodes v_i and v_j, $E(v_i, v_j)$ represents a directed edge from node v_i to v_j, and the relationship between v_i and v_j is father-child relationship. There are two kinds of nodes in fuzzy XML: crisp nodes V_C and fuzzy nodes V_F. The former are the ordinary XML nodes, and the

fuzzy nodes (Dist or Val nodes) are the description of the fuzzy information over the subsets of their children. E is the set of edges of fuzzy XML, and it is composed of edges $E_{(C\text{-}C)}$, $E_{(C\text{-}F)}$, $E_{(F\text{-}C)}$, $E_{(F\text{-}F)}$, which represent edges between nodes in V_C and nodes in V_C, edges between nodes in V_C and nodes in V_F, edges between nodes in V_F and nodes in V_C and edges between nodes in V_F and nodes in V_F, respectively.

```
1. <course CName = "Computer Composition Principles">
2.  <Val Poss = "0.9">
3.  <department DName = "Computer Science and Technology">
4.    <teacher TID = "211">
5.     <Dist type = "disjunctive">
6.      <Val Poss = "0.8">
7.        <tname>William James</tname>
8.        <title>Associate Professor</title>
9.        <salary>6000</salary>
10.       < tel>024-83680001</tel>
11.     </Val>
12.     <Val Poss = "0.6">
13.       <tname>William James</tname>
14.       <title>Professor</title>
15.       <salary>8000</salary>
16.       <tel>024-83680001</tel>
17.     </Val>
18.    </Dist>
19.   </teacher>
20.   <student SID = "20123056">
21.    <age>
22.     <Dist type = "disjunctive">
23.       <Val Poss = "0.8">27</Val>
24.       <Val Poss = "1.0">30</Val>
25.       <Val Poss = "0.9">28</Val>
26.     </Dist>
27.    </age>
28.    <email>
29.      <Dist type = "conjunctive">
30.       <Val Poss = "0.65">Tom_Smith@yahoo.com</Val>
31.       <Val Poss = "0.85">Tom_Smith@hotmail.com</Val>
32.       <Val Poss = "0.75">TSmith@hotmail.com</Val>
33.      </Dist>
34.    </email>
35.   </student>
36.  </department >
37. </Val>
38. </course>
```

Figure 2. A fragment of fuzzy XML document.

3. Semantics of Object-Oriented Keyword Querying over Fuzzy XML

3.1. Object-Oriented Concept

Objects are applied to model real-world entities or to abstract concepts [21]. Objects have two characteristics:

(1) an object has attributes and values of the attributes;

(2) an object has a correlation with other objects.

The objects having the same properties are gathered into classes, and theoretically, a class can be considered from two different viewpoints:

(1) an extensional class, where the class is defined by the list of its object instances, and

(2) an intensional class, where the class is defined by a set of attributes and their admissible values.

Based on the object-oriented concept, the element, subelement and attributes in XML data can be naturally mapped into the objects. Considering the fuzzy XML data in Figure 2, the data in lines 4-19 can be mapped into two objects: one is object named teacher, has four attributes and their values, that are tname = "William James", title = "Associate Professor", salary = "6000", tel = "024-83680001". The other is the object also named teacher, has four attributes and their values, that are tname = "William James", title = "Professor", salary = "8000", tel = "024-83680001". The element teacher can be regarded as an object node, the fuzzy nodes are the description of the fuzzy information of the children nodes which are below them and can be neglected in the mapping phase. Then the descendant elements tname, title, salary and tel can be mapped into the attributes. For a group of nodes with no fuzzy information, the elements, subelements and attributes can be mapped into the objects naturally. The object here represents a real entity in the reality and it has a special attribute or a set of attributes for the characteristic. It is noticed that, the object-oriented concept here is different from the object-oriented concept in DOM (Document object model) proposed in [22]. DOM is an object model for document and its specification represents a significant advance-

ment in the handling of semi-structured documents. The DOM represents an XML document using a tree structure, and each node is an object representation of a particular element in the document's content. It describes the structure of the documents as well as its behavior and behavior of its objects.

As a fuzzy XML tree consists of fuzzy nodes and crisp nodes. Based on the object-oriented concept and method, the main nodes in the fuzzy XML documents can be classified into the element node, attribute node, object node, value node (similar to the text node in DOM, which can be the textual content or values of an element), connect node and fuzzy node. An object can be a crisp object or a fuzzy object. An object is regarded as a crisp object if the values of its attributes are crisp values. An object is regarded as a fuzzy object if it has at least one attribute whose value is a fuzzy set. To classify the nodes of document in the fuzzy XML tree, we refer to the nodes identification method in XSeek [23] together with the consideration of the characteristic of fuzzy XML data. The classification of different nodes can be described as follows:

1. A node is an object node if it corresponds to a *-node in the DTD.
2. A node denotes an attribute if it does not correspond to a *-node, and it has only one child which is a value or has children which is a set of possible values.
3. A node is a connect node if it connects nodes within the same category. A connect node can have a child that is an object node, an attribute node or a connect node.
4. A node is a fuzzy node if it is the value node or Dist node.
5. A node is a value node if it contains the textual content or values.
6. A node is an element node if it is not an object node, an attribute node or a value node, but represents the actual content of the document.

3.2. Object Tree

Given a fuzzy XML document D with its tree structure T, T can be regarded as a fuzzy object $O_{T(F)}$. In the tree T, a group of nodes, starting at an object node, followed by some non-object nodes is regarded as an object. As the representation form of XML data is the tree structure, for a subtree $T_s \subseteq T$ with root node $r(T_s)$, if the children nodes of $r(T_s)$ have the attribute nodes, then T_s can be regarded as an object O_s, and $O_s \subseteq O_{T(F)}$. Next, we give the definition of "object tree" as follows:

Definition 1 (object tree). Given an XML tree T_i with its root node $r(T_i)$, if the children nodes of $r(T_i)$ contain at least one attribute node, then T_i is regarded as an "object tree", denoted as O_i. Its root node $r(T_i)$ is called the object node of O_i.

We give some explanation about Definition 1, for an XML tree T_i rooted at $r(T_i)$, if there is an attribute node z which is the child node of $r(T_i)$ (the relationship between the attribute node z and $r(T_i)$ is parent-child relationship), then T_i can be regarded as an object tree O_i with its root node $r(O_i)$ ($r(O_i) = r(T_i)$, and in the following, we use $r(O_i)$ to denote the root node of the object tree O_i). If the values of attributes in O_i are all crisp values, then O_i is a crisp object tree. If the children nodes of $r(O_i)$ have at least one attribute node whose value is a fuzzy set, then O_i is a fuzzy object tree. And when the object tree O_i contains a fuzzy object tree, O_i is also regarded as a fuzzy object tree. For a fuzzy XML document D, if its tree structure T is a fuzzy object tree $O_{T(F)}$, then there may be multiple object trees which are crisp object trees and fuzzy object trees in tree T.

Considering the characteristic of the object together with the characteristics of fuzzy XML data model, there is a special case for the object tree. Seen in Figure 1, nodes $d1$ and $d2$ are also object nodes although they do not have any attribute nodes in their children nodes (the relationship between node $d1$ and the attribute node is not parent-child relationship). But node $d1$ is the root node of two object trees. One is the object tree with the possibility of 0.7, and has attribute z_3 in its tree structure, and the other is the object tree with the possibility of 0.8, and has attributes z_1 and z_3 in its tree structure.

Definition 2 (minimum object tree). Given an object tree O with its root node $r(O)$ in its tree structure, if the attribute nodes only exist in the children nodes of the root node $r(O)$, then the object tree O is regarded as a "minimum object tree", denoted as $O_{\min}$.

We give some explanation about Definition 2: in an object tree O, if all the attribute nodes are the children nodes of the root node $r(O)$ (the relationship between the attribute node and $r(O)$ is only the parent-child relationship), then O is a minimum object tree. For a minimum object tree $O_{\min}$, when it has at least one attribute whose value is a fuzzy set, then $O_{\min}$ can be regarded as a minimum fuzzy object tree. For a set of n minimum object trees, we use $O^1_{\min}$, $O^2_{\min}$, …, $O^n_{\min}$ to denote them. And for a minimum object tree $O_{\min}$, its root node $r(O_{\min})$ is called a minimum object node.

3.3. Relationship Between Object Trees

Figure 3 presents a simplified structure of a fuzzy XML tree structure T, based on the object. We only represent the root nodes of the object trees in T. As shown in Figure 3, the nodes of circle shape represent the root nodes of object trees. If an object is a fuzzy object, we use FO_i to denote it. Especially, we use the node of circle shape named FO_R to represent the root node of the whole fuzzy XML object tree T. If an object is a crisp object, we use O_i to denote it. Nodes of rectangular shape express the types of the possibility distributions between its children object nodes, which can be disjunctive or conjunctive possibility distribution. The value on the edge denotes the membership degree on the path from the parent node to child node, which are the two ends of the edge. The membership degree of edges unlabeled defaults to 1.

Through the analysis of the simplified structure of a fuzzy XML tree in Figure 3, we can identify the relationships between two connected object trees as follows.

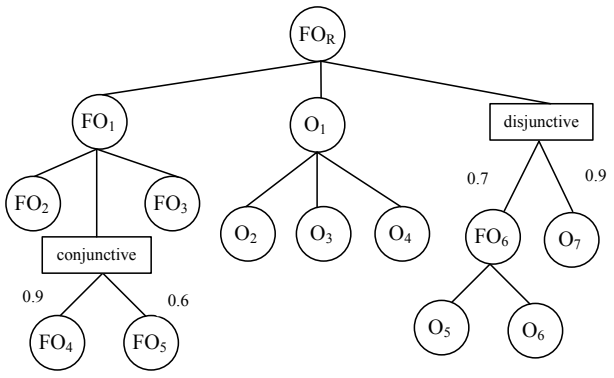


Figure 3. The simplified structure of a fuzzy XML tree based on the object.

(1) An *O-O* relationship: For two connected object trees, the father object tree O_i is a crisp object, and the child object tree O_j is also a crisp object, and there exists a path from nodes $r(O_i)$ to $r(O_j)$ in XML tree, denoted as $p_{r(O_i)\to r(O_j)}$. For example, in Figure 3, the relationship between object trees O_1 and O_4 is the *O-O* relationship.

(2) A *FO-O* relationship: For two connected object trees, the father object tree FO_i is a fuzzy object, and the child object tree O_j is a crisp object, and there exists a path from nodes $r(FO_i)$ to $r(O_j)$ in XML tree, denoted as $p_{r(FO_i)\to r(O_j)}$. For example, the relationship between object trees FO_6 and O_6 is the *FO-O* relationship.

(3) A *FO-FO* relationship: For two connected object trees, the father object tree FO_i is a fuzzy object, and the child object tree FO_j is also a fuzzy object, and there exists a path from nodes $r(FO_i)$ to $r(FO_j)$ in XML tree, denoted as $p_{r(FO_i)\to r(FO_j)}$. For example, the relationship between object trees FO_1 and FO_2 is the *FO-FO* relationship.

It is worth noting that, for the relationships between two connected objects, there are no *O-FO* relationships. Here an *O-FO* relationship means that the father object tree is a crisp object and the child object tree is a fuzzy object. According to the object's characteristics, if an object O contains a fuzzy object, O is also a fuzzy object.

3.4. Matching Result Object Trees and Object-Oriented Keyword Query Semantics

When users propose the keyword queries, they are interested not only in the results matching all keywords, but also in the results matching partial keywords. Based on the object-oriented concept, we should find the "*smallest information objects*" which contain all keywords and "*information objects*" which contain partial keywords. According to the traditional SLCA semantics of keyword queries on crisp XML documents, a SLCA node of m keywords k_1, k_2, …, k_m is a "lowest node" whose subtree is the "smallest" tree containing all keywords. Inspired by this query semantics, given a set

of "minimum object nodes" that the minimum object trees rooted at them contain partial keywords, and a set of nodes whose label directly contain partial keywords, we can find the "lowest common ancestor object nodes" which are the root nodes of the "*smallest object trees*" containing all keywords. Based on the above descriptions, we will show our query semantics and the method of the object-oriented keyword query, starting from the following definition:

Definition 3 (SLCA object tree). For a keyword query $k_1, k_2, \ldots, k_m$, given n minimum object trees $O^1_{\min}, O^2_{\min}, \ldots, O^n_{\min}$ which contain partial keywords in the nodes of their tree structures and a set of nodes $\{U\}$ ($U \notin O_{\min}$) which contain partial keywords, the "SLCA object trees" are:

(1) the "*smallest object tree*" which contains the minimum object trees $\{O^i_{\min}\}$ ($i \in [1, n]$) and nodes $\{U\}$, that nodes in the combination of $\{O^i_{\min}\}$ and $\{U\}$ contain all keywords; and

(2) the "*smallest object tree*" which contains the minimum object trees $\{O^j_{\min}\}$ ($j \in [1, n]$), that nodes in the combination of $\{O^j_{\min}\}$ contain all keywords.

Here, a SLCA object tree is the "*smallest object tree*" containing all keywords, and that means that none of the object trees which are contained in the SLCA object tree contain all keywords. A SLCA object tree is denoted as *SLCAO*. For a SLCA object tree which belongs to type (1), we use $SLCAO^{ov}$ to denote it; and for a SLCA object tree which belongs to type (2), we use $SLCAO^{oo}$ to denote it. The root node of the SLCA object tree is the "smallest lowest common ancestor object node", that is the SLCA object node, denoted as $r(SLCAO)$.

Now, we analyze the different types of matching result object trees when proposing a keyword query on the XML tree based on the object-oriented method. Given an XML tree T and a set of m keywords $\{k_1, k_2, \ldots, k_m\}$, the matching result object trees RO on T can be separated into the following several cases:

(1) The target object tree *TRO*
For a minimum object tree $O_{\min}$, if the nodes of its tree structure contain all keywords, then $O_{\min}$ is regarded as a target object tree *TRO*.

(2) The single target object tree *STRO*
For a minimum object tree $O_{\min}$, if the nodes of its tree structure contain partial keywords of $\{k_1, k_2, \ldots, k_m\}$, then $O_{\min}$ is regarded as a single target object tree *STRO*.

(3) SLCA object tree *SLCAO*
If a matching result object tree $RO \subseteq SLCAO$, then it belongs to one of the following two cases:

A: $SLCAO^{ov}$

A matching result object tree is a $SLCAO^{ov}$ when it contains *STRO* which contain partial keywords in the nodes of their tree structures and nodes U whose labels directly contain partial keywords ($U \notin STRO$). Given a set of single target object trees $\{STRO_{(1)}, STRO_{(2)}, \ldots, STRO_{(m-1)}\}$ and a set of nodes $\{U_{(1)}, U_{(2)}, \ldots, U_{(m-1)}\}$, the set of $SLCAO^{ov}$ can be obtained by the following formula:

$$SLCAO^{ov} = \{SLCAO(STRO_{(1)}, U_{(m-1)}), SLCAO(STRO_{(2)}, U_{(m-2)}), \ldots, SLCAO(STRO_{(m-1)}, U_{(1)})\} \quad (4)$$

In the above formula, set of $SLCAO^{ov}$ is the set of SLCA object trees which contain $STRO_{(q)}$ and nodes $U_{(m-q)}$ in their tree structures. Here, $STRO_{(q)}$ represents two types of minimum object trees, one type $\langle 1 \rangle$ is the minimum object tree which contains q ($1 \le q \le m-1$) keywords in the nodes of its tree structure, the other type $\langle 2 \rangle$ is a set of minimum object trees and the nodes in their combination contain q keywords. And $STRO_{(1)}$ represents the minimum object tree which contains one keyword. $U_{(m-q)}$ also represents two types of nodes, one type $\langle 3 \rangle$ is the node whose label directly contains m-q keywords and the other type $\langle 4 \rangle$ is a set of nodes and their combination contains $m-q$ keywords. $U_{(1)}$ represents the node whose label directly contains one keyword. In one combination of $\{STRO_{(q)}, U_{(m-q)}\}$, $STRO_{(q)}$ can be one type of $\langle 1 \rangle$, $\langle 2 \rangle$, $U_{(m-q)}$ can be one type of $\langle 3 \rangle$, $\langle 4 \rangle$, and nodes in the combination contain all keywords.

B: $SLCAO^{oo}$

A matching result object tree is a $SLCAO^{oo}$ when it only contains the single target object trees *STRO* which contain partial keywords in the nodes of their tree structures. Given a set of single target object trees $\{STRO_1, STRO_2,$

…, $STRO_{m-1}$}, when the nodes in the set of {$STRO_1$, $STRO_2$, …, $STRO_i$} contain all keywords, a $SLCAO^{oo}$ can be obtained by the following formula:

$$SLCAO^{oo} = SLCAO\{STRO_1, STRO_2, \ldots, STRO_i\} \quad (5)$$

Here, $STRO_1$, $STRO_2$, …, $STRO_i$ denote the single target object trees which contain 1, 2, …, i $(1 \le i \le m-1)$ keywords in the nodes of their tree structures. The above formula (5) is an example to obtain $SLCAO^{oo}$. Given a set of *STRO*, assuming that the number of *STRO* which contain keyword k_1 is equal to e_1, the number of *STRO* which contain keyword k_2 is equal to e_2, and similarly, the number of *STRO* which contain keyword k_m is equal to e_m, there are $C_{e_1}^1 \times C_{e_2}^1 \times \ldots \times C_{e_m}^1 = e_1 \times e_2 \times \ldots \times e_m$ possible combinations of {$STRO_i$} for computing the $SLCAO^{oo}$. The set of $SLCAO^{oo}$ is equivalent to the set of SLCA object trees, in which each one is the *smallest object tree* that contains one possible combination of {$STRO_i$}.

In order to simplify the description, in this paper, *RO* can represent a set of matching result object trees or a matching result object tree. $r(RO)$ can represent the root nodes of a set of matching result object trees or the root node of a matching result object tree. And these are also the same for $O_{\min}$, *TRO*, *STRO*, $SLCAO^{ov}$ and $SLCAO^{oo}$. For a matching result object tree *RO*, its root node $r(RO)$ is called the "matching result object node" of *RO*. We use RO_M to denote the matching result object tree which contains all keywords in the nodes of its tree structure. And we use RO_P to denote the matching result object tree which contains partial keywords in the nodes of its tree structure. We know that sets of *TRO*, $SLCAO^{ov}$ and $SLCAO^{oo}$ belong to the set of RO_M, and set of *STRO* is equal to the set of RO_P. We regard RO_M as the minimum result object tree which contains all keywords in its tree structure, and regard RO_P as the result object tree which contains partial keywords in its tree structure.

Given a crisp XML tree T_C, the object-oriented keyword query on T_C returns a set of subtrees which are the minimum result object trees RO_M containing all keywords and result object trees RO_P containing partial keywords. For a set of keywords {k_1, k_2,…, k_m}, the query semantics of an object-oriented keyword query k_1, k_2, …, k_m on T_C is to find:

(1) the root nodes $r(RO_M)$ of the minimum result object trees RO_M which contain all keywords in the nodes of RO_M, and

(2) the root nodes $r(RO_P)$ of the result object trees RO_P which contain partial keywords in the nodes of RO_P.

The set of result nodes $r(RO_M)$ and $r(RO_P)$ is denoted as $\{r(RO_M), r(RO_P)\}$. Next, we show the object-oriented keyword query semantics on the fuzzy XML document.

Definition 4 (Object-oriented keyword query semantics on fuzzy XML). Given a fuzzy XML tree *T* and a set of keywords {k_1, k_2, …, k_m}, the query semantics of an object-oriented keyword query k_1, k_2, …, k_m on *T* is to find a set of pairs of nodes and its possibility $\{(r(RO_M), \lambda), (r(RO_P), \sigma)\}$. In each pair of $(r(RO_M), \lambda)$, $r(RO_M)$ represents the root node of the minimum result object tree RO_M which contains all keywords in the nodes of RO_M, and λ represents the possibility of $r(RO_M)$. And in each pair of $(r(RO_P), \sigma)$, $r(RO_P)$ represents the root node of the result object tree RO_P which contains partial keywords in the nodes of RO_P, and σ represents the possibility of $r(RO_P)$.

3.5. Possibility Computation of Matching Result Object Node

Given a fuzzy XML document *D* with its tree structure *T* and a set of keywords {k_1, k_2, …, k_m}, $r(RO)$ is a matching result object node of the matching result object *RO*, and the whole possibility of $r(RO)$ can be computed by the following formula:

$$P(r(RO)) = P_{\text{path}}(r(RO)) \times P_{\text{local}}(r(RO)) \quad (6)$$

In the above formula, if the membership degrees on the path from the root node of the document to node $r(RO)$ are {φ_1, φ_2, …, φ_n}, $P_{\text{path}}(r(RO)) = \varphi_1 \times \varphi_2 \times \ldots \times \varphi_n$, and $P_{\text{path}}(r(RO))$ is the existence possibility of the matching result object node $r(RO)$. $P_{\text{local}}(r(RO))$ is the local possibility of the matching result object node $r(RO)$, and the computation of $P_{\text{local}}(r(RO))$ can be separated into the following cases:

(a) $RO \in STRO$ (or TRO)

If there is a matching result object tree RO, and the nodes containing keywords in its tree structure are $v_1, v_2, \ldots, v_l$, the membership degrees on the path from node $r(RO)$ to node v_i $(1 \le i \le l)$ are $\{u_1, u_2, \ldots, u_t\}$, then

$$P_{\text{local}}(r(RO)) = u_1 \times u_2 \times \ldots \times u_t \quad (7)$$

(b) $RO \in SLCAO^{\text{ov}}$

If a matching result object tree RO is obtained by the $SLCAO$ $\{STRO_1, STRO_2, v_1, v_2\}$, where $\{STRO_1, STRO_2\}$ are two single target object trees, and $\{v_1, v_2\}$ are nodes containing partial keywords in $SLCAO$, the membership degrees on the path from $r(RO)$ to $r(STRO_1)$ and $r(STRO_2)$ are u_1 and u_2, and the membership degrees on the path from $r(RO)$ to v_1 and v_2 are φ_1 and φ_2, then,

$$\begin{aligned} P_{\text{local}}(r(RO)) = {} & P_{\text{local}}(r(STRO_1)) \\ & \times P_{\text{local}}(r(STRO_2)) \\ & \times \varphi_1 \times \varphi_2 \times u_1 \times u_2 \end{aligned} \quad (8)$$

(c) $RO \in SLCAO^{\text{oo}}$

If a matching result object tree RO contains the single target object trees $\{STRO_1, STRO_2, STRO_3\}$, and the membership degrees on the path from $r(RO)$ to $r(STRO_1)$, $r(STRO_2)$, $r(STRO_3)$ are u_1, u_2 and u_3, respectively, then,

$$\begin{aligned} & P_{\text{local}}(r(RO)) = P_{\text{local}}(r(STRO_1)) \\ & \times P_{\text{local}}(r(STRO_2)) \times P_{\text{local}}(r(STRO_3)) \\ & \times u_1 \times u_2 \times u_3 \end{aligned} \quad (9)$$

4. Algorithm of Object-Oriented Keyword Querying over Fuzzy XML

Based on the object-oriented keyword query semantics on fuzzy XML documents, we propose a keyword query algorithm *ROstack* to obtain the query results together with their possibilities. In the algorithm, we adopt the Dewey code [24] which is widely used in the keyword search algorithms of XML documents to encode the nodes of the fuzzy XML documents. Dewey is an encoding mode which directly puts the Dewey code of a node's parent node as the prefix of the Dewey code of the node. For example, for a node v_g in the tree, its Dewey code is represented as $D(v_g)$. And for the child node v_h of node v_g, the Dewey code of v_h is represented by $D(v_h) = D(v_g) \cdot \rho$, where ρ is the order number of node v_h in all the children nodes of node v_g. The Dewey code has the lexicographical orders. It can effectively support the calculation of inclusion relationship among nodes and support the calculation of position relationship in XML documents. (The Dewey encode mode can be seen in Figure 4).

For the computation of possibilities values, we need to build the index to record the membership degrees on the path from the root node of the document to the keyword nodes (or the object nodes contain keywords in the nodes of the object trees) and index to record the membership degrees on the path from the object node to the keyword nodes (or object nodes) in the object tree. According to the object-oriented query semantics, the nodes processed are classified into two major types in *ROstack*: the object node and non-object node. The distinction between crisp node and fuzzy node is not important in the algorithm, because we return the result node at the object-level, and the fuzzy information (membership degrees values) can be obtained when the root nodes of the fuzzy object trees are returned as the query results, and the possibilities of the result nodes are computed at the same time. But the distinction between object node and non-object node becomes important. So, we also need indexes to record the object nodes and the minimum object nodes.

In the computation phase of the algorithm *ROstack*, to compute the SLCA object nodes, it only needs to process the root nodes of single target object trees and the keyword nodes which are not in the minimum object trees. The *ROstack* is an efficient and effective algorithm to obtain the result nodes and their possibilities, and it performs well when the tree structures of XML documents are complex and deep. Next, we start from introducing the indexes built in our method.

4.1. Index Construction

In order to realize the object-oriented keyword queries on the fuzzy XML document, we build five indexes to serve the algorithm proposed below:

(1) The keywords index: $\{M_i\}$
For a set of keywords $\{k_1, k_2, \ldots, k_m\}$, $\{M_1\}$ denotes the set of nodes containing keyword k_1, and similarly, $\{M_i\}$ denotes the set of nodes containing keyword k_i. If the node $v_h \in O_{\min}$, where v_h contains keywords and $O_{\min}$ is a minimum object tree, we store the $D(r(O_{\min}))$ instead of $D(v_h)$ in the list $\{M_i\}$, $D(r(O_{\min}))$ is the Dewey code of node $r(O_{\min})$, and $r(O_{\min})$ is the root node of $O_{\min}$.

(2) The list of minimum object trees: $L_{O_{\min}}$
$L_{O_{\min}}$ stores the root nodes $r(O_{\min})$ of the minimum object trees $O_{\min}$ and the ordinary nodes in their tree structures. For a minimum fuzzy object tree, we also store only the ordinary nodes in its tree structure.

(3) The list of existence membership degrees of nodes: $L_E\{v, \omega_i, \varepsilon\}(0 < \omega_i \leq 1, 0 < \varepsilon \leq 1)$
L_E records the membership degrees $\{\omega_1, \omega_2, \ldots, \omega_i\}$ on the path from the root node of the document to node v, and the existence possibility value ε, where $\varepsilon = \omega_1 \times \omega_2 \times \ldots \times \omega_i$. It is worth noticing that node v can be an ordinary node containing keywords, or an object node of an object tree which contains keywords in the nodes of its tree structure.

(4) The list of the local membership degrees of object nodes: $L_L\{r(O), \sigma_j, \tau\}(0 < \sigma_j \leq 1, 0 < \tau \leq 1)$;
L_L records the membership degrees $\{\sigma_1, \sigma_2, \ldots, \sigma_j\}$ on the path from the object node $r(O)$ to nodes $\{v_1, v_2, \ldots, v_g\}$ which contain keywords in O, and its local possibility value τ, where node v_g can be an ordinary node or an object node.

(5) The list of object nodes: L_O
After pre-processing the fuzzy XML document with the object identification operation, all the Dewey codes of the object nodes in the fuzzy XML document are recorded into the list L_O.

We give some interpretations of the list $L_{O_{MIN}}$. Let us look at Figure 4, node a represents the node which contains a, the node z_i is the attribute node. In this paragraph, for a simple and intuitive interpretation, we use the form of O_d to represent the crisp object tree rooted at node d (the Dewey code is 1 in B), and FO_a to represent the fuzzy object tree rooted at node a (the Dewey code is 1.7 in A). Figure 4 (A) represents a minimum fuzzy object tree FO_a, and (B) represents a crisp object tree O_d. According to Definition 2, the object tree O_d contains four minimum object trees, which are O_c, O_e, O_h, and O_g, respectively. For the minimum fuzzy object tree FO_a, there are two possible values of attribute z_1, which are x_2 and x_3. And the entry stored in $L_{O_{MIN}}$ is $\{D(a), (z_1, x_2, x_3)\}$. For O_c, O_e, O_h and O_g, the entries stored in $L_{O_{MIN}}$ are $\{D(c), (z_2, x_1)\}$, $\{D(e), (z_3, x_2), (z_6, x_3)\}$, $\{D(h), (z_4, x_4)\}$ and $\{D(g), (z_5, x_5)\}$. $D(a)$ represents the Dewey code of node a (e.g., 1.7 in Figure 4 (A)).

4.2. Algorithm of Object-Oriented Keyword Query

According to the semantics of object-oriented keyword query on the fuzzy XML document,

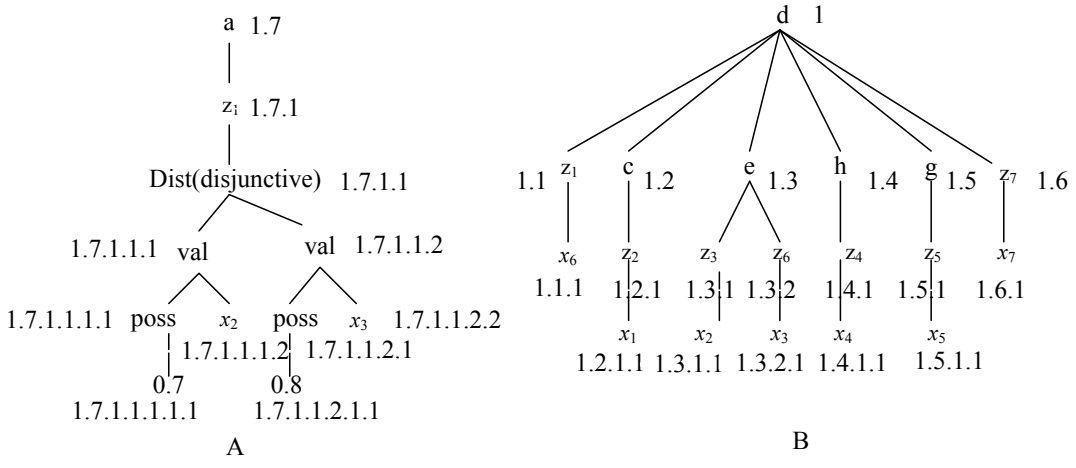


Figure 4. The tree structures of object trees encoded with Dewey.

when users input keywords k_1, k_2, …, k_m, we need to obtain the object nodes $r(RO_{\mathrm{M}})$ of the minimum result object trees RO_{M} and their possibilities λ, and the object nodes $r(RO_{\mathrm{P}})$ of the result object trees RO_{P} and their possibilities σ. In order to realize the query semantics, we propose the following *ROstack* algorithm.

The detailed procedure of *ROstack* algorithm is shown in Algorithm 1. When users input a set of keywords $\{k_1, k_2, \ldots, k_{\mathrm{m}}\}$, the algorithm loads and visits the keyword list$\{M_i\}$, the minimum object tree list $L_{O_{MIN}}$ and the object node list L_O. According to the nodes v_i which contain keywords and minimum object nodes $r(O_{min})$ which contain keywords in the nodes of minimum object trees O_{min}, we create the list $L_E\{v, \omega_i, \varepsilon\}$. From list $L_{O_{MIN}}$ we find object nodes $r(TRO_\alpha)$ of the minimum object trees

Algorithm 1. ROstack.

Input: A set of keywords $\{k_1, k_2, \ldots, k_m\}$, and a fuzzy XML document encoded with Dewey
Output: The matching result object nodes and their possibilities: $\{(r(RO_1), \xi_1), (r(RO_2), \xi_2), \ldots, (r(RO_n), \xi_n)\}$

1: Load and visit keyword index$\{M_i\}(i = 1, 2, ..., m)$, the lists $L_{O_{MIN}}$, L_O, create and update the list $L_E\{v, \omega_i, \varepsilon\}$;
2: Find the set of object nodes $r(TRO_\alpha)$, $r(STRO_\gamma)$, and create the list L_L $\{r(O), \sigma_j, \tau\}$ of nodes $r(TRO_\alpha)$ and $r(STRO_\gamma)$,
3: Compute $P(r(TRO_\alpha))$, $P(r(STRO_\gamma))$;
4: Delete entries of $r(TRO_\alpha)$ from index$\{M_i\}$;
5: Initialize a stack ST = empty,
6: v = get smallest node (),
7: Push node v into the stack ST, and set the Dewey(v) as the initial value of the stack ST;
8: **while** (not reach the end entry of keyword list$\{M_i\}$), **do**{
9: v' = get next smallest node (),
10: $pre = lcp(ST, v')$, //compute the longest common prefix pre between node v and node v' such that $ST[i] = v[i]$, $1 \le i \le pre.length$;
11: **while** ($stack\ ST.size > pre.length$) **do**
12: ST entry $s = ST$.pop();
13: **if** s is SLCA() **then**{
14: when (Dewey (s) $\in$ LO), record s into list $L_E\{v, \omega_i, \varepsilon\}$;
15: when (Dewey (s) $\notin$ LO), find $s' = parent(s)$ (or $ancestor(s)$) and $s' \in L_O$, record s' into list $L_E\{v, \omega_i, \varepsilon\}$;
16: get type s, $s.keyword[j]_1$ (j = 1, 2, …, m),
17: **if** $s \in r(SLCAO^{\mathrm{ov}})$ **then**
18: get $P_{\mathrm{local}}(r(STRO_\gamma))$ from list $L_L\{r(O), \sigma_j, \tau\}$, membership degrees ω_i^s, $\omega_i^{r(STRO)}$, ω_i^v from list $L_E\{v, \omega_i, \varepsilon\}$, and record s into list $L_L\{r(O), \sigma_j, \tau\}$;
19: compute $P(r(SLCAO^{\mathrm{ov}}))$;
20: **if** $s \in r(SLCAO^{\mathrm{oo}})$ **then**
21: get $P_{\mathrm{local}}(r(STRO_\gamma))$ from list $L_L\{r(O), \sigma_j, \tau\}$, membership degrees ω_i^s, $\omega_i^{r(STRO)}$ from list $L_E\{v, \omega_i, \varepsilon\}$, and record s into list $L_L\{r(O), \sigma_j, \tau\}$;
22: compute $P(r(SLCAO^{\mathrm{oo}}))$;}
23: **else** for ($1 \le l \le m$)
24: **if** ($s.keyword[l] = \theta(k_l)$) **then**
25: ST.top.$keyword[l] = \theta(k_l)$;
26: when ($pre.length < d \le v'.length$), ST.push($v'(d)$);
27: when ST is empty, initialize ST with the next smallest node **until** all nodes in list $\{M_i\}$ are processed. }
28: **return** the matching result object nodes and their possibilities: $((r(RO_1), \xi_1), (r(RO_2), \xi_2), \ldots, (r(RO_n), \xi_n))$.

which contain all keywords in the nodes of TRO_α, and object nodes $r(STRO_\gamma)$ of the minimum object trees which contain partial keywords in the nodes of $STRO_\gamma$, and create the list L_L $\{r(TRO_\alpha), \sigma_j, \tau\}$ and L_L $\{r(STRO_\gamma), \sigma_j, \tau\}$. We get $P_{\text{local}}(r(TRO_\alpha))$, $P_{\text{path}}(r(TRO_\alpha))$ from lists L_L $\{r(TRO_\alpha), \sigma_j, \tau\}$ and L_E $\{r(TRO_\alpha), \omega_i, \varepsilon\}$ and compute $P(r(TRO_\alpha)) = P_{\text{path}}(r(TRO_\alpha)) \times P_{\text{local}}(r(TRO_\alpha))$, and similarly, compute the $P(r(STRO_\gamma))$. We delete the entries of $r(TRO_\alpha)$ from the index $\{M_i\}$, as the set of $r(TRO_\alpha)$ are returned as the minimum result object nodes. We compute the SLCA object nodes among the nodes $r(STRO_\gamma)$ and nodes v_i which contain partial keywords. We initialize a stack *ST*, and get node v with the smallest Dewey code in the index $\{M_i\}$, and initiate stack *ST* with the components of the Dewey of node v. We get the next node v' with the smallest Dewey in the index $\{M_i\}$, and compute the longest common prefix *pre* of node v and v'. If the length of the longest common prefix *pre* is smaller than the size of the *ST* (the size of *ST* is equal to the length of the components of the Dewey (v)), the top entries which are not the components of *pre* are popped out. After this, the last component of the *pre* in the stack *ST* becomes the top entry. In the stack *ST*, we use keyword arrays $\{[\kappa_1][\kappa_2]\ldots[\kappa_n]\}$ which are stored behind each entry of Dewey component to denote the subtree rooted at the entries in *ST*, whether containing i^{th} keyword or not. For example, there is an entry (1, [0.7][0.8][0][0.4]) stored at the top entry of *ST*, the first "1" is one component of Dewey, and keyword array [0.7][0.8][0][0.4] denotes nodes of the subtree rooted at the entries of *ST* contains keywords k_1, k_2 and k_4. When processing the keyword node v, we make $keyword[l] = \theta(k_l)$ if v contains keyword k_l and the existence possibility value $P_{\text{path}}(v)$ will be given to $keyword[l]$ at the top entry of the stack. And when processing the keyword node $r(STRO_\gamma)$, if nodes in $STRO_\gamma$ contain keywords k_1 and k_2, then the existence possibility value $P_{\text{path}}(r(STRO_\gamma))$ will be given to $keyword[1]$ and $keyword[2]$ at the same time. If node v (or $r(STRO_\gamma)$) does not contain keyword k_3, then $keyword[3] = 0$. After processing node v, $keyword[l] = \theta(k_l)$ will be transmitted to the top entry of remaining entries in the stack *ST* (see in lines 23-25). After popping out top entries of v, we push the components of Dewey which are not the *pre* of node v' into the stack and get next node in the list $\{M_i\}$ to process. During this process, when finding the component of the popping entry is all denoted by $keyword[j] = \theta(k_j)$ ($j = 1, 2, ..., m$) (which means all $keyword[j] > 0$), then the node s, determined by the entries from the bottom entry to the popping entry in the *stack ST,* is reported as a SLCA node. We pop out the node s, and push the next smallest node as a new initial value of the empty stack, and process the rest keyword nodes in the list $\{M_i\}$.

For a SLCA node s popped out, we should judge whether s is an object node or an ordinary node (see lines 13–15). If Dewey(s) $\in L_O$, s is an SLCA object node and it can be returned as a matching result object node. If Dewey(s) $\notin L_O$, s is a non-object node, we should find its parent node $s' = parent(s)$ along the path in the XML tree. And $parent(s)$ is returned as a matching result object node if Dewey($parent(s)$) $\in L_O$. If Dewey($parent(s)$) $\notin L_O$, we should find the ancestor node $s' = ancestor(s)$ when $ancestor(s) \in L_O$, and return it as the matching result object node. We record $s' = parent(s)$ (or $ancestor(s)$) into the list $L_E\{s', \omega_i, \varepsilon\}$. When popping out node s, we can get the array $keyword[j]_1$ (where $keyword[j]_1 > 0$, $1 \le j \le m$) of the top entry from the stack *ST*, according to the array $keyword[j]_1$ of the top entry of s, we can get the keyword nodes in the subtree T^s_{sub} rooted at node s which contains all keywords. The keyword nodes v_i containing keyword k_2 which have been visited before generating node s with existence possibility value $P_{\text{path}}(v_i)$ in $keyword[2]_1$ will be the nodes in T^s_{sub}.The nodes v_i can be an ordinary node or an object node. There are two types of the SLCA object nodes: $r(SLCAO^{\text{oo}})$ and $r(SLCAO^{\text{ov}})$. $r(SLCAO^{\text{oo}})$ are the root nodes of SLCA object trees of the combinations of single target object trees $STRO_\gamma$. $r(SLCAO^{\text{ov}})$ are the root nodes of SLCA object trees of the combinations of single target object trees $STRO_\gamma$ with nodes v_i which contain partial keywords. For example, if $keyword[j]_1$ is $\{[0.7][0.8][0.6][0.8]\}$, for nodes $v_1 \in M_1$, $r(STRO_2) \in \{M_2, M_4\}$, $v_3 \in M_3$ which have been visited, if v_1, $r(STRO_2)$ and v_3 satisfy the condition: $P_{\text{path}}(v_1) = 0.7$, $P_{\text{path}}(r(STRO_2)) = 0.8$, $P_{\text{path}}(v_3) = 0.6$, $\text{LCA}(v_1, r(STRO_2), v_0) = s$, then nodes v_1, $r(STRO_2)$, v_3 are keyword nodes in T^s_{sub}. After obtaining keyword nodes and visiting the lists $L_L\{r(O), \sigma_j, \tau\}$ and $L_E\{v, \omega_i, \varepsilon\}$, we

can obtain the membership degrees on the paths from node s to keyword nodes, then record s into list $L_L\{s, \sigma_j, \tau\}$, and get $P_{\text{local}}(s)$ by the calculation method shown in Subsection 3.5. For example, for a SLCA object node s and its keyword nodes $STRO_2$, v_1, v_2, if the $P_{\text{local}}(STRO_2) = 0.8$, the membership degrees on the paths from the root node r to node s, $STRO_2$, v_1 and v_2 are {0.8, 0.9, 0.9}, {0.8, 0.9, 0.9, 0.9, 0.8}, {0.8, 0.9, 0.9, 0.7} and {0.8, 0.9, 0.9, 0.9, 0.9} respectively. Then, the membership degrees on the paths from node s to node $STRO_2$, v_1 and v_2 are {0.9, 0.8},{0.7},{0.9, 0.9}, and $P(s) = P_{\text{path}}(s) \times P_{\text{local}}(s) = 0.8 \times 0.9 \times 0.9 \times 0.8 \times 0.8 \times 0.9 \times 0.7 \times 0.9 \times 0.9 = 0.21$. If $keyword[\,j]_1$ is {[0.9][0.9][0.7][0.7][0.7]}, for nodes $r(STRO_2) \in \{M_1, M_2\}$, $r(STRO_3) \in \{M_3, M_4, M_5\}$ which have been visited, if $r(STRO_2)$ and $r(STRO_3)$ satisfy the condition: $P_{\text{path}}(r(STRO_2)) = 0.9$, $P_{\text{path}}(r(STRO_3)) = 0.7$, LCA $(r(STRO_2), r(STRO_3)) = s$, then nodes $r(STRO_2)$ and $r(STRO_3)$ are keyword nodes in T^s_{sub} and the $P_{\text{local}}(s)$ can be calculated with the method shown in Subsection 3.5. Finally, we return all the matching result object nodes and their possibilities $(r(RO_1), \xi_1)$, $(r(RO_2), \xi_2)$, …, $(r(RO_n), \xi_n)$ as query results.

5. Experiments

5.1. Experimental Setting

The algorithm proposed in this paper is implemented with Java on a laptop with 2.13 GHz Intel core i3 with 3 GB memory on Windows 7 system. For testing our algorithm we use a real dataset DBLP [25] and a synthetic dataset XMark [26] . We choose the two data sets since they represent two important characteristics of the data: DBLP is a relatively shallow dataset of a large size; XMark is a balanced dataset with complex structure, varied depth and varied size.

For DBLP, we generate five datasets D_1, D_2, D_3, D_4, D_5 sized of 50 M, 70 M, 90 M, 110 M and 130 M, respectively. For XMark, we also generate five datasets X_1, X_2, X_3, X_4, X_5 sized of 20 M, 40 M, 60 M, 80 M and 100 M, respectively. For each dataset, we use the random fuzzy information generation method used in [8] to transform the crisp XML documents into fuzzy XML documents. The new generated fuzzy XML documents are represented by FD_1, FD_2, FD_3, FD_4, FD_5 and FX_1, FX_2, FX_3, FX_4, and FX_5, respectively. We pre-process the fuzzy XML documents with the object identification operation and identify the object nodes contained in the documents. And the Dewey codes of the object nodes are recorded into the list L_O.

5.2. Evaluation of Query Quality

Precision, recall and F-measure are the evaluation standards for the quality of a query technique of accuracy and completeness which are borrowed from the IR literature. Precision measures accuracy, indicating the fraction of results in the approximate answer that are correct, and recall measures completeness, indicating that the fraction of all correct results actually captured in the approximate answer.

Table 1. Keyword query examples for datasets.

ID	keyword query	ID	keyword query
DQ_1	XML, model, Algebra	XQ_1	Buyer, open_auction
DQ_2	Relational, model, fuzzy, query	XQ_2	person40, phone
DQ_3	Information, retrieval	XQ_3	America, item, address
DQ_4	Fuzzy, XML	XQ_4	buyer, ship, Ed, phone
DQ_5	XML, twig, query	XQ_5	United States, close_auction

For keywords shown in Table 1, we build the structure query statement for each keyword query with the algorithm *LTwig* [8]. *LTwig* is a holistic algorithm which can efficiently evaluate twig queries over the fuzzy XML document, and we get a set of query results R_i and their possibilities λ_i are no less than the given threshold U for each query from the *LTwig* algorithm. We obtain the matching result object nodes $r(RO_i)$ and their possibilities ξ_i from the algorithm *ROstack*. Given a keyword query Q and its corresponding transformed *LTwig* query LQ, the results set of Q (the answers of the keyword query Q), denoted as R_{P}, are the approximate results. And the results set of LQ (the answers of the transformed *LTwig* query), denoted as R_{A}, are the accurate results. Precision and recall of an algorithm are defined as follows. Preci-

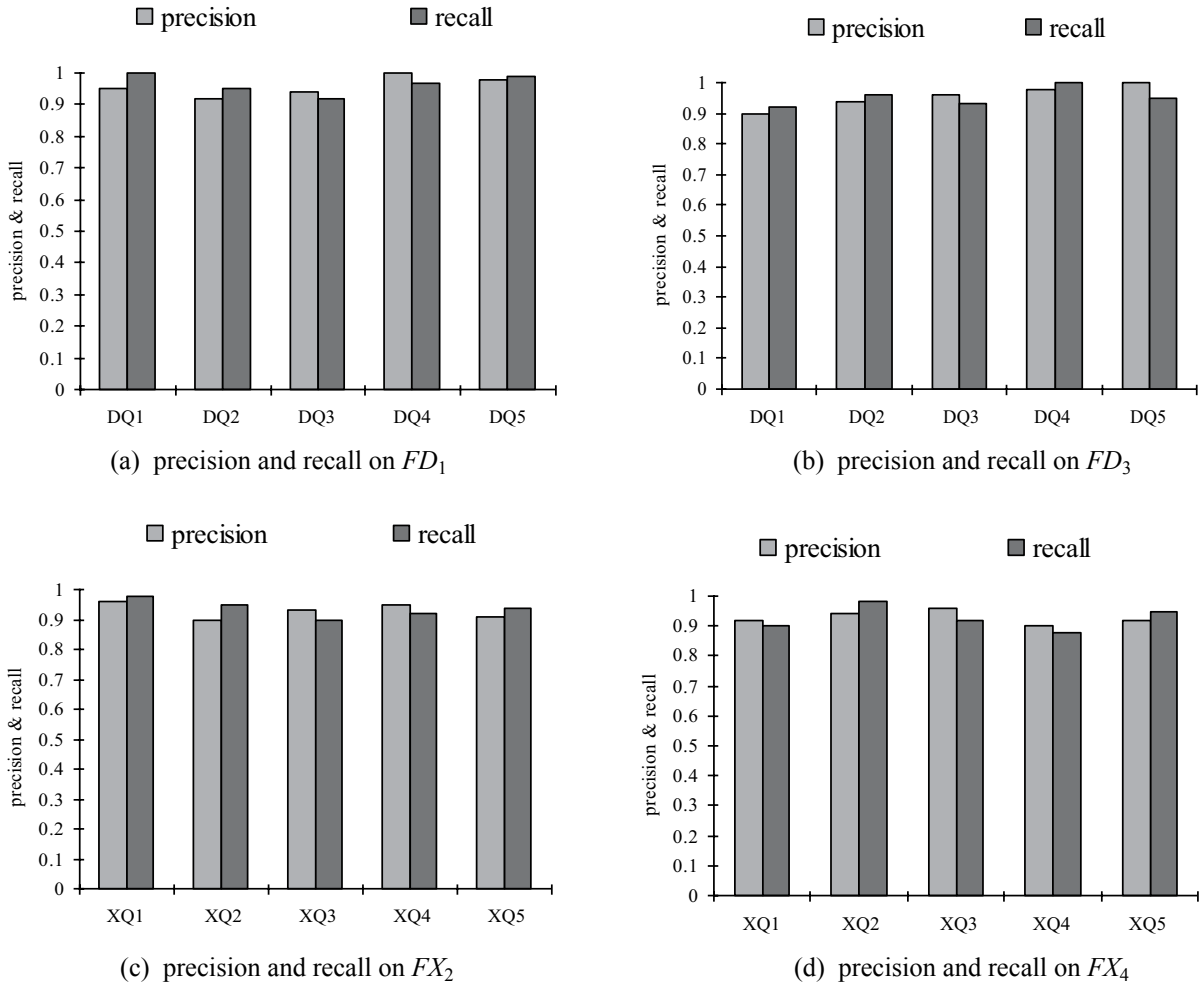
sion of an algorithm is the ratio between $|R_A \cap R_P|$ and $|R_P|$, and recall is the ratio between $|R_A \cap R_P|$ and $|R_A|$. That is, $p_{precision} = |R_A \cap R_P| / |R_P|$, $p_{recall} = |R_A \cap R_P| / |R_A|$. Let f denote the F-measure, then $f = \frac{2 \times p_{precision} \times p_{recall}}{p_{precision} + p_{recall}}$, when $f \neq 0$, $p_{recall} \neq 0$.

We run keyword queries DQ_1–DQ_5 over datasets FD_1 and FD_3, and keyword queries XQ_1–XQ_5 over datasets FX_2 and FX_4. The results of precision and recall of algorithm *ROstack* on different datasets are shown in Figure 5. The experimental results show that for different keyword queries on different fuzzy XML datasets, algorithm *ROstack* has high precision and recall. On the *FD* datasets, the average precision is 0.963, and the average recall is 0.96. On the *FX* datasets, the average precision is 0.927, and the average recall is 0.933. With our approach proposed, we consider not only the AND semantics among keywords, which can return the root nodes of minimum result object trees and their possibilities matching all keywords, but also the OR semantics among keywords, which can return the root nodes of the result object trees and their possibilities matching partial keywords. The answers obtained with the object-oriented query semantics are more meaningful at the object-level and more complete.

Table 2 demonstrates the average F-measure on different datasets. On the fuzzy XML datasets *FD*, the F-measure of *ROstack* reaches 96%, and on the fuzzy XML datasets *FX*, the F-measure of *ROstack* reaches 92%.

Table 2. F-measure.

F-measure	*FD*	*FX*
ROstack	96%	92%



(a) precision and recall on FD_1

(b) precision and recall on FD_3

(c) precision and recall on FX_2

(d) precision and recall on FX_4

Figure 5. The precision and recall of algorithm *ROstack* on datasets FD_1, FD_3, FX_2 and FX_4.

6. Conclusion

In this paper, we propose a semantics of object-oriented keyword querying over fuzzy XML. By introducing the concept *object tree*, we can get the matching result object trees which are the minimum result object trees RO_M containing all keywords in their tree structures, and result object trees RO_P containing partial keywords in their tree structures. The root nodes of RO_M and RO_P, which are $r(RO_M)$ and $r(RO_P)$, together with their possibilities are returned as the computational results. Based on our keyword query semantics, we can not only get the query results matching all keywords at the object level, but also the query results matching partial keywords at the object level.

As the number of results returned is enormous and disordered, it is difficult for users to filter the useful information quickly and effectively from the large number of results. In the future, we will devote our effort to the issue of filtering and ranking the query results as well as to the issue of query optimization to obtain higher quality query results.

Acknowledgments

The work is supported by the National Natural Science Foundation of China (61370075).

References

[1] A. Nierman and H. V. Jagadish, "ProTDB: Probabilistic data in XML", in *Proc. 28th Int. Conf. on Very Large Data Bases*, Hong Kong, 2002, pp. 646–657.

[2] E. Hung *et al*., "PXML: a probabilistic semistructured data model and algebra", in *Proc. 19th Int. Conf. on Data Engineering*, Bangalore, 2003, pp. 467–478.
http://dx.doi.org/10.1109/ICDE.2003.1260814

[3] B. Kimelfeld *et al*., "Query efficiency in probabilistic XML models", in *Proc. 2008 ACM SIGMOD Int. Conf. on Management of Data*, Vancouver, 2008, pp. 701–714.
http://dx.doi.org/10.1145/1376616.1376687

[4] C. Zhang *et al*., "Keywords Filtering over Probabilistic XML Data", in *Proc. 14th Asia-Pacific Web Conf.*, Kunming, 2012, pp. 183–194.
http://dx.doi.org/10.1007/978-3-642-29253-8_16

[5] J. Li *et al*., "Quasi-SLCA Based Keyword Query Processing Over Probabilistic XML Data", *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, pp. 957–969, Apr., 2014.
http://dx.doi.org/10.1109/TKDE.2013.67

[6] G. Panić *et al*., "Fuzzy XML and prioritized fuzzy XQuery with implementation", *Journal of Intelligent & Fuzzy Systems*, vol. 26, no. 1, pp. 303–316, 2014.
http://dx.doi.org/10.3233/IFS-120739

[7] Z. M. Ma and L. Yan, "Fuzzy XML data modeling with the UML and relational data models", *Data & Knowledge Engineering*, vol. 63, no. 3, pp. 972–996, 2007.
http://dx.doi.org/10.1016/j.datak.2007.06.003

[8] J. Liu *et al*., "Efficient processing of twig query with compound predicates in fuzzy XML", *Fuzzy sets and systems*, vol. 229, pp. 33–53, 2013.
http://dx.doi.org/10.1016/j.fss.2012.11.004

[9] J. Liu *et al*., "Dynamically Querying Possibilistic XML Data", *Information Sciences*, vol. 261, pp. 70–88, 2014.
http://dx.doi.org/10.1016/j.ins.2013.11.011

[10] Z. M. Ma *et al*., "Matching twigs in fuzzy XML", *Information Sciences*, vol. 181, no. 1, pp. 184–200, 2011.
http://dx.doi.org/10.1016/j.ins.2010.09.001

[11] Y. Xu and Y. Papakonstantinou, "Efficient Keyword Search for Smallest LCAs in XML Databases", in *Proc. 2005 ACM SIGMOD Int. Conf. on Management of Data*, Baltimore, 2005, pp. 527–538.
http://dx.doi.org/10.1145/1066157.1066217

[12] L. Guo *et al.*, "XRANK: Ranked keyword search over XML documents", in *Proc. 2003 ACM SIGMOD Int. Conf. on Management of Data*, San Diego, 2003. pp. 16–27.
http://dx.doi.org/10.1145/872757.872762

[13] Y. Xu and Y. Papakonstantinou, "Efficient LCA Based Keyword Search in XML data", in *Proc. 11th Int. Conf. on Extending Database Technology: Advances in database technology*, Nantes, 2008, pp. 535–546.
http://dx.doi.org/10.1145/1353343.1353408

[14] G. Li *et al*., "Efficient Keyword Search for Valuable LCAs over XML Documents", in *Proc. 16th ACM Conf. on Conf. on Information and Knowledge Management*, Lisbon, 2007, pp. 31–40.
http://dx.doi.org/10.1145/1321440.1321447

[15] G. Bhalotia *et al*., "Keyword Searching and Browsing in Databases Using BANKS", in *Proc. 18th Int. Conf. on Data Engineering*, San Jose, 2002, pp. 431–440.
http://dx.doi.org/10.1109/ICDE.2002.994756

[16] V. Hristidis *et al*., "Keyword Proximity Search in XML Trees", *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, pp. 525–539, Apr., 2006.
http://dx.doi.org/10.1109/TKDE.2006.1599390

[17] P. Smets, "Imperfect information: imprecision-uncertainty", in Uncertainty Management in Information Systems: from Needs to Solutions, Dordrecht, The Netherlands: *Kluwer Academic Publishers*, pp. 225–254, 1997.

[18] L. A. Zadeh, "Fuzzy sets as a basis for a theory of possibility", *Fuzzy Sets and Systems*, vol. 1, pp. 3–28, 1978.

[19] L. A. Zadeh, "Similarity relations and fuzzy orderings", *Information Sciences*, vol. 3, no. 2, pp. 177–200, 1971.
http://dx.doi.org/10.1016/S0020-0255(71)80005-1

[20] V. Hristidis *et al*., "Keyword proximity search on XML graphs", in *Proc. 19th Int. Conf. on Data Engineering*, Bangalore, 2003, pp. 367–378.
http://dx.doi.org/10.1109/ICDE.2003.1260806

[21] Z. M. Ma *et al*., "Extending object-oriented databases for fuzzy information modeling", *Information Systems*, vol. 29, no. 5, pp. 421–435, 2004.
http://dx.doi.org/10.1016/S0306-4379(03)00038-3

[22] J. Marini, *Document Object Model*, New York: McGraw-Hill, Inc., 2002.

[23] Z. Liu and Y. Chen, "Identifying meaningful return information for XML keyword search", in *Proc. 2007 ACM SIGMOD Int. Conf. on Management of Data*, Beijing, 2007, pp. 329–340.
http://dx.doi.org/10.1145/1247480.1247518

[24] Dewey Decimal Classification [Online]. Available: http://www.oclc.org/dewey

[25] DBLP Bibliography [Online]. Available:
http://dblp.uni-trier.de/xml/

[26] XMARK the XML-benchmark Project [Online]. Available:
http://www.monetdb.cwi.nl/xml/index.html

Received: November 2015
Revised: June 2016
Accepted: July 2016

Contact address:
Ting Li
School of Computer Science and Engineering
Northeastern University
Shenyang 110819, China
e-mail: kitehyabc@163.com

Ting Li is currently a PhD candidate at the School of Computer Science and Engineering, Northeastern University, China. Her research interests include XML data management, keyword query processing and query optimization.