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 incomplete, 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 Panakontaetinou [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 zj is an attribute node and xj is the value of the attribute (e.g., xj is the value of attribute zj). 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 \{x₁, x₂\} on this tree, the query result nodes will be the node Dist (conjunctive) and node e 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 x₁, as node h has an attribute z₁ and x₁ has a value x₁. Node g can be a result node matching keyword x₂, as node g has an attribute z₂ and x₂ has a value x₂.

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 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.

Figure 1. A tree structure of fuzzy XML document.

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 characteristics, 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 \( \mu_F : H \rightarrow [0, 1] \) is defined for \( F \), where \( \mu_F(\eta_i) \), for each \( \eta_i \in H \), denotes the membership degree of \( \eta_i \) in the fuzzy set \( F \). Then, the fuzzy set \( F \) is described as follows:

\[
F = \{\mu_F(\eta_1)/\eta_1, \mu_F(\eta_2)/\eta_2, \ldots, \mu_F(\eta_n)/\eta_n\}
\]

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

\[
F = \int_{\eta \in H} \mu_F(\eta)/\eta
\]

In the above two formulas, \( \mu_F(\eta) \) is used to rep-
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 $z_2$ is an attribute node and $x_i$ is the value of the attribute (e.g., $x_4$ is the value of attribute $z_4$). 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 $e$ under the traditional SLCA semantics. However, the result node Dist is fuzzy 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 $z_2$, and $z_2$ 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 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 results of possibility 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. 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 outlines 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 $\mu_F: H \to [0, 1]$ is defined for $F$, where $\mu_F(\eta_i)$, for each $\eta_i \in H$, denotes the membership degree of $\eta_i$ in the fuzzy set $F$. Then, the fuzzy set $F$ is described as follows:

$$F = \{\mu_F(\eta_1)/\eta_1, \mu_F(\eta_2)/\eta_2, \ldots, \mu_F(\eta_i)/\eta_i\}$$

(1)

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

$$F = \int_{\eta \in H} \mu_F(\eta)/\eta$$

(2)

In the above two formulas, $\mu_F(\eta)$ is used to rep-
resent the membership degree that \( \eta \) belongs to fuzzy set \( F \), and when the \( \mu_F(\eta) \) is explained to be a measure of the possibility that a variable \( X \) has the value \( \eta \), where \( X \) takes values in \( H \), then a fuzzy value can be described by a possibility distribution \( \rho_F \).

\[
\rho_F = \{ \mu_F(\eta_1)/\eta_1, \mu_F(\eta_2)/\eta_2, \ldots, \mu_F(\eta_n)/\eta_n \} \tag{3}
\]

Here, \( \mu_F(\eta_i), \eta_i \in H \) denotes the possibility that \( \eta_i \) is true. Let \( \rho_F \) 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 structure, a possibility attribute value 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, \(<\text{Val Poss} = "0.9"/>\) 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 is \(<\text{Val Poss} = "1.0"/>\) and \(<\text{Val}/>)\) 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 \( V \) is the set of nodes, \( E \) is the set of edges. For each \( v \in V \), it can be denoted by label \( (v) \). For some nodes \( v \) that is a father-child relation, \( v \) represents a directed edge from node \( v_o \) to \( v \), and \( v \) is a father-child relation. There are two kinds of nodes in fuzzy XML: crisp nodes \( V_F \) and fuzzy nodes \( V_C \). 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 \( E_{VF}, E_{VF}, E_{VF}, E_{VF}, E_{VF}, E_{VF} \) which represent edges between nodes in \( V_F \) 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 \( V_C \) and nodes in \( V_F \), respectively.

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

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 parse 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 = "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.
resent the membership degree that \( \eta_i \) 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 \( \eta_i \), where \( X \) takes values in \( H \), then a fuzzy value can be described by a possibility distribution \( \rho_F \).

\[
\rho_F = \{ \rho_X(\eta_1), \rho_X(\eta_2), \ldots, \rho_X(\eta_n) \} \tag{3}
\]

Here, \( \rho_X(\eta_i) \), \( \eta_i \in H \) denotes the possibility that \( \eta_i \) is true. Let \( \rho_F \) 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 is: \(<Val Poss = "1.0">\) and \(<Val Val \) 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. 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 a data tree, it has an ordered and directed tree structure, where \( V \) is a set of nodes, \( E \) is the set of edges, and each node in \( V \) is associated with a possibility distribution. For each node \( v \in V \), its label is denoted by \( \text{label}(v) \). For each node \( v \) and \( \alpha \), \( \text{val}(\alpha, v) \) represents a directed edge from node \( v \) to \( v_\alpha \), and the relationship object between \( v_\alpha \) and \( v \) is father-child relationship. There are two kinds of nodes in fuzzy XML: crisp nodes \( V_N \) 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 \( E_{(F, F)}, E_{(F, N)}, E_{(N, F)}, E_{(N, N)} \), which represent edges between nodes in \( V_F \) and nodes in \( V_N \), edges between nodes in \( V_F \) and nodes in \( V_F \), edges between nodes in \( V_N \) and nodes in \( V_F \) and edges between nodes in \( V_N \) and nodes in \( V_N \), respectively.

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. (1) an extensional class, where the class is defined by the list of its object instances, and
2. (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 pose 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 name = "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 name = "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 name, 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.

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 $r(O_i)$ = $r(T_i)$, and in the following, we use $r(O_i)$ to denote the root node of the object $O_i$. 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 $O$, $O_i$ is also regarded as a fuzzy object tree. For a fuzzy XML document $D$, if its tree structure $T$ is a fuzzy object tree $O_{tree}$, 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 $d_1$ and $d_2$ are also object nodes although they do not have any attribute nodes in their children nodes (the relationship between node $d_1$ and the attribute node is not parent-child relationship). But node $d_1$ is the root node of two object trees. One is the object tree with the possibility of 0.7, and has attribute $z_2$ 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 $O_{min}^1, O_{min}^2, \ldots, O_{min}^n$ 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 trees in $T$. 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 represent it. Especially, we use the node of circle shape named $FO_0$ 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 which are the two ends of the edge. The membership degree on the edge denotes the membership degree on the object trees, and for a minimum fuzzy object tree, the father object tree and the child object tree is a fuzzy object. If an object is a crisp object, and the child object tree is a fuzzy object, and there exists a path from nodes $r(FO)$ to $r(O)$ in XML tree, denoted as $P_{(FO,O)}$. For example, the relationship between object trees $O_{2}$ and $O_{3}$ is the O-O relationship.

**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 object" 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 $k_1, k_2, \ldots, k_m$ is a "lowest node" whose subtree is the "smallest" tree containing all keywords. Inspired by this query semantics, given a set
Given a fuzzy XML document as a sample, we can regard it as a fuzzy object \( O_{N_D} \). 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 \( T_s \subset T \) 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 \( O_{s} \subset O_{N_D} \). Next, we give the definition of "object tree" as follows:

**Definition 1 (object tree).** Given an XML tree \( T \) with its root node \( r(T) \), if the children nodes of \( r(T) \) contain at least one attribute node, then \( T \) is regarded as an "object tree", denoted as \( O \). If the root node \( r(T) \) is called the object node of \( O \).

We give some explanation about Definition 1, for an XML tree \( T \) rooted at \( r(T) \), if there is an attribute node \( z \) which is the child node of \( r(T) \) (the relationship between the attribute node \( z \) and \( r(T) \) is parent-child relationship), then \( T \) can be regarded as an object tree \( O \) with its root node \( r(O) \). If the values of attributes in \( O \) are all crisp values, then \( O \) is a crisp object tree. If the children nodes of \( r(O) \) have at least one attribute node whose value is a fuzzy set, then \( O \) is a fuzzy object tree. And when the object tree \( O \) contains a fuzzy object tree, \( O \) is also regarded as a fuzzy object tree. For a fuzzy XML document \( D \), if its tree structure \( T \) is a fuzzy object tree \( O_{N_D} \), 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 \( d_1 \) and \( d_2 \) are also object nodes although they do not have any attribute nodes. We also regard \( d_1 \) and \( d_2 \) as object nodes and \( O_{N_D} \) as object tree.

**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 \( O_{\text{min}} \).

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 \( O_{\text{min}} \), when it has at least one attribute whose value is a fuzzy set, then \( O_{\text{min}} \) can be regarded as a minimum fuzzy object tree. For a set of \( n \) minimum object trees, we use \( O_{\text{min}}^1, O_{\text{min}}^2, \ldots, O_{\text{min}}^n \) to denote them. And for a minimum object tree \( O_{\text{min}} \), its root node \( r(O_{\text{min}}) \) 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 trees in \( T \). For each of the root nodes of the object trees in \( T \), we use \( O_{r(T)} \) to denote the root node of the object tree. If an object is a crisp object, we use \( F_{O_{r(T)}} \) to denote it. Especially, we use the node of circle shape named \( F_{O_{r(T)}} \) to represent the root node of the whole fuzzy XML object tree. If an object is a crisp object, we use \( O_{r(T)} \) to denote it. Nodes of rectangular shape express the types of the possibility distributions between its children 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.

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 \( R_{r(O_i)-r(O_j)} \). For example, in Figure 3, the relationship between object trees \( O_i \) and \( O_j \) is the O-O relationship.

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

3. **A FO-FO relationship:** For two connected object trees, the father object tree \( F_{O_i} \) is a fuzzy object, and the child object tree \( F_{O_j} \) is also a fuzzy object, and there exists a path from nodes \( r(F_{O_i}) \) to \( r(F_{O_j}) \) in XML tree, denoted as \( R_{r(F_{O_i})-r(F_{O_j})} \). For example, the relationship between object trees \( F_{O_i} \) and \( F_{O_j} \) 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 define 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 \( k_1, k_2, \ldots, k_m \) 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 \( k_1, k_2, ..., k_m \), given \( n \) minimum object trees \( O_{min}^1, O_{min}^2, ..., O_{min}^n \) which contain partial keywords in the nodes of their tree structures and a set of nodes \( \{U_i\} \cup O_{min} \) which contain partial keywords, the "SLCA object trees" are:

1. The "smallest object tree" which contains the minimum object trees \( \{O_{min}^i\} \) \( (i \in \{1, n\}) \) and nodes \( \{U_i\} \) that nodes in the combination of \( \{O_{min}^i\} \) and \( \{U\} \) contain all keywords; and
2. The "smallest object tree" which contains the minimum object trees \( \{O_{min}^i\} \) \( (j \in \{1, n\}) \), that nodes in the combination of \( \{O_{min}^j\} \) 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 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 \( \{k_1, k_2, ..., k_m\} \), 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 \( \{k_1, k_2, ..., k_m\} \), 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:** \( SLCAO^m \)

A matching result object tree is a \( SLCAO^m \) when it contains \( STRO \) which contain partial keywords in the nodes of their tree structures and nodes \( U \) whose labels directly contain partial keywords (\( U \not\in STRO \)).

**B:** \( SLCAO^o \)

And the method of the object-oriented keyword descriptions, we will show our query semantics containing all keywords. Based on the above smallest object trees the root nodes of the "lowest common ancestor object node" of \( r(SLCAO) \) can be obtained by the following formula:

\[
SLCAO^o = SLCAO(STRO_1, STRO_2, ..., STRO_i) \cup \text{nodes} (U_{m-1}) \cup \text{nodes} (U_{m})
\]

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

**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:

\[
P(RO) = \sum_{i=1}^{m} P_{match}(\phi_i) \times P_{local}(\phi_i)
\]

In the above formula, if the membership degrees on the path from the root node of the document to node \( \phi_i \) are \( \{\phi_1, \phi_2, ..., \phi_m\} \), \( P_{match}(\phi_i) \times P_{local}(\phi_i) \) is the existence possibility of the matching result object node \( r(RO) \). \( P_{match}(\phi_i) \) is the local possibility of the matching result node \( r(RO) \), and the computation of \( P_{match}(\phi_i) \) can be separated into the following cases:
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 \(O_{m1}, O_{m2}, \ldots, O_{mn}\) which contain partial keywords in the nodes of their tree structures and a set of nodes \(\{U\} \cup O_{mn}\) which contain partial keywords, the "SLCA object trees" are:

1. the "smallest object tree" which contains the minimum object nodes \(\{O_{mi}\} \subseteq \{1, n\}\) and nodes \(\{U\}\), that nodes in the combination of \(\{O_{mi}\} \cup \{U\}\) contain all keywords;
2. the "smallest object tree" which contains the minimum object nodes \(\{O_{mi}\} \subseteq \{1, n\}\), that nodes in the combination of \(\{O_{mi}\}\) 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 \(SLCAO^m\) to denote it; and for a SLCA object tree which belongs to type (2), we use \(SLCAO^n\) to denote the root node of the SLCA object tree. A SLCA object tree is the "smallest lowest common ancestor object node", that is the SLCA object node, denoted as \(\rho(SLCAO)\).

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 \(O_{mn}\), if the nodes of its tree structure contain all keywords, then \(O_{mn}\) is regarded as a target object tree \(TRO\).
2. The single target object tree \(STRO\) For a minimum object tree \(O_{mn}\), if the nodes of its tree structure contain partial keywords of \(\{k_1, k_2, \ldots, k_m\}\), then \(O_{mn}\) is regarded as a single target object tree \(STRO\). (2) 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 \(\sigma\) represents the possibility of \(r(RO_m)\). And in each pair of \((r(RO_m), \sigma)\), \(r(RO_m)\) represents the root node of the result object tree \(RO_m\) which contains partial keywords in the nodes of \(RO_m\), and \(\sigma\) represents the possibility of \(r(RO_m)\).

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 \(\{k_1, k_2, \ldots, k_m\}\), \(r(RO)\) 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:

\[ P(r(RO)) = P_{\text{path}}(r(RO)) \times P_{\text{local}}(r(RO)) \]  

In the above formula, if the membership degrees on the path from the root node of the document to node \(r(RO)\) are \(\{\phi_1, \phi_2, \ldots, \phi_p\}\), then \(P_{\text{path}}(r(RO)) = P_{\text{path}}(\phi_1 \cap \phi_2 \cap \ldots \cap \phi_p)\) 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:
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 node 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 \( v \), in the tree, its Dewey code is represented as \( D(v) \). And for the child node \( v_j \) of node \( v_i \), the Dewey code of \( v_j \) is represented by \( D(v_j) = D(v_i) \cdot \rho \), where \( \rho \) is the order number of node \( v_j \) in all the children nodes of node \( v_i \). The Dewey code has the lexicographical orders. It can effectively support the calculation of inclusion association 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 object 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 mode, 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: \( \{ M \} \)
   For a set of keywords \( \{ k_1, k_2, \ldots, k_n \} \), \( \{ M \} \) denotes the set of nodes containing keyword \( k_1 \), and similarly, \( \{ M \} \) denotes the set of nodes containing keyword \( k_2 \). If the node \( v \in \{ M \} \), \( v \) contains keywords and \( O_{v} \) is a minimum object tree, we store the \( D(O_{v}) \) instead of \( D(v) \) in the list \( \{ M \} \), \( D(O_{v}) \) is the Dewey code of node \( O_{v} \), and \( r(O_{v}) \) is the root node of \( O_{v} \).

2. The list of minimum object trees: \( L_{\min} \)
   \( L_{\min} \) stores the root nodes \( O_{\min} \) of the minimum object trees \( O_{\min} \) and the ordinary nodes in their tree structures.

3. The list of existence membership degrees of nodes: \( \{ E \} \)
   for different values of attribute \( \tau \), \( \{ E \} \) denotes the set of nodes on the path from the root node to the node \( v \), and the existence possibility value \( e \), where \( 0 < \tau \leq 1 \) is the membership degree on the path from \( r(O_{v}) \) to \( v \), and \( O_{v} \) represents the fuzzy object tree rooted at node \( v \). The node \( v \) can be an ordinary node containing keywords, or an object node which contains keywords in the nodes of its tree structure.

4. The list of the local membership degrees of object nodes: \( L_{L} \)
   \( L_{L} \) records the membership degrees \( \{ \sigma \} \) on the path from the object node \( r(O) \) to the nodes \( \{ v_1, v_2, \ldots, v_i \} \) which contain keywords in \( O \), and its local possibility value \( \tau \), where node \( v_i \) can be an ordinary node or an object node.

5. The list of object nodes: \( L_{O} \)
   Pre-processing the fuzzy XML document with the object identification operation, all the object nodes of the documents in the fuzzy XML document are recorded into the list \( L_{O} \).

We give some interpretations of the list \( L_{O} \).

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 \( O_a \) to represent the crisp object tree rooted at node \( a \) (the Dewey code is 1 in B), and \( F_O \) 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 \( F_O \), and (B) represents a crisp object tree \( O_a \) according to Definition 2, the object node \( O_a \) contains four minimum object trees, which are \( O_{a1}, O_{a2}, O_{a3} \) and \( O_{a4} \) respectively. For the minimum fuzzy object tree \( F_O \), there are two possible values of attribute \( z_i \), which are \( z_1 \) and \( z_2 \). And the entry stored in \( L_{O_a} \) is \( \{ D(a), (z_1, z_2, z_3) \} \). For \( O_{a1}, O_{a2}, O_{a3} \) and \( O_{a4} \), the entries stored in \( L_{O_a} \) are \( \{ D(c), (z_1, z_2, z_3) \} \) \( \{ D(b), (z_2, z_3) \} \) and \( \{ D(g), (z_2, z_3) \} \). \( D(a) \) 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,}
(a) \( RO \in \text{STRO} \) (or \( \text{TRO} \))

If there is a matching result object tree \( RO \) and the nodes containing keywords in its tree structure are \( v_1, v_2, \ldots, v_x \) to the membership degrees on the path from node \( r(RO) \) to node \( v_j \) \((1 \leq j \leq x)\) are \( \{u_1, u_2, \ldots, u_x\} \), then

\[
P_{\text{local}}(r(RO)) = u_1 \times u_2 \times \ldots \times u_x \tag{7}
\]

(b) \( RO \in \text{SLCAO} \)

If a matching result object tree \( RO \) is obtained by the \( \text{SLCAO} \) \( \{\text{STRO}_1, \text{STRO}_2, \ldots, \text{STRO}_y\} \) two single target object trees, and \( \{v_1, v_2\} \) nodes containing partial keywords in \( \text{SLCAO} \), the membership degrees on the path from \( r(RO) \) to \( v_1 \) and \( v_2 \) and the membership degrees on the path from \( r(RO) \) to \( v_1 \) and \( v_2 \) respectively then,

\[
P_{\text{local}}(r(RO)) = P_{\text{local}}(\text{STRO}_1) \times P_{\text{local}}(\text{STRO}_2) \times \ldots \times P_{\text{local}}(\text{STRO}_y) \tag{8}
\]

(c) \( RO \in \text{SLCAO} \)

If a matching result object tree \( RO \) contains the single target object trees \( \{\text{STRO}_1, \text{STRO}_2, \ldots, \text{STRO}_y\} \), and the membership degrees on the path from \( r(RO) \) to \( r(\text{STRO}_1) \), \( r(\text{STRO}_2), \ldots, r(\text{STRO}_y) \) \( \{u_1, u_2, \ldots, u_x\} \) respectively, then,

\[
P_{\text{local}}(r(RO)) = P_{\text{local}}(\text{STRO}_1) \times P_{\text{local}}(\text{STRO}_2) \times \ldots \times P_{\text{local}}(\text{STRO}_y) \tag{9}
\]

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 \( \text{ROstack} \) 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 node 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 \( v_1 \) in the tree, its Dewey code is represented as \( D(v_1) \). And for the child node \( v_2 \) of node \( v_1 \), the Dewey code of \( v_2 \) is represented by \( D(v_2) = D(v_1) \cdot 1 \), \( \rho \) is the order number of node \( v_1 \) in all the children nodes of node \( v_1 \). The Dewey code has the lexicographical orderly. It can effectively support the calculation of inclusion association among nodes and support the calculation of position relationship in XML documents. (The Dewey encode node 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 tree) 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 \( \text{ROstack} \): 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 \( \text{ROstack} \), 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 \( \text{ROstack} \) 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 \( \{k_1, k_2, \ldots, k_x\} \), \( \{M_i\} \) denotes the set of nodes containing keyword \( k_i \), and similarly, \( \{M_i\} \) denotes the set of nodes containing keyword \( k_i \). If the node \( v_k \in \{M_i\} \) contains keywords and \( O_k \) is a minimum object tree, we store the \( D(O_k) \) instead of \( D(v_k) \) in the list \( \{M_i\} \), \( D(O_k) \) is the Dewey code of node \( O_k \), and \( r(O_k) \) is the root node of \( O_k \).

2. The list of minimum object trees: \( L_{\text{min}} \)
   \( L_{\text{min}} \) stores the node roots \( r(O_k) \) of the minimum object trees \( O_k \) 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: \( \{L_z\} \)
   \( L_z \) records the membership degrees \( \{\sigma_1, \sigma_2, \ldots, \sigma_x\} \) on the path from the root node of the document to the node \( v_z \), and the existence possibility value \( \varepsilon \), where \( \varepsilon = \sigma_1 \times \sigma_2 \times \ldots \times \sigma_x \). It is worth noting that node \( v_z \) 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: \( L_L \)
   \( L_L \) records the membership degrees \( \{\sigma_1, \sigma_2, \ldots, \sigma_j\} \) on the path from the object node \( r(O) \) to nodes \( \{v_1, v_2, \ldots, v_x\} \) which contain keywords in \( O \), and its local possibility value \( \tau \), where node \( v_1 \) can be an ordinary node or an object node.

5. The list of object nodes: \( L_{\text{obj}} \)
   After preprocessing the fuzzy XML document with the object identification operation, all the object nodes of the nodes in the fuzzy XML document are recorded into the list \( L_{\text{obj}} \). We give some interpretations of the list \( L_{\text{obj}} \). 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 \( O \) to represent the crisp object tree rooted at node \( d \) (the Dewey code is 1 in B), and \( F_o \) 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 \( F_o \), and (B) represents a crisp object tree \( O_d \). According to Definition 2, the object tree \( O_z \) contains four minimum object trees, which are \( O_1, O_2, O_3 \), and \( O_4 \) respectively. For the minimum fuzzy object tree \( F_o \), there are two possible values of attributes \( z_i \), which are \( x_5 \) and \( x_6 \). And the entry stored in \( L_{\text{obj}} \) is \( \{D(a), (z_1, z_2, x_3)\} \). For \( O_2, O_3, O_4 \), the entries stored in \( L_{\text{obj}} \) are \( \{D(c), (z_2, x_4)\} \) and \( \{D(g), (z_3, x_5)\} \). \( D(a) \) represents the Dewey code of node \( a \) (e., 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,
when users input keywords $k_1, k_2, \ldots, k_n$, we need to obtain the object nodes $r(RO_i)$ of the minimum result object trees $RO_i$ and their possibilities $\lambda_i$ and the object nodes $r(RO_i)$ of the result object trees $RO_i$ and their possibilities $\sigma_i$.

In order to realize the query semantics, we propose the following $RO$-stack algorithm:

The detailed procedure of $RO$-stack algorithm is shown in Algorithm 1. When users input a set of keywords $\{k_1, k_2, \ldots, k_n\}$, the algorithm loads and visits the keyword list $\{M_i\}$, the minimum object tree list $L_{max}$, and the object node list $L_{RO}$. According to the nodes $v$ which contain keywords and minimum object nodes $r(O_{max})$ which contain keywords in the nodes of minimum object trees $O_{max}$ we create the list $L_{v}(\{v_o, o_e\})$. From list $L_{v_{max}}$ we find object nodes $r(RO_j)$ of the minimum object trees which contain all keywords in the nodes of $TRO_j$ and object nodes $r(STR_j)$ of the minimum object trees which contain partial keywords in the nodes of $STR_j$.

Algorithm 1. $RO$-stack.

**Input:** A set of keywords $\{k_1, k_2, \ldots, k_n\}$ and a fuzzy XML document encoded with Dewey.

**Output:** The matching result object nodes and their possibilities: $\{(r(RO_1), \lambda_1), (r(RO_2), \lambda_2), \ldots, (r(RO_n), \lambda_n)\}$

1: Load and visit keyword index $(M_i)|i=1,2,\ldots,m|$ lists $L_{max}$. Create and update the list $L_{v}(\{v_o, o_e\})$.
2: Find the set of object nodes $r(RO_j)$, $r(STR_j)$, and create the list $L_{v}(\{v_o, o_e\})$ of nodes $r(RO_j)$ and $r(STR_j)$.
3: Compute $P(\lambda(RO_j))$, $P(\lambda(STR_j))$.
4: Write entries of $r(RO_j)$ from index $M_i$.
5: Initialize a stack $ST$ empty.

6: $v = \text{get smallest node}()$.
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 of keyword index $M_i$) do
9: \hspace{1em} $v = \text{get next smallest node}()$.
10: \hspace{2em} $pre = \text{length}(ST, v)$; \hspace{1em} /\* compute the longest common prefix between node $v$ and node $v'$ such that $ST \{v'\} = \{v\}$. 1 \leq \text{pre:length} \leq \text{length}(ST) \*/
11: \hspace{2em} while (length(ST, $v$) $<$ pre-length) do
12: \hspace{3em} $ST$ entry $= ST\text{pop}()$.
13: \hspace{3em} if $v$ is SLCA then \{\begin{align*}
14: & \text{when (Dewey $(s) \notin L_{RO}$), record $s$ into list $L_{v}(\{v_o, o_e\})$;}
15: & \text{when (Dewey $(s) \in L_{RO}$), find $s' = \text{parent}(s)$ (or ancestor$(s)$) and $s' \in L_{RO}$, record $s'$ into list $L_{v}(\{v_o, o_e\})$;}
16: & \text{get $s$ from $s$. key.\{j\}$ (j = 1, 2, \ldots, m) \};}
17: & \text{if $s \in r(\text{SLCA}(\lambda))$ then}
18: & \hspace{1em} get $P(\lambda(\text{SLCA}(\lambda)))$ from list $L_{v}(\{v_o, o_e\})$, membership degrees $\alpha_{v_{max}}^{\lambda}$, $\beta_{v_{max}}^{\lambda}$ from list $L_{v}(\{v_o, o_e\})$, and record $s$ into list $L_{v}(\{v_o, o_e\})$;}
19: & \hspace{1em} compute $P(\lambda(\text{SLCA}(\lambda)))$;}
20: & \text{else for (1 $\leq \text{pre:length}$ $\leq \text{length}(ST)) do}
21: & \hspace{2em} if (\text{keyword}$(v)$) \{ \text{keyword}$(s)$ \} then
22: & \hspace{3em} ST.top.keyword$(v) = \text{keyword}$(s)$;}
23: & \hspace{3em} while (\text{pre:length $<$ d $\leq \text{length}(ST)$}) \text{push}(\text{v'});
24: & \hspace{3em} when (ST is empty, initialize ST with the next smallest node until all nodes in list $M_i$ are processed.)
25: \hspace{2em} return the matching result object nodes and their possibilities: $\{(r(RO_1), \lambda_1), (r(RO_2), \lambda_2), \ldots, (r(RO_n), \lambda_n)\}$.
26: **end for**
27: **end while**
28: **end while**
29: return the matching result object nodes and their possibilities: $\{(r(RO_1), \lambda_1), (r(RO_2), \lambda_2), \ldots, (r(RO_n), \lambda_n)\}$.

With this processed, we can return the final matching result object nodes and their possibilities.
when users input keywords \( k_1, k_2, \ldots, k_n \), we need to obtain the object nodes \( r(TRO) \) of the minimum result object trees \( RO_n \) and their possibilities \( \lambda \), and the object nodes \( r(RO) \) of the result object trees \( RO \) and their possibilities \( \sigma \).

In order to realize the query semantics, we propose the following ROStack algorithm.

The detailed procedure of ROStack is shown in Algorithm 1. When users input a set of keywords \( \{k_1, k_2, \ldots, k_n\} \), the algorithm loads and visits the keyword list \( \{M_i\} \), the minimum object tree list \( L_{RO_n} \), and the object node list \( L_{RO} \). According to the nodes \( v \), which contain keywords and minimum object nodes \( r(O_m) \) which contain keywords in the nodes of minimum object trees \( O_m \), we create the list \( L_{RO} \{v_i, \alpha_v\} \). From list \( L_{RO_n} \) we find object nodes \( r(TRO) \) of the minimum object trees which contain all keywords in the nodes of \( TRO \) and object nodes \( r(STRO) \) of the minimum object trees which contain partial keywords in the nodes of \( STRO \), and create the list \( L_2 \{r(TRO), \sigma_v, t\} \). We get \( P_{match}(\{TRO\}) \), \( P_{match}(\{STRO\}) \) from lists \( L_1 \{r(TRO), \sigma_v, t\} \) and \( L_2 \{r(STRO), \sigma_v, t\} \) and compute \( P(\{TRO\}) = P_{match}(\{TRO\}) \times \lambda \), and similarly, compute \( P(STRO) \).

Deplete the entries of \( r(TRO) \) from the index \( \{M_i\} \), as the set of \( r(TRO) \) are returned as the minimum result object nodes. We compute the SLCA object nodes among the nodes \( r(STRO) \) and nodes \( v \), which contain partial keywords. We initialize a stack \( ST \), and get node \( v \) with the smallest Dewey code in the index \( \{M_i\} \), and store node \( ST \) 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 \( v' \)), then \( v \) is reported as the set 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 \( \{[s_1], [s_2], \ldots, [s_m]\} \) which are stored in the node list \( ST \). We can get the Dewey component to denote the subtree rooted at the entries in \( ST \), whether containing \( s_m \) keyword or not. For example, there is an entry \( \{1, [0.70.80.9]\} \) at the top entry of \( ST \), the first \( 1 \) is one component of keyword and Dewey array \( [0.70.80.9] \) denotes nodes of the subtree rooted at the entries in \( ST \) contains keywords \( k_1, k_2 \) and \( k_3 \) has the existence possibility value \( P_{match} \) will be given to \( keyword[k_1] \) at the top entry of the stack. And when processing the keyword node \( v \), we make \( keyword[k_j] = \theta(k_j) \) if \( v \) contains keyword \( k_j \) and the existence possibility value \( P_{match} \) will be given to \( keyword[k_1] \) and \( keyword[k_2] \) at the same time. If node \( v \) does not contain keyword \( k_m \), then \( keyword[k_m] = 0 \). After processing node \( v \), \( keyword[k] \) will be transferred 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 \( keyword[j] = \theta(k_j) \) \( j = 1, 2, \ldots, m \) (which means all \( keyword[j] = 0 \)), then the node \( v \) stored, 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 \( v \), 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 \( Dewey(s) \notin L_{RO} \), \( s \) 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 \( Dewey(parent(s)) \in L_{RO} \). If \( Dewey(s) \notin L_{RO} \), \( s \) should find the ancestor node \( s' = ancestor(s) \) when \( ancestor(s) \in L_{RO} \) and return it as the matching result object node. We record \( s \) as the smallest object node in the SLCA. And return it as the returning result object node if \( Dewey(parent(s)) \in L_{RO} \).
can obtain the membership degrees on the paths from node \( s \) to keyword nodes, then record \( s \) into list \( L_p \), and get \( P_{\text{ref}i}(s) \) by the calculation method shown in Subsection 3.5. For example, for a SLCA object node \( s \) and its keyword nodes \( \text{STRO}_{1}, \text{STRO}_{2}, \text{STRO}_{3} \), if the \( P_{\text{ref}i}(\text{STRO}_{2}) \) \( = 0.8 \), the membership degrees on the paths from the root node \( r \) to node \( r \), \( r \) to \( s \), \( r \) to \( s \), respectively. Then, the membership degrees \( \{0.8, 0.9, 0.7\} \) and \( \{0.8, 0.9, 0.9, 0.7\} \) respectively. Then, the membership degrees on the paths from node \( s \) to node \( \text{STRO}_{2} \), node \( v_{1} \), respectively. Then, the membership degrees \( \{0.8, 0.9, 0.7\} \) and \( \{0.8, 0.9, 0.9, 0.9\} \) respectively. Then, the membership degrees on the paths from node \( s \) to node \( \text{STRO}_{2} \), node \( v_{1} \), respectively. Then, the membership degrees \( \{0.8, 0.9, 0.7\} \) and \( \{0.8, 0.9, 0.9, 0.9\} \) respectively.

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.

\[
P_{\text{precision}}(s) = \frac{|\text{RO}(s) \cap \text{STRO}(s)|}{|\text{RO}(s)|}, \text{ and on the fuzzy XML datasets } F_{\text{FX}} \text{, the average recall is } 0.933. \text{ 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 \( \text{ROstack} \) reaches 96\%, and on the fuzzy XML datasets \( FX \), the F-measure of \( \text{ROstack} \) reaches 92\%.

Table 2. F-measure.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>FD</td>
<td>96%</td>
</tr>
<tr>
<td>FX</td>
<td>92%</td>
</tr>
</tbody>
</table>

Figure 5. The precision and recall of \( \text{ROstack} \) on datasets \( FD_1, FD_2, FX_2 \) and \( FX_4 \).
can obtain the membership degrees on the paths from node $s$ to keyword nodes, then record $s$ into list $L_s$ and the precompute all the paths from the root node $r$ to node $s$, $\text{STRO}_s$, $v_1$, and $v_2$ are $[0.8, 0.9, 0.9], [0.8, 0.9, 0.9, 0.8], [0.8, 0.9, 0.9, 0.7]$ and $[0.8, 0.9, 0.9, 0.9]$ respectively. Then, the membership degrees on the paths from the root node $r$ to node $s$, $\text{STRO}_s$, $v_1$ and $v_2$ are $[0.8, 0.9, 0.9], [0.8, 0.9], [0.9, 0.9], [0.9, 0.9, 0.9]$. The algorithm proposed in this paper is implemented with Java on a laptop with 2.13 GHz Intel Core i3 with 4 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 small 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 $FD_1, FD_2, FD_3, FD_4, FD_5$, respectively. The recall measures completeness, indicating that the fraction of all correct results actually captured in the approximate answer.

### Table 1. Keyword query examples for datasets.

<table>
<thead>
<tr>
<th>ID</th>
<th>Keyword query</th>
<th>ID</th>
<th>Keyword query</th>
</tr>
</thead>
<tbody>
<tr>
<td>$DQ_1$</td>
<td>XML model, Algebra</td>
<td>$XQ_1$</td>
<td>Buyer, open_auction</td>
</tr>
<tr>
<td>$DQ_2$</td>
<td>Relational model, fuzzy, query</td>
<td>$XQ_2$</td>
<td>person40_auction</td>
</tr>
<tr>
<td>$DQ_3$</td>
<td>Information, retrieval</td>
<td>$XQ_3$</td>
<td>America, item, address</td>
</tr>
<tr>
<td>$DQ_4$</td>
<td>Fuzzy, XML, query</td>
<td>$XQ_4$</td>
<td>buyer, ship, Ed, phone</td>
</tr>
<tr>
<td>$DQ_5$</td>
<td>XML, twig, query</td>
<td>$XQ_5$</td>
<td>United States, close_auction</td>
</tr>
</tbody>
</table>

For keywords shown in Table 1, we build the structure query statement for each keyword query with the algorithm $\text{LtTwig}$ [8]. $\text{LtTwig}$ 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 $ξ_i$ which are less than or equal to the given threshold $U$ for each query from the $\text{LtTwig}$ algorithm. We obtain the matching result node object nodes $\text{RO}(v_1)$ and their possibilities $ξ_1$, from the algorithm $\text{ROstack}$. Given a keyword query $Q$, and its corresponding transformed $\text{LtTwig}$ query $LQ$, the results set of $Q$ the answers of the keyword query $Q$, denoted as $R_i$, are the approximate results. And the results set of $LQ$ (the answers of the transformed $\text{LtTwig}$ query), denoted as $R_q$, are the accurate results. Precision and recall of an algorithm are defined as follows. Precision of an algorithm is the ratio between $|R_q \cap R_i|$ and $|R_q|$, and recall is the ratio between $|R_q \cap R_i|$ and $|R_i|$. That is, $p_{\text{precision}} = |R_q \cap R_i| / |R_q|$, $p_{\text{recall}} = |R_q \cap R_i| / |R_i|$. Let $f$ denote the $F$-measure, then $f = 2 \times p_{\text{precision}} \times p_{\text{recall}}$, when $f \neq 0$, $p_{\text{precision}} = 0$.

We run keyword queries $DQ_1$–$DQ_5$ over datasets $FD_1$ and $FD_5$, and keyword queries $XQ_1$–$XQ_5$ over datasets $FX_2$ and $FX_4$. The results of precision and recall of algorithm $\text{ROstack}$ on different datasets are shown in Figure 5. The experimental results show that for different keyword queries on different fuzzy XML datasets, algorithm $\text{ROstack}$ 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 $\text{ROstack}$ reaches 96%, and on the fuzzy XML datasets $FX$, the $F$-measure of $\text{ROstack}$ reaches 92%.

### Table 2. $F$-measure.

<table>
<thead>
<tr>
<th>$F$-measure</th>
<th>$FD$</th>
<th>$FX$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{ROstack}$</td>
<td>96%</td>
<td>92%</td>
</tr>
</tbody>
</table>

![Figure 5. The precision and recall of algorithm $\text{ROstack}$ on datasets $FD_1$, $FD_2$, $FX_2$ and $FX_4$.](image)
6. Conclusion

In this paper, we propose a semantics of object-oriented keyword querying over fuzzy XML. By introducing the concept of object tree, we can get the matching result object trees which are the minimum result object trees RO containing all keywords in their tree structures, and result object trees RO containing partial keywords in their tree structures. The root nodes of RO, and RO, which are n(RO) and n(RO), 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 and rank the query results as well as to obtain higher quality query results.

Acknowledgments

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

References


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 including all keywords in their tree structures, and result object trees RO containing partial keywords in their tree structures. The root nodes of RO and RO_M, which are r(RO) and r(RO_M), 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


