Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.12785/amis/080459

Mining Approximate Keys based on Reasoning from XML Data

Yijun Liu[1,*], *Feiyue Ye*[1,*], *Jixue Liu*[2] *and Sheng He*[1]

[1] Key Laboratory of Cloud Computing & Intelligent Information Processing of Changzhou City, Jiangsu University of Technology, Changzhou, 213001, China
[2] School of Computer and Information Science, University of South Australia, Adelaide, Australia

Received: 28 Aug. 2013, Revised: 30 Nov. 2013, Accepted: 1 Dec. 2013
Published online: 1 Jul. 2014

Abstract: Keys are very important for data management. Due to the hierarchical structure and syntactic flexibility of XML, mining keys from XML data is a more complex and difficult task than from relational databases. In discovering keys from XML data there are some challenges in practice such as unclearness of keys, storage of enormous keys, efficient mining algorithms, etc. In this paper, in order to fill the gap between theory and practice, we propose a novel approximate measure of the support and confidence for XML keys on the basis of the number of null values on key paths. In the mining process, inference rules are used to derive new keys. Through the two-phase reasoning, a target set of approximate keys and its reduced set are obtained. Our research conducted experiments over ten benchmark XML datasets from XMark and four files in the UW XML Repository. The results show that the approach is feasible and efficient, with which effective keys in various XML data can be discovered.

Keywords: XML, keys, support and confidence, key implication

1 Introduction

The eXtensible Markup Language (XML) is a common form of semi-structured documents and data on the World Wide Web, and XML databases usually store semi-structured data integrated from various types of data sources. Considered to be one of the most important and challenging areas in the XML studies, integrity constraints attract much attention [1,2,3]. Much work has been done in applying traditional integrity constraints in relational databases to XML databases over the last decade, such as keys [4,5], functional dependencies [6,7,8,9], path constraints [10], inclusion constraints [11] and numerical constraints [12]. As the unique identifiers of records, keys are significantly important for database design and data management [13]. There are various forms of key specification for XML in the XML Standard and XML Schema [14,15,16], and more forms of key constraints are proposed and investigated in [17,18,19,20].

Though key definitions and their implication are suggested and researchers have analyzed their expressiveness and computational properties in theory and experiments [21,22], there are still some challenges encountered in the practical mining of XML keys, as pointed out in [23]. Firstly, due to the reason that the semi-structured XML data is usually integrated from multiple heterogeneous data sources and provides a high degree of syntactic flexibility, there could be no clear keys, that is, keys can not be expected to be satisfied at 100% in the data. Secondly, an XML database may have a large number of keys and therefore we should consider how to store them appropriately. Thirdly, it's important to find out the keys holding in a given XML dataset in an efficient way. Currently there is not much work in the literature in practical mining of keys from XML data. Gösta Grahne et al. in [23] address this topic adopting a data mining point of view. To over the first obstacle they propose discovering approximate keys which need not be satisfied in the whole XML data and allow a violation in a small part. The approximation of a key expression is measured by the support and confidence similar to those of association rules. For the second and third problem, a partial order on the set of all keys is defined and finally a reduced set of approximate keys are obtained. In this paper, we also investigate the issue of mining approximate keys from XML data. Considering the

* Corresponding author e-mail: yijunliu@vip.sina.com, yfy@jsut.edu.cn

characteristics of XML data, we present an alternative general approach for mining keys. We use the most influential proposal for XML keys by Buneman et al. in [19,20]. On the basis of an XML tree model, they propose not only the concepts of absolute keys and relative keys independent of schema, which are in keeping with the hierarchically structured nature of XML, but also a sound and complete axiomatization for key implication. By using these inference rules, the keys can be reasoned about efficiently.

The rest of this paper is organized as follows. Section 2 recalls and discusses some basic notions of XML keys used through this work. Section 3 proposes the approximate measures for XML keys and Section 4 exploits the target set of keys and its minimal cover. Section 5 contains our implementation of the mining method and the essential ideas of our algorithms. Section 6 contains details of experiments that were performed over publicly available XML data to asses the effectiveness and efficiency of our approach. In Section 7 we conclude this paper highlighting the contributions and future enhancements.

2 Preliminary definitions

2.1 The tree model for XML

An XML document is typically modeled as a labeled tree. A node of the tree represents an element, attribute or text(value), and edges represent the nested relationships between nodes. Node labels are divided into three pairwise disjoint sets: E the finite set of element tags, A the finite set of attribute names, and the singleton $\{S\}$, where S represents text (PCDATA). An XML tree is formally defined as follows.

Definition 2.1. An XML tree is a 6-tuple $T = (r, V, lab, ele, att, val)$, where (1) r is the unique root node in the tree, i.e. the document node, and $r \in V$. (2) V is a finite set containing all nodes in T. (3) lab is a function from V to $E \cup A \cup \{S\}$. For each $v \in V$, v is an element if $lab(v) \in E$, an attribute if $lab(v) \in A$, and a text node if $lab(v) = S$. (4) Both ele and att are partial functions from V to V^*. For each $v \in V$, if $lab(v) \in E$, $ele(v)$ is a sequence of elements and text nodes in V and $att(v)$ is a set of attributes in V; For each $v' \in ele(v)$ or $v' \in att(v)$, v' is the child of v and there exists an edge from v to v'. (5) val is a partial function from V to string, mapping each attribute and text node to a string. For each $v \in V$, if $lab(v) \in A$ or $lab(v) = S$, $val(v)$ is a string of v.

2.2 Path expressions

In the XML tree, a node is uniquely identified by a path of node sequence. Because the concatenation operation does not have a uniform representation in XPath used in XML-Schema, Buneman et al.[20] have proposed an alternative syntax. For identifying nodes in an XML tree, we use their path languages called PL_s, PL_w and PL, where ε represents the empty path, l is a node label in $E \cup A \cup \{S\}$, and "." is the concatenation of two path expressions. In PL_s, a valid path is the empty path or the sequence of labels of nodes. PL_w allows the symbol "_" which can match any node label. PL includes the symbol "_*" matching any sequence of node labels. The notation $P \subseteq Q$ denotes that the language defined by P is a subset of the language defined by Q. For the path expression P and the node n, the notation $n[P]$ denotes the set of nodes in T that can be reached by following a path that conforms to P from n. The notation $[P]$ is the abbreviation for $r[P]$, where r is the root in T. The notation $|P|$ denotes the number of labels in the path. $|\varepsilon|$ is 0, and "_" and "_*" are both counted as labels with length 1. The paths which are merely sequences of labels are called simple paths.

2.3 Definitions on keys

We firstly recall and discuss definitions on keys for XML from [19,20] and further propose definitions of the strong key and the minimal key.

Definition 2.2. A key constraint φ for XML is an expression $(Q', (Q, \{P_1, ..., P_k\}))$ where Q', Q and $P_i (1 \le i \le k)$ are path expressions. Q' is called the context path, Q is called the target path, and P_i is called the key path of φ. If $Q' = \varepsilon$, φ is called an absolute key, otherwise φ is called a relative key. The expression (Q, S) is the abbreviation of $(\varepsilon, (Q, S))$, where $S = \{P_1, ..., P_k\}$.

Definition 2.3. Let $\varphi = (Q', (Q, \{P_1, ..., P_k\}))$ be a key expression. An XML tree T satisfies φ, denoted as $T \models \varphi$, if and only if for every $n \in [Q']$, given any two nodes $n_1, n_2 \in n[Q]$, if for all i, $1 \le i \le k$, there exist $z_1 \in n_1[P_i]$ and $z_2 \in n_2[P_i]$ such that $z_1 =_v z_2$, then $n_1 = n_2$. That is,

$$\forall n_1, n_2 \in n[Q]$$

$$\left(\left(\bigwedge_{1 \le i \le k} \exists z_1 \in n_1[P_i] \exists z_2 \in n_2[P_i] (z_1 =_v z_2) \right) \rightarrow n_1 = n_2 \right)$$

The definition 2.3 of keys is quite weak. The key expression could hold even though key paths are missing at some nodes. This definition is consistent with the semi-structured nature of XML, but does not mirror the requirements imposed by a key in relational databases, i.e. uniqueness of a key and equality of key values. The definition 2.4 meets both two requirements.

Definition 2.4. Let $\varphi = (Q', (Q, \{P_1, ..., P_k\}))$ be a key expression. An XML tree T satisfies φ, if and only if for any $n \in [Q']$, (1) For any n' in $n[Q]$ and for all $P_i (1 \le i \le k)$, P_i exists and is unique at n'. (2) For any two nodes $n_1, n_2 \in n[Q]$, if $n_1[P_i] =_v n_2[P_i]$ for all i, $1 \le i \le k$, then $n_1 = n_2$.

The definition 2.4 of keys is stronger than the definition 2.3, and the key paths are required to exist and be unique. Note that there probably are empty tags in XML documents. A consequence is that some nodes in $n'[P_i]$ are null-valued, which is allowed in the definition 2.4. However the attributes of the primary key in relational databases are not allowed null. Here we explore a strong key definition which captures this requirement.

Definition 2.5. Let $\varphi = (Q', (Q, \{P_1, ..., P_k\}))$ be a key expression. An XML tree T satisfies φ, if and only if for any $n \in [Q']$, (1) For any n' in $n[Q]$ and for all $P_i (1 \leq i \leq k)$, P_i exists and is unique at n', and all nodes in $n'[P_i]$ are not null valued. (2) For any two nodes n_1, $n_2 \in n[Q]$, if $n_1[P_i] =_v n_2[P_i]$ for all i, $1 \leq i \leq k$, then $n_1 = n_2$.

In the definition 2.5 of strong keys, the key paths are required to exist, be unique and not have a null value. Note that in relational databases, a tuple can be identified by more than one group of key attributes. Analogously, given a context path Q' and a target path Q in the XML tree T, there exist probably multiple sets S of key paths such that $T \models (Q', (Q, S))$.

Definition 2.6. Let $\varphi = (Q', (Q, S))$ be a key expression satisfied in the XML tree T. If for any key expression $\varphi' = (Q', (Q, S'))$ satisfied in T, $|S| \leq |S'|$, then φ is called the minimal key.

In other words, a minimal key has the least number of key paths with the determined Q' and Q. Note that there are probably multiple minimal keys with the fixed Q' and Q.

2.4 Node equality and value equality

The key definition involves node equality and value equality.

1. Value equality

Value equality in XML-Schema is restricted to text nodes. Buneman et al.[19] propose a more general way of describing value equality by using tree equality. The example they provided is that as a key for *person* nodes, *name* may have a complex structure consisting of *first-name* and *last-name* subelements. However, since S in $(Q', (Q, S))$ is the set consisting of multiple key paths, the key with a complex structure can be decomposed to several simple key paths. For *person* nodes, the union of name.first-name and name.last-name can substitute for *name*. Hence in this paper we use the equality of text nodes but not tree quality. $n_1 =_v n_2$ denotes that n_1 and n_2 are value equal.

2. Node equality

In an XML tree, a path starting from the root uniquely identifies a node. The nodes n_1 and n_2 are equal, denoted as $n_1 = n_2$, indicating that n_1 and n_2 represent one node in the tree. In a relational database, the key uniquely identifies a record. In XML data, the key is analogously considered to uniquely identify a node, that is, the two distinct nodes n_1 and n_2 certainly cannot have identical values on key paths. However, XML data is flexibly organized, and different nodes may indicate the same entity in real-world. In the example in Figure 1, both the *teacher* nodes represent Li Wen. Node equality needs more consideration especially when discovering absolute keys in the large range of the entire document.

```
<root>
  <course>
   <cno> 009 </cno>
   <cname>Data Structure</cname>
   <teacher>
     <name> Li Wen </name>
     <gender> Male </gender>
     <age> 35 </age>
   </teacher>
  </course>
  <course>
   <cno> 010 </cno>
   <cname>Operating System</cname>
   <teacher>
     <name> Li Wen </name>
     <gender> Male </gender>
   </teacher>
  </course>
</root>
```

Fig. 1: An example of XML document

We redefine the node equality. If one of the two nodes n_1 and n_2 contains all information of the other, the two nodes are considered to represent the same entity, and consequently they are equal. In XML trees, Let $T(n)$ be the subtree rooted at node n. n_1 and n_2 represent the same entity if and only if $T(n_1)$ is the subgraph of $T(n_2)$, or vice versa, where n_1 matches n_2.

3 Approximate measures for keys

Due to the unclearness of keys in XML data, Gösta Grahne et al. [23] propose approximate keys with measures of their support and confidence similar to those of association rules [24]. The support and confidence is defined respectively according to the number of branches of key paths and the number of distinct values on key paths. Here we give the measures in another way. For convenience we consider every element node having a child of text node. In particular, if the element has no text, its child text node is treated as null-valued.

3.1 The support of keys

Given a node n, let $size(n)$ be the number of child nodes of n, indicating the size of n. For a key expression $\varphi = (Q', (Q, S))$ in an XML tree, the size of φ in the tree T is

$$size(T, \varphi) = \max\{size(n) \mid n \in [Q'.Q]\} \quad (1)$$

The size of φ is the maximum size of nodes reached by $Q'.Q$ of φ from root. For those nodes in $[Q'.Q]$ with extremely small size, e.g. those leaf nodes without children whose size is zero, it becomes meaningless to find their key paths. Therefore *min_size* is set to be the minimum threshold of node size. The support of φ in the whole XML tree T is

$$support(T, \varphi) = \begin{cases} 0 & \text{if } size(T, \varphi) < min\_size \\ \sum_{n' \in [Q']} |n'[Q]| & \text{otherwise} \end{cases} \quad (2)$$

The support of φ is assigned to 0 when the size of φ is less than *min_size*. Consequently, the category of nodes with too small size is abandoned. When the size of φ is not less than *min_size*, the support of φ is the number of nodes in $[Q'.Q]$.

3.2 The confidence of keys

Section 2.3 gives several definitions on keys for XML, among which the definition 2.3 is the weakest, the definition 2.4 is stronger, and the definition 2.5 is the strongest. The choice of key definitions will affect the final mining results and thus should be ultimately determined by practice. In the process of mining keys, when choosing the strong key definition, probably some meaningful keys are not discovered due to missing information in the XML documents. While the weak key definition probably results in meaningless key paths. Therefore we utilize the confidence for a compromise.

Given a key expression $\varphi = (Q', (Q, S))$ in an XML tree T, $S = \{P_1, P_2, ..., P_k\}$. Define a two-valued function $vals(n, P)$, where n is a node and P a path expression. If there exists $z \in n[P]$ and z is not null valued, $vals(n, P)$ is assigned to 1, and otherwise 0. The confidence of φ in the tree T is

$$confidence(T, \varphi) = \frac{\min\{\sum_{n' \in [Q']} \sum_{n \in n'[Q]} vals(n, P_i) | P_i \in S\}}{support(T, \varphi)} \quad (3)$$

In particular, we set $confidence(T, \varphi) = 1$ when $support(T, \varphi) = 0$.

The support of φ is defined associated with target paths and the confidence associated with key paths. The key paths of φ are computed if φ satisfies the specified support threshold *min_sup*. Specify the confidence threshold min_conf, allowing null values or missing paths but confine their number. In particular, with $min\_conf = 0$, null values or missing paths have no impact on satisfaction of φ, and with $min\_conf = 1$, they are not allowed.

3.3 The measures of absolute keys

An absolute key is a special case of a relative key. Given an absolute key expression $\varphi = (Q, S)$ in the XML tree T, where Q' is ε and omitted. The size, support and confidence of φ are

$$size(T, \varphi) = \max\{size(n) | n \in [Q]\} \quad (4)$$

$$support(T, \varphi) = \begin{cases} 0 & \text{if } size(T, \varphi) < min\_size \\ |[Q]| & \text{otherwise} \end{cases} \quad (5)$$

$$confidence(T, \varphi) = \frac{\min\{\sum_{n \in [Q]} vals(n, P_i) | P_i \in S\}}{support(T, \varphi)} \quad (6)$$

3.4 An illustration of approximate measures

We now illustrate the approximate measures by an example. Table 1 shows the support and confidence of some key expressions in the XML document in Figure 2 by using two measures. The sup_1 and $conf_1$ are calculated as discussed above, and the sup_2 and $conf_2$ are calculated as introduced by Gösta Grahne et al [23].

The measure results for the third and fifth key expression are significantly different. The confidences of (_*.course, {tutor}) are 40% and 100% receptively. Considering elements of *tutor* only exist in two elements of *course*, it is not reasonable to regard *tutor* as the key of *course*, and hence the first result is more acceptable. For the relative key expression (courses.course, (teacher, {name})), two confidences vary significantly. Its $conf_2$ is the minimum confidence of all subtrees rooted at the nodes in [courses.course], while in our measure all subtrees contribute to the ultimate confidence $conf_1$. Hence our measure of the confidence is more comprehensive and anti-noise.

In addition, the second key expression (_*.course, {cname, dept}) also has distinct measure values. The second support is the number of key path branches in those subtrees rooted at the nodes in [_*.course], while the support in our measure depends on the number of target path branches but not key path branches. As a consequence, our approach needn't compute the support for every key path combination and hence has a smaller time cost.

```
<courses>
<course>
  <cno> 009 </cno>
  <cname>Data Structure</cname>
  <dept> CSE </dept>
  <teacher>
    <name> Li Hongwei </name>
    <title> Prof. </title>
    <age> 35 </age>
  </teacher>
  <tutor>Cai Qiuru</tutor>
  <tutor>Luo Ye</tutor>
  <tutor>Li Bingzhang</tutor>
</course>
<course>
  <cno> 010 </cno>
  <cname>Operating System</cname>
  <dept> CSE </dept>
  <teacher>
    <name> Wang Wei </name>
    <title> Prof. </title>
    </teacher>
  <teacher>
  <name> Liu Daoyu </name>
  </teacher>
  <tutor>Jiang Hongfen</tutor>
  <tutor>Wang Bo</tutor>
  <tutor>Chen Dan</tutor>
</course>
```

```
<course>
  <cno> 011 </cno>
  <cname>Numerical Methods </cname>
  <dept> CSE </dept>
  <teacher>
    <name> Lin Chengsen </name>
  </teacher>
</course>
<course>
  <cno> 012 </cno>
  <cname>Communications</cname>
  <dept> </dept>
  <teacher>
    <name> Li Wen </name>
    <title> A.P. </title>
  </teacher>
  <teacher>
    <name> Zhu Lin </name>
  </teacher>
</course>
<course>
  <cno> 110 </cno>
  <cname>Numerical Methods</cname>
  <dept> MATH </dept>
  <teacher>
    <name></name>
    <title> Prof. </title>
  </teacher>
</course>
</courses>
```

Fig. 2: An example of XML document

Table 1: The support and confidence by two measures

no.	key expression	sup_1	$conf_1$	sup_2	$conf_2$
1	(_∗.course, {cno})	5	5/5	5	5/5
2	(_∗.course,{cname,dept})	5	4/5	10	9/10
3	(_∗.course,{tutor})	5	2/5	6	6/6
4	(_∗.teacher,{name})	7	6/7	7	6/7
5	(courses.course, (teacher, {name}))	7	6/7	7	0/1

4 The target set of keys and its minimal cover

Philosophically, if a tuple is treated as representing some real world entity, the key provides an invariant connection between the tuple and entity. In relational databases, the key uniquely identifies a tuple and is the link between the tuple and entity [19]. In XML databases, the node set of $n[Q](n \in [Q'])$ is analogous to a set of tuples in a relation and the key paths to the key attributes of the relation. In an empirical study of mining keys, we wish to find out the accurate and meaningful link between the tuple and its corresponding entity. Moreover, an XML document possibly has enormous number of keys and thus it is space consuming to store all of them. Here we suggest a target set of keys and its reduced representation, i.e. its non-redundant cover.

4.1 The target set of keys to be mined

Due to the flexibility of XML data, the nodes associated with a special category of entities may appear at an arbitrary position in the XML tree. The keys that can accurately and globally identify those nodes are the absolute keys in form of $(\_*.l, S)$, where l is an element label.

Proposition 4.1. If $T \models (\_*.l,S)$, and for the pairwise context path Q' and target path Q in T, Q has the suffix of l, then $T \models (Q',(Q,S))$.

$[Q'.Q] \subseteq [\_*.l]$ since Q has the suffix of l. Obviously, S can certainly identify the l nodes in the scope within the subtree if it can identify them in the entire XML tree. The absolute keys of $(\_*.l,S)$ are of special significance.

This paper aims to find out the set K_G of key expressions holding in T. K_G contains all absolute keys of (Q,S) and relative keys of $(Q',(Q,S))$, where $S = \{P_1,...,P_k\}$. The paths of keys are restricted to simple paths and the key paths for Q' and Q can globally identify a special category of nodes. More precisely, we have the following requirements:

(1) The path expressions of Q', Q and $P_i(1 \leq i \leq k)$ are all in PL_s.

(2) Let l be the suffix node label of Q. If $T \models (Q',(Q,S))$ or $T \models (Q,S)$, then $T \models (\_*.l,S)$.

Let K_I be the set of key expressions in form of $(\_*.l,S)$ satisfied in T, that is $K_I = \{(\_*.l,S) \mid T \models (\_*.l,S)\}$. K_G can be obtained by two-phase reasoning starting from K_I.

4.2 The selected inference rules

For convenience we recall some terminology about logical implication on keys by Gösta Grahne et al.[23]. Given two key expressions φ_1 and φ_2, if every XML tree T that satisfies φ_1 also satisfies φ_2, then φ_1 logically implies φ_2, denoted as $\varphi_1 \models \varphi_2$. Given a set K of key expressions and a key expression φ, if every XML tree T that satisfies all key expressions in K also satisfies φ, then K logically implies φ, denoted as $K \models \varphi$. K^+ denotes the set of key expressions implied by K, that is $K^+ = \{\varphi \mid K \models \varphi\}$, and K^+ is called the closure of K. For a key expression $\varphi \in K$, if $K\backslash\{\varphi\} \models \varphi$, that is $\{K\backslash\{\varphi\}\}^+ = K^+$, φ is called a redundant key in K. For the target set K_G of key expressions, we wish to find a non-redundant set K of key expressions such that $K^+ = K_G$. K is called a minimal cover for K_G.

Buneman et al.[20] propose the sound and complete axiomatization, solving the problem of key implication for XML. In order to compute the K_G starting from the initial set K_I of absolute keys, only two inference rules are needed.

$R_1 : (Q',(Q_1,S)), Q_2 \subseteq Q_1 \Rightarrow (Q',(Q_2,S))$

$R_2 : (Q',(Q_1.Q_2,S)) \Rightarrow (Q'.Q_1,(Q_2,S))$

R_1 is the rule of target-path-containment and R_2 is the rule of context-target. Let K be the set of absolute keys in K_G. We firstly compute K from K_I by applying rule R_1, and then K_G can be obtained from K by applying rule R_2.

4.3 The minimal cover and reasoning

Now we are concerned with showing that K_G can be inferred from K_I by the rules of R_1 and R_2. A two-phase inference process for keys is as follows.

1. Phase I: Infer K from K_I

K can be inferred from K_I with R_1. The context path Q' in R_1 is ε since K_I and K contain only absolute keys.

The rule R_1 is complete for K. That is, for every key expression $\varphi \in K$, φ is inferable from K_I by applying R_1. For a key expression (Q,S) in K, Q is a simple path and has the suffix of node label l. Because $(\_*.l,S) \in K_I$ and $Q \subseteq \_*.l$, (Q,S) can be inferred from $(\_*.l,S)$ by using R_1.

It should be noted that if we set the thresholds of support and confidence, the keys inferred from K_I are probably fake keys which don't satisfy the specified thresholds.

2. Phase II: infer K_G from K

Here we prove that K is a minimal cover for the target set K_G of key expressions by using R_2.

Proposition 4.2. By applying the rule R_2, K is a minimal cover for K_G.

Proof. We will show that $K^+ = K_G$ and K is non-redundant.

(1) $K^+ = K_G$. The set K_G contains absolute keys and relative keys. In the following, we distinguish two different cases.

Case 1. For any absolute key (Q,S), $(Q,S) \in K_G$ iff. $(Q,S) \in K$ due to the fact that the subset K of K_G contains all absolute keys of K_G.

Case 2. Consider the keys of $\varphi_1 = (Q'.Q,S)$ and $\varphi_2 = (Q',(Q,S))$. φ_2 can be inferred from φ_1 by using R_2. Due to the definitions of support and confidence it follows that $support(T,\varphi_1) = support(T,\varphi_2)$ and $confidence(T,\varphi_1) = confidence(T,\varphi_2)$. Hence, no fake key is generated in reasoning by the rule R_2. Here that $\varphi_1 \in K$ iff. $\varphi_2 \in K_G$ results from the following reasons. On the one hand, if $\varphi_1 = (Q'.Q,S)$, equivalent to $(\varepsilon,(Q'.Q,S))$, is in K, by the rule R_2 we obtain $\varphi_2 = (\varepsilon.Q',(Q,S)) = (Q',(Q,S))$. $Q'.Q$ is in PL_s, and then Q' and Q are both in PL_s. Hence φ_2 is in K_G by the requirements of keys in K_G. On the other hand, if φ_2 is in K_G, by the rule R_2, φ_2 can be inferred from φ_1, equivalent to $(\varepsilon,(Q'.Q,S))$. Q' and Q are both in PL_s, and then $Q'.Q$ is also in PL_s. Hence φ_1 is in K.

(2) K is non-redundant. The set K contains no relative keys. For any key expression φ in K, φ is an absolute key and is not inferable from $K\backslash\{\varphi\}$ by applying R_2. Therefore K is non-redundant.□

5 Key mining process in detail

5.1 A sketch of key mining process

There could be a large number of key expressions in K_G. Due to that K is a compact representation for K_G by proposition 4.2 in Section 4, this paper only discovers the set K and hence solves the storage problem of enormous keys. Here we propose the procedure to obtain the minimal cover K of K_G. A two-step key mining process is

followed, consisting of discovery and inference. In practice, the measures of support and confidence can be used if needed.

1. The discovery step. We find out K_I, the original set of key expressions in form of $(\_*.l,S)$ which hold in the XML tree T. This step is divided into two phases. In the first phase, the target paths of $\_*.l$ satisfying the support threshold are generated. In the second phase, for each target path, the set S of key paths which satisfies the confidence threshold is generated. Note that here only minimal keys are discovered.

2. The inference step. The set K is inferable from K_I with R_1. Apply the rule R_1 to every key expression $(\_*.l,S)$ in K_I. By traversing the XML tree, the path set of $QS = \{Q_1,Q_2,...,Q_n\}$ is generated, where Q_i is in PL_s and $Q_i \subseteq \_*.l$, that is, Q_i is a simple path with the suffix of l. The key expressions of $(Q_i,S)(1 \leq i \leq n)$ are obtained. Here we mine approximate keys with the support and confidence thresholds, and therefore need to examine whether the keys of (Q_i,S) satisfy the thresholds and remove those fake keys.

5.2 Algorithm

Here we give the algorithms of finding target paths and finding key path sets in the discovery step.

5.2.1 Generation of target paths

To discover those absolute key expressions in form of $(\_*.l,S)$, $S = \{P_1,P_2,...,P_n\}$, it needs to find all target paths with support larger than *min_sup* firstly. The symbol "$\_*$" matches any path and the list of node labels is obtained by traversing the whole tree preorderly. Let $L = \{l_i\}$, $C = \{c_i\}$ and $S = \{s_i\}$ be three lists, where l_i denotes the label of element nodes, c_i denotes the number of nodes with label l_i, and s_i denotes the maximum size of the element nodes with label l_i. Let $size(e)$ be the size of the element node e, i.e. the number of children of e. The algorithm TargetPath_gen of finding target paths is outlined in Figure 3, where for each target path $\_*.l_i \in TP$, the support of $(\_*.l_i,S)$ is sup_i, $sup_i \in SP$.

5.2.2 Generation of key path sets

Generation of key path sets for a fixed target path has two major steps. The first step is to find candidate key paths. The second step is to check whether a set of key expressions are satisfied in an XML document and select those suitable subsets of candidates.

For an key expression $\varphi = (Q,S)$ with the fixed target path Q in the XML tree T, let C_P be the set of candidate key paths, $C_P = \{P'_1,P'_2,...,P'_m\}$. $P'_i \in C_P$ if and only if:

$$\frac{\sum_{n\in[Q]} vals(n,P'_i)}{support(T,\varphi)} \geq min\_conf$$

By the definition of confidence, $confidence(T,\varphi) \geq min\_conf$ if and only if $S \subseteq C_P$. Therefore S can be generated by checking whether the subsets of C_P are the set of key paths for φ. The algorithm KeySet_gen of discovering the key set $K_I = \{(\_*.l_i,S)\}$ is summarized in Figure 4. Note that the keys produced by this algorithm are minimal keys.

6 Experimental study

In this section, we perform experiments on two categories of XML datasets to evaluate our approach. The first category of datasets is from XMark which is an XML benchmark project, and the second is from UW XML Repository. The algorithms are implemented in the C/C++ language and programs are executed on Microsoft Visual C++ 6.0. All experiments are conducted on computers with an INTEL Core 2DuoProcessorG630 and 3G memory, running Windows XP.

6.1 Experiments on the XMark datasets

XML data generator *xmlgen* produces scaled documents according to the DTD specified in the project of XMark. Further details are provided on http://www.xml-benchmark.org. Ten XML documents D1-D10 with different size are generated as benchmark datasets for our experiments.

Here some experiments have been performed to monitor the time needed to discover the keys in the set K_I and infer new keys from K_I to construct the set K, i.e. the minimal cover for K_G. Figure 5 shows the empirical results obtained on the ten benchmark XML documents with different size, given the minimum support threshold $min\_sup = 30$ with $min\_size = 2$ and the minimum confidence threshold $min\_conf = 80\%$. The discovery time and the inference time indicate the running time of discovering the keys in K_I and inferring the set K from K_I respectively, and their sum is the total time. These results indicate the good scalability of our mining approach since the total running time is nearly linear in the size of the XML document. The increase of the inference time is slight while that of the discovery time is much steeper. The discovery step of finding K_I is much more time consuming, compared with which the time cost of inference is negligible.

Figure 6 shows the performance of the total mining time on the documents D1, D5 and D10 with various confidence thresholds, given $min\_sup = 30$. All the three time curves are downward generally. The curves seem smooth when min_conf is less than 85%. The reason is that almost all the keys with confidence greater than 50% have a confidence not less than 85%. But the mining time decreases slightly when min_conf is greater than 85%, because the keys with confidence not satisfying the

Input: XML tree T, node size threshold *min_size*, support threshold *min_sup*.
Output: the set TP of target paths, the set SP of supports.
Begin
  initialize L, C and S to $\emptyset$;
  k=1;
  for each node element e read from T
    if there exists $l_i \in L$, such that $l_i = lab(e)$
      c_i++;
      $s_i = \max(s_i, size(e))$;
    else
      add $l_k = lab(e)$ to L, $c_k = 1$ to C and $s_k = size(e)$ to S;
      k++;
  for each label $l_i \in L$
    support = $(s_i < min\_size)$? 0 : c_i;
    if support < *min_sup*
      delete l_i from L, c_i from C and s_i from S;
  return TP={_*.l_i | $l_i \in L$) and SP={sup_i | sup_i=c_i and $c_i \in C$ };
End

Fig. 3: Algorithm TargetPath_gen

Input: XML tree T, the set TP of target paths, the set SP of supports, confidence threshold *min_conf*.
Output: the key set K_I.
Begin
  for each target path _*.$l \in TP$
  //sp is the support corresponding to _*.l, $sp \in SP$
    C_P=**find_candidate_keypath**(T, _*.l, sp, *min_conf*);
    initialize the set KP to $\emptyset$;
    k=1;
    while ($KP == \emptyset$ and $k \leq |C_P|$)
      for each k-subset S of C_P
        if S is a set of key paths for _*.l
          add S to KP;
      k++;
    for each key path set $S \in KP$
      add the key expression (_*.l, S) to K_I;
  return K_I;
End

Fig. 4: Algorithm KeySet_gen

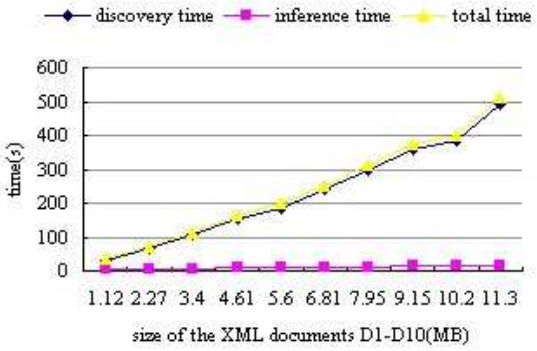

Fig. 5: Performance on the ten XMark datasets(D1-D10)

Fig. 6: Performance on D1, D5 and D10, given *min_sup* = 30

Table 2: Original keys discovered in D10

no.	key	support	confidence
1	(_*.item, {id})	2175	100%
2	(_*.mail, {from})	2139	100%
3	(_*.mail, {text})	2139	90%
4	(_*.category, {id})	100	100%
5	(_*.category, {name})	100	100%
6	(_*.edge, {from, to})	100	100%
7	(_*.person, {id})	2550	100%
8	(_*.address, {city, street})	1256	100%
9	(_*.address, {street, zipcode})	1256	100%
10	(_*.open_auction, {id})	1200	100%
11	(_*.open_auction, {itemref.item})	1200	100%
12	(_*.bidder, {date, increase, personref.person, time})	6182	100%
13	(_*.interval, {start, end})	1200	100%
14	(_*.closed_auction, {itemref.item})	975	100%

Table 3: The minimal cover for keys of D10

no.	key	support	confidence
1	(site.regions.africa.item, {id})	55	100%
	(site.regions.asia.item, {id})	200	100%
	(site.regions.australia.item, {id})	220	100%
	(site.regions.europe.item, {id})	600	100%
	(site.regions.namerica.item, {id})	1000	100%
	(site.regions.samerica.item, {id})	100	100%
2	(site.regions.africa.item.mailbox.mail, {from})	53	100%
	(site.regions.asia.item.mailbox.mail, {from})	210	100%
	(site.regions.australia.item.mailbox.mail,{from})	212	100%
	(site.regions.europe.item.mailbox.mail, {from})	590	100%
	(site.regions.namerica.item.mailbox.mail, {from})	986	100%
	(site.regions.samerica.item.mailbox.mail, {from})	88	100%
3	(site.regions.africa.item.mailbox.mail, {text})	53	89%
	(site.regions.asia.item.mailbox.mail, {text})	210	90%
	(site.regions.australia.item.mailbox.mail, {text})	212	92%
	(site.regions.europe.item.mailbox.mail,{text})	590	91%
	(site.regions.namerica.item.mailbox.mail, {text})	986	90%
	(site.regions.samerica.item.mailbox.mail, {text})	88	90%
4	(site.categories.category, {id})	100	100%
5	(site.categories.category,{name})	100	100%
6	(site.catgraph.edge, {from, to})	100	100%
7	(site.people.person, {id})	2550	100%
8	(site.people.person.address, {city, street})	1256	100%
9	(site.people.person.address, {street, zipcode})	1256	100%
10	(site.open_auctions.open_auction, {id})	1200	100%
11	(site.open_auctions.open_auction, {itemref.item})	1200	100%
12	(site.open_auctions.open_auction.bidder, {date, increase, personref.person, time})	6182	100%
13	(site.open_auctions.open_auction.interval, {start, end})	1200	100%
14	(site.closed_auctions.closed_auction, {itemref.item})	975	100%

threshold are removed, saving a small amount of inference time.

Table 2 tabulates the original set K_I of key expressions discovered in the XML file D10 and their support and confidence. Table 3 tabulates the set K of key expressions inferred from that in Table 2, i.e. the minimal cover for K_G. Due to the regularness of the data generated by *xmlgen*, most keys have confidence 100%. Comparing the document D10 with Table 2 and Table 3, it is obvious that almost all meaningful keys in the file have been mined, which shows that the key mining method presented in this paper is effective.

6.2 Experiments on the UW datasets

In mining keys from UW datasets, we set a minimum length threshold for key paths. In other words, a depth bound is used to prevent the search process from running away toward nodes of unbounded depth from the node at the target path. We select four xml files of SIGMODRecord, reed, uwm and wsu with various node depth. Further details are provided on http://www.cs.washington.edu/research/xmldatasets/. The *min_sup* is set to 5 with *min_size* = 2 for all, while the *min_conf* is set to 80% for SIGMODRecord, reed and wsu and 50% for uwm.

The time costs on the four test files are shown in Figure 7-10, where "-1" represents a unbounded depth, and the keys discovered and inferred are listed in Table 4 and Table 5. In each of these figures the curves rise at first and then flatten with the depth bound increasing. In the case of SIGMODRecord with max-depth 6 and avg-depth 5.14107, the key paths discovered with unbounded depth are all have length 1, and therefore the depth bound 1 is sufficient to find out all key paths. From Figure 7, setting the depth bound to a smaller value helps reduce the discovery time significantly. Intuitively, this is due to that the attribute or element nodes as keys are usually close to the target nodes in hierarchical XML data.

7 Conclusions

At present, many achievements have already been attained on the issue of XML keys. However further studies are still necessary for some practical problems in mining keys from real XML data. In this paper, we discuss definitions on keys for XML without considering foreign keys and DTDs and have proposed a practical

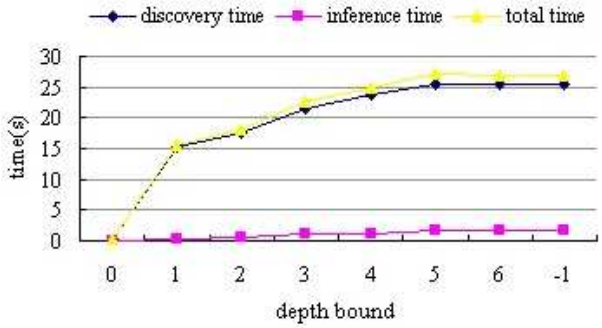

Fig. 7: Performance on SIGMODRecord.xml

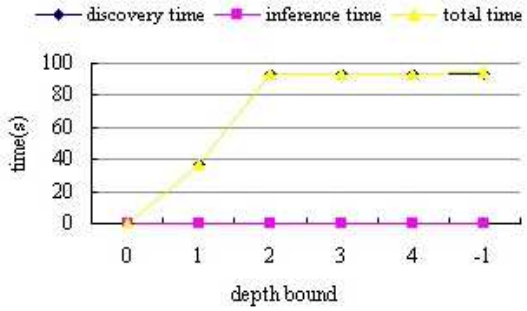

Fig. 8: Performance on reed.xml

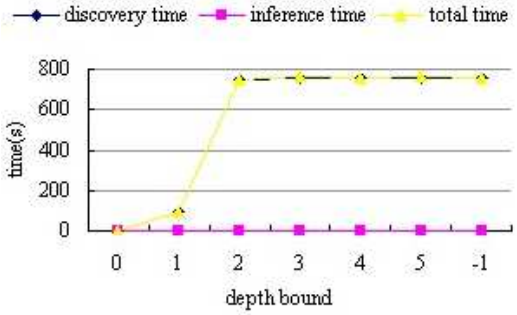

Fig. 9: Performance on uwm.xml

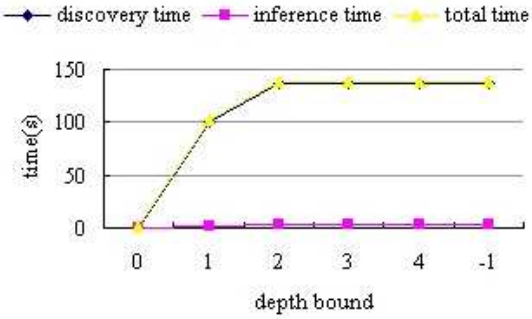

Fig. 10: Performance on wsu.xml

Table 4: Original keys discovered in four UW XML files

filename	key	support	confidence
SIGMODRecord	(_*.issue, {number, volume})	67	100%
	(_*.article, {endPage, initPage, title})	1504	100%
reed	(_*.course, {days, reg_num})	703	98%
	(_*.time, {start_time, end_time})	703	98%
	(_*.place, {Building, room})	703	93%
uwm	(_*.course_listing, {course})	2112	100%
	(_*.hours, {start, end})	4575	53%
	(_*.bldg_and_rm, {bldg, rm})	4575	74%
wsu	(_*.course, {sln})	3924	95%
	(_*.place, {bldg, room})	3924	100%

Table 5: Minimal covers for keys in four UW XML files

filename	key	support	confidence
SIGMODRecord	(SigmodRecord.issue, {number, volume})	67	100%
	(SigmodRecord.issue.articles.article, {endPage, initPage, title})	1504	100%
reed	(root.course, {days, reg_num})	703	98%
	(root.course.time, {start_time, end_time})	703	98%
	(root.place, {Building, room})	703	93%
uwm	(root.course_listing, {course})	2112	100%
	(root.course_listing.section_listing.hours, {start, end})	4575	53%
	(root.course_listing.section_listing.bldg_and_rm, {bldg, rm})	4575	74%
wsu	(root.course, {sln})	3924	95%
	(root.course.place, {bldg, room})	3924	100%

approach for mining keys from XML data. Due to that keys are usually not be satisfied at 100% in XML data which has a hierarchical and flexible structure and is usually integrated from heterogeneous sources, we apply an improved approximate measure of the support and confidence for key expressions. To find out all satisfiable absolute keys and relative keys with simple target and key paths in an XML tree, an initial set of keys are discovered firstly and then two phases of reasoning are used. A reduced set of all target keys are obtained after the first phase of reasoning. The results of our experiments on ten benchmark datasets of XMark and the four chosen files of UW repository show the effectiveness and feasibility of our approach. A point worth noting is that new keys can be reasoned about efficiently during the mining process while the discovery of initial keys is time consuming. In the future we wish to improve computational efficiency of the discovery stage with alternative approaches and further tunings.

Acknowledgement

This work is supported by National Natural Science Foundation of China (no.61142007), Applied Basic Research Program of Jiangsu University of Technology (no.KYY10059) and the Key Laboratory of Cloud Computing & Intelligent Information Processing of Changzhou City under Grant No. CM20123004. Furthermore, we are indebted to the support and encouragements received from the staff and colleagues of the school of computer engineering.

References

[1] D. Suciu, On database theory and XML, SIGMOD Record, **30**, 39-45 (2001).
[2] W. Fan, L. Libkin. On XML integrity constraints in the presence of DTDs, J. ACM, **49**, 368-406 (2002).
[3] W. Fan. XML constraints, in: DEXA Workshops, (2005).
[4] S. Hartmann, S. Link. Expressive, yet tractable XML keys, in: EDBT, ACM International Conference Proceeding Series, ACM, **360**, 357-367 (2009).
[5] S. Hartmann, S. Link. Efficient reasoning about a robust XML key fragment, ACM Trans. Database Syst., **34**, (article 10) (2009).
[6] M. Arenas, L. Libkin. A normal form for XML documents, ACM Trans. Database Syst., **29**, 195-232 (2004).
[7] S. Hartmann, S. Link. More functional dependencies for XML, in: AdBIS, Lecture Notes in Computer Science, Springer, Berlin, **2798**, 355-369 (2003).
[8] M. Vincent, J. Liu, C. Liu. Strong functional dependencies and their application to normal forms in XML, ACM Trans. Database Syst., **29**, 445-462 (2004).
[9] S. Hartmann, T. Trinh. Axiomatising functional dependencies for XML with frequencies, in: FoIKS, Lecture Notes in Computer Science, Springer, Berlin, **3861**, 159-178 (2006).
[10] P. Buneman, W. Fan, J. Simon, S.Weinstein. Constraints for semi-structured data and XML, SIGMOD Record, **30**, 47-54 (2001).
[11] W. Fan, J. Siméon. Integrity constraints for XML, J. Comput. Syst. Sci., **66**, 254-291 (2003).
[12] Sven Hartmanna, Sebastian Link. Numerical constraints on XML data, Information and Computation, **208**, 521-544 (2010).
[13] David M. Kroenke. Database Processing: Fundamentals, Design and Implementation, Prentice Hall, (2010).
[14] T. Bray, J. Paoli, and C.M. Sperberg-McQueen. Extensive Markup Language (XML) 1.0. World Wide Web Consortium(W3C), Feb. (1988). http://www.w3.org/TR/REC-xml.
[15] A. Layman, E. Jung, E. Maler, and Henry S. Thompson. XML-Data. W3C Note, January (1998). http://www.w3.org/TR/1998/NOTE-XML-data.
[16] Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn. XML Schema Part 1:Structures, W3C Working Draft, April (2000). http://www.w3.org/TR/xmlschema-1/.
[17] Md. Sumon Shahriar and Jixue Liu. On Defining Keys for XML, IEEE 8th International Conference on Computer and Information Technology Workshops, 86-91.
[18] S. Hartmann, H. Koehler, S. Link, T. Trinh, J.Wang, On the notion of an XML key, in: SDKB, Lecture Notes in Computer Science, Springer, Berlin, **4925**, 103-112 (2008).
[19] Peter Buneman, Susan Davidson, Wenfei Fan, Carmem Hara and Wang-Chiew Tan. Keys for XML, Comput. Networks, **39**, 473-487 (2002).
[20] P. Buneman, S. Davidson, W. Fan, C. Hara, W. Tan. Reasoning about Keys for XML, Inform. Syst., **28**, 1037-1063 (2003).
[21] Flavio Ferrarotti, Sven Hartmann, Sebastian Link, etc. Performance analysis of algorithms to reason about XML keys, Database and Expert Systems Applications, Lecture Notes in Computer Science, **7446**, 101-115 (2012).
[22] Md. Sumon Shahriar and Jixue Liu. On the Performances of Checking XML Key and Functional Dependency Satisfactions, On the Move to Meaningful Internet Systems: OTM 2009, Lecture Notes in Computer Science, **5871**, 1254-1271 (2009).
[23] Gösta Grahne and Jianfei Zhu. Discovering Approximate Keys in XML Data. CIKM, 453-460 (2002).
[24] Jiawei Han, Micheline Kamber and Jian Pei. Data Mining: Concepts and Techniques, 3rd ed[M]. Morgan Kaufmann Publishers. July (2011).

Yijun Liu got her bachelor's degree in engineering from Nanjing University in 2000, her Master's degree in engineering from Nanjing University in 2003. She currently works in School of Computer Engineering of the Jiangsu University of Technology. Her research interests include machine learning, data mining and intelligent information system.

Feiyue Ye received the PhD degree from College of Information Science and Technology, Nanjing University of Aeronautics and Astronautics, in June 2006 and his Masters degree (by research) in engineering from Xi'an University of Architecture and Technology in 1997 and he got his bachelor's degree in engineering from Xian University of Architecture and Technology in 1984. He currently works in the Jiangsu University of Technology. He worked in School of Computer and Information Science, the University of South Australia as a visiting scholar from 07 Mar 2009 to 08 Sept. 2009. His current research interests include data mining and intelligent information system. Feiyue Ye has published more than 30 papers in journals and conference in databases and data mining (DKE, Control and Decision, LNCS, etc).

Jixue Liu got his bachelor's degree in engineering from Xian University of Architecture and Technology in 1982, his Master's degree (by research) in engineering from Beijing University of Science and Technology in 1987, and his PhD in computer science from the University of South Australia in 2001. He currently works in the University of South Australia. His research interests include view maintenance in data warehouses, the transformation of data, constraints, and queries between XML and relational data, XML data and integrity constraint integration and transformation, data privacy, trust management on the internet, and integrity constraint discovery from data. Jixue Liu has published in world's top journals in Databases (TODS, JCSS, TKDE, Acta Informatica, etc).

Sheng He received the Ph.D. degree from Jiangnan University, China. Currently he is employed as associate professor at School of Computer Engineering of Jiangsu University of Technology, China. He is mainly engaged in teaching and research in the fields of data mining and bioinformatics. He has produced a new network visualization algorithm named "Fast Grid Layout" which has been proved to be an efficient network layout algorithm in the field of bioinformatics.