WUJNS
Wuhan University Journal of Natural Sciences

Vol.11 No.1 2006 127-132

Article ID:1007-1202(2006)01-0127-06

An Optimized Approach for Extracting Approximate Functional Dependencies in XML Documents

SHI Lei, YANG Xiao-chun†, YU Ge, WANG Bin, ZHOU Hua-hui

School of Information Science and Engineering, Northeastern University, Shenyang 110004, Liaoning, China

Abstract: In this paper, the definition of approximate XFDs based on value equality is proposed. Two metrics, support and strength, are presented for measuring the degree of approximate XFD. A basic algorithm is designed for extracting minimal set of approximate XFDs, and then two optimized strategies are proposed to improve the performance. Finally, the experimental results show that the optimized algorithms are correct and effective.

Key words: XML; functional dependencies; strength of functional dependencies

CLC number: TP 311.13

Received date: 2005-04-10

Foundation item: Supported by the National Natural Science Foundation of China (60173051), Teaching and Research Award Program for Outstanding Young Teachers in Higher Education Institution of the Ministry of Education, the National Research Foundation for the Doctoral Program of Higher Education of China(20030145029), and the Natural Science Foundation for Doctoral Career Award of Liaoning Province(20041016).

Biography: SHI Lei(1980-), male, Master candidate, research direction: XML access control, data mining. E-mail:neushi@126.com

† To whom correspondence should be addressed. E-mail: yangxc@mail.neu.edu.cn

0 Introduction

Functional dependency (FD) is one of most important integrity constraints in databases. FDs, expacially approximate FDs, are widely used on knowledge discovering, privacy protection, data inference[1,2], etc. For instance, pathologists wish to acquire the relation between various dietetic habits and diseases, salesmen want to know the relation between different age-grades and consuming patterns, etc. Those prevailing non-precise relationships between data are called approximate FDs[3]. XML (eXtensible Markup Language) becomes the standard for data description and data exchange, recently. However due to semi-structured nature, XML functional dependencies(XFDs) differ a lot from those in traditional relational databases and extracting approximate XFDs is quite different from that in relational database.

Some work[4-7] has been done on XFDs and most of them focuses on precise XFDs. Ref. [4] defines XFD on Tree Tuples. In Ref. [4], an XFD is defined between either internal nodes or leaf nodes of an XML tree, whereas in Ref. [5], an XFD only holds between leaf nodes. In addition, in terms of judging the equality of two nodes, Ref. [4] considers two nodes equal under the condition that the identities of two nodes are the same.

However, in the real world, contents of the document are more meaningful than the identities of nodes in the document. For instance, an element part has subelements shape and size. Given parts n_1 and n_2, it is more appropriate to compare not only the two elements themselves but also their sub-

elements shape and size. We will adopt such way (called value equality) to evaluate the equality of two nodes, so that, approximate XFDs we define later, will reflect the patterns of the XML document more reasonable. In addition, Ref. [4-6] do not mention how to measure approximate XFDs.

Current algorithms mainly focus on extracting approximate FDs in relational databases[3,8-10]. Buneman, et al use an Apriori-alike algorithm for mining a reduced set of approximate keys from an XML document[11].

In this paper, we mainly focus on the issues of defining and extracting approximate XFDs in XML documents. Our contributions are listed as follows.

1) We propose a new definition of XFDs based on value equality. Two criteria namely strength and support for measuring approximate XFDs are given as well.

2) We introduce the concept of minimal set of approximate XFDs. The basic algorithm and two optimized strategies are proposed to extract such a set.

3) We use experiments to show the proposed algorithm is feasible and effective.

1 Definition of Approximate XFDs

An XML document can be viewed as a tree called XML tree. For further understanding of XML tree, path and related concepts, one can refer to Ref. [4,12]. Figure 1 is an example of XML tree containing information of patients in several hospitals, where "/Hospitals/Hospital/hName/S" is a path. From the document, we can easily find an XFD, if patients live in the same "ward" then they suffer the same "disease".

Our XFD definition is based upon value equality ($=_v$)[10] to express the dependency relationships between a node and its sub-trees. Comparing our definition with Ref. [4], the major differences are ① We use value equality to evaluate the relationship between two nodes; ② We do not need to transform XML documents into tree tuples.

XFDs are concerned with related nodes, therefore, we first define related-nodes set.

Definition 1 Related-nodes set: Given paths p and q, n is a node in an XML document T and $n \in [[q]]^2$, then we call a set of nodes S ($S \subseteq [[p]]$) are related with n in T as n's related nodes set about path p, denoted as $n(p/q)$, $n(p/q) = \{v \mid v \in [[p]], a$ is the closest ancestor of both n and v and $a \in [[\ p \cap q\ ]]\ \}$. Where $[[q]]$ denotes a set of nodes that path q can reach starting from the root of an XML document[10], $p \cap q$ denotes the longest public path of p and q.

For instance, in Fig. 1, assume path p="/Hospitals /Hospital/patient/ward" and path q="/Hospitals/Hospital /patient/disease", $p \cap q$="/Hospitals/Hospital/patient", node $v_{10} \in [[q]]$, then v_{10}'s related nodes set about path p is $v_{10}(p/q) = \{v_8\}$, where, v_{10} and v_8 belong to the same patient.'

With the Definition 1, we are ready to define XFDs based on value equality.

Definition 2 XFDs based on value equality: An XFD F is an expression of the form F: $X \rightarrow q$ where $X = \{p_1, p_2, \cdots, p_n\}$ are set of paths and q is a single path. An XML document T satisfies the XFD F, if for any n_1, $n_2 \in [[q]]$, if $n_1 \neq_v n_2$, then there exists $i \in [1, n]$, such that $n_1(p_i/q) \neq_v n_2(p_i/q)$. Paths p_1, p_2, $\cdots$, p_n are called determining paths and q is called depending path.

Definition 3 XFD Tuples: We assume an approximate XFD F: $X \overset{\approx}{\longrightarrow} q$ ($X = \{p_1, p_2, \cdots, p_n\}$ is a set of paths and q is a single path). The XFD Tuples R of F is

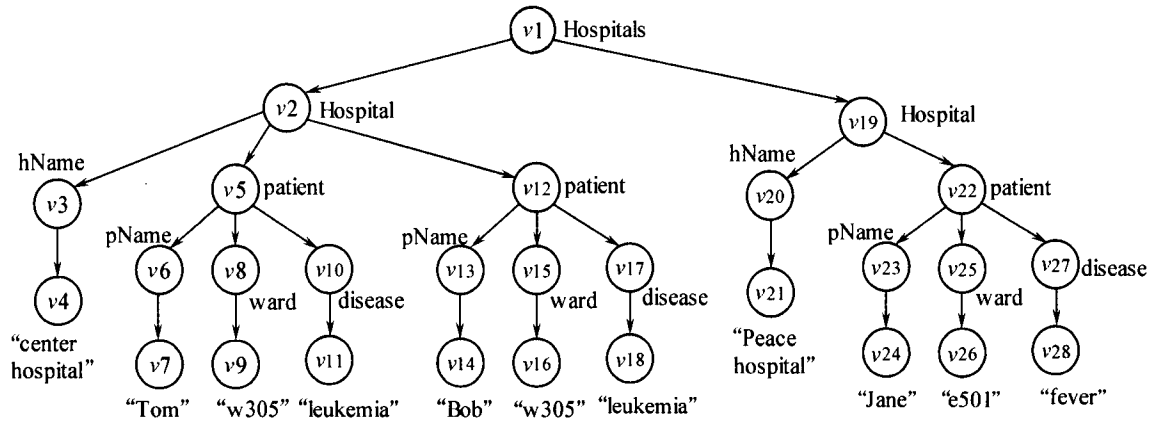


Fig. 1 An example of XML tree

a table with $(n+1)$ columns, and each column name of this table is "p_1", "p_2", ⋯ , "p_n", "q", respectively. In addition, it satisfies: ① for each $v \in [[q]]$, there exists a tuple $r \in R$ such that $r["q"] = \{v\}$; ② for each $r \in R$, suppose $r["q"] = \{v\}$, for each p_i, $i \in [1, n]$, $r["p_i"] = v(p_i/q)$.

Obviously, every column value of any tuple in R is a set of nodes, and the total amount of tuples equals to the number of $[[q]]$ in the XML document, i.e. $|[[q]]|$.

XFD tuples R satisfies the precise XFD $X \rightarrow q$, if for every two tuples $r_1, r_2 \in R$, $r_1["q"] \neq_v r_2["q"]$, there exists $i \in [1, n]$, $r_1["p_i"] \neq_v r_2["p_i"]$. R satisfies a precise XFD $X \rightarrow q$, denoted as $R < (X \rightarrow q)$.

After mapping an XML document into XFD Tuples, we introduce support and strength as two criteria for measuring approximate XFDs based on XFD Tuples.

Definition 4 Given an XML document T: one of its approximate XFDs $F: X \overset{\approx}{\longrightarrow} q$ (X is a set of paths and q is a path) and R is the XFD tuples of F, then the strength of F is the percentage of maximum XFD Tuples satisfying the precise XFD.

$$\text{strength} = \frac{\max\{|S| \mid S \subseteq R, S < (X \rightarrow q)\}}{|R|} \quad (1)$$

The support of $X \overset{\approx}{\longrightarrow} q$ is the total amount of tuples in R.

$$\text{support}(F, T) = |R| \quad (2)$$

Intuitively, $0 \leqslant \text{strength}(F, T) \leqslant 1$, if and only if $\text{strength}(F, T) = 1$, can we say that F is a precise XFD, otherwise F is an approximate XFD. From our point of view, only those whose support and strength are great enough are interesting and worthy of extracting, therefore, in this paper, we focus on extracting approximate XFDs, such that, for each of those, the support and strength are not smaller than two given thresholds minimal support and minimal strength, respectively.

2 Extracting Approximate XFDs

In this section, we first present a minimal set of approximate XFDs, and then introduce the algorithm of extracting it from an XML document. Finally, several optimization policies are proposed and applied to this algorithm.

2.1 Minimal Set of Approximate XFDs

We consider an approximate XFD candidate a qualified one only if its support and strength are no smaller than given minimal support and minimal strength, respectively. In order to extract the minimal set of qualified approximate XFDs, we give two definitions here.

Definition 5 Non-trivial approximate XFD: Given an approximate XFD $X \overset{\approx}{\longrightarrow} q$, if $q \notin X$, then, $X \overset{\approx}{\longrightarrow} q$ is called a non-trivial approximate XFD.

Definition 6 Minimal approximate XFD: Given $X \overset{\approx}{\longrightarrow} q$ is an qualified approximate XFD, if there does not exist any $Y \subset X$, where $Y \overset{\approx}{\longrightarrow} q$ is also a qualified approximate XFD, then we say that $X \overset{\approx}{\longrightarrow} q$ is a minimal approximate XFD.

We denote a set of approximate XFDs in an XML document T as minXFDs(T), which contains all minimal non-trivial XFDs whose strength and support satisfy two minimal constraints respectively. Obviously, minXFDs(T) is a minimal set of approximate XFDs, therefore we just need to extract all minimal non-trivial approximate XFDs from the XML document.

2.2 Candidate-Paths Set

Since the determining paths and depending path of a candidate XFD are respectively a set of paths and one single path, therefore, prior to extracting approximate XFDs, we should acquire all paths.

We denote all paths in an XML document T as Paths(T)[4] which can be generated from searching the whole document. However, some paths are semantically overlapped. For instance, "/Hospitals/Hospital/hName" and "/Hospitals/Hospital/hName/S" actually both refer to the name of a hospital. We phase out paths ending with "S", and the rest are called candidate paths, denoted as EPaths(T). Obviously, EPaths(T) is a subset of Path(T), therefore can significantly reduce the searching space.

We design an algorithm called extractEPaths for extracting all candidate paths from an XML document. Since extracting candidate paths is out of the scope of this paper, we will not discuss it here.

2.3 Strategies for Extracting Approximate XFDs

In this section, we will first introduce the searching space for extracting approximate XFDs. And then we will give a method of extracting approximate XFDs in the searching space, two optimized searching strategy are also given.

2.3.1 Searching space

Suppose an XML document T and its candidate path

set EPaths(T), we will search XFDs in a space made up of EPaths(T). The searching space S is:

$$S = \{X \xrightarrow{\approx} q \mid X \subseteq \text{Epaths}(T), X \neq \varnothing \text{ and } q \in \text{EPaths}(T)\}$$

We organize the searching space in a lattice. The 0-th level of the lattice is empty, and the ith level contains in total C_n^i (n is the total number of candidate paths) elements each of which is a composition of i different candidate paths. Every element X in the lattice, can be determining paths of a candidate for approximate XFDs, and we denote a set of paths that can possibly be X's depending path by Rhs(X), where Rhs(X)=EPaths(T). In this way, the more candidate paths a element X contains, the higher level X will stay in the lattice.

2.3.2 Basic searching strategy

Searching space S is now orderly organized into a lattice, we can extract approximate XFDs on this lattice.

Since the 0-th level is empty, searching process starts from the 1st level to the last level. At the i-th level L_i, for every element X in L_i and for any $q \in$ Rhs(X), a candidate $F(X \xrightarrow{\approx} q)$ is formed and its strength and support are computed(one can follow definition 4 to calculate strength and support, in addition, we will offer some optimizations in Section 2.4). If the calculated strength and support are no smaller than minimal support and minimal strength, respectively, furthermore, if F is minimal and non-trivial, then, F will be put out.

2.3.3 Optimized searching strategy 1

To speed up the searching, we apply some pruning rules into the basic searching algorithm, so that, all trivial candidates and all non-minimal candidates will be pruned away from searching space. The optimized algorithm is also based on lattice. However, the searching process goes interactively with dynamic construction of the lattice, in detail:

1) Construct the ith level of this lattice. If $i=1$ then $L_i = \{\{p\} \mid p \in \text{EPaths}(T)\}$; if $i > 1$, then L_i is constructed based on the previous level L_{i-1}: $L_i = \{(X \cup Y) \mid X, Y \in L_{i-1} \text{ and } |X-Y|=1\}$;

2) At level L_i, for every element X in L_i, generate Rhs(X). If $i=1$, Rhs($\{p\}$) $= \{q \mid q \in \text{EPaths}(T) \text{ and } q \neq p\}$; else Rhs($X$) $= \Phi_{p \in X}$ Rhs($X-p$). If every element X at this level satisfies Rhs(X) $= \varnothing$, then it suggests no candidate left, end the algorithm;

3) For every element X of L_i and $\forall q \in$ Rhs(X), check whether the strength and support of $X \xrightarrow{\approx} q$ satisfy the two constraints. If satisfy, then put $X \xrightarrow{\approx} q$ out as a qualified approximate XFD, and delete q from Rhs(X). If Rhs(X)=$\varnothing$, deleted X from L_i.

2.3.4 Optimized searching strategy 2

For a candidate $X \xrightarrow{\approx} q$, its support equals to $|[[q]]|$. This inspires us to prune the searching space furthermore.

Based on strategy 1, when generating the Rhs of 1st level' elements X, set Rhs(X)=$\{p \mid p \in \text{Epaths}(T)$ and $|[[q]]| \geqslant \text{minSupport}\}$. Then every candidate generated is guaranteed to satisfy the minimal support constraint.

2.4 Optimized Computation of Strength

To avoid repeated comparisons of different nodes, prior to searching, we mark any two nodes, which are value equal with the same integer value. In this way, for later value equal comparison, we only need compare the integer value of two nodes instead of their sub-trees.

3 Algorithm

Algorithm extractAXFD adopts both the optimized Strategy 1 and optimized Strategy 2. |EPaths| denotes the number of candidate paths, and $|L_i|$ denotes the number of elements in ith level.

Agorithm: extractAXFD

Input: XML document T, minSupport, minStrength

Output: approximate XML Functional Dependencies

```
1. EPaths=extractEPaths(T);
2. L1 = {{p} | p∈EPaths};
3. for each {p} in L1
4.       Rhs(p) = {q | q∈EPaths,
            |[[q]]|≥minSupport and q≠ p};
5. end for
6. AXFDs=checkAXFD(L1);
7. for i=2 to |EPaths|
8.       Li=genNextLevel(Li-1);
9.       if |Li| =0 then break;
10.      AXFDs += checkAXFD(Li);
11. end for
12. return AXFDs;
```

Procedure checkAXFD searches all candidates at ith level. In detail, for every element X of ith level and every $p \in$ Rhs(X), it first check whether the strength of $X \xrightarrow{\approx} q$ is no smaller than minimal strength, if so, X

$\overset{\approx}{\longrightarrow} q$ is put out as a qualified approximate XFD, meanwhile p is removed from Rhs(X).

Procedure genNextLevel constructs the next level L_i with the searching results of previous level L_{i-1}.

4 Experimental Results

We implement the algorithms using JAVA programming language, and apply it to real-world hospital XML documents that have the same DTD with Fig. 1.

The experimental environment is as follows: Pentium4 2.4 GHz CPU, 512 MB memory, 80 GB hard disk. The operating system is Windows XP and we use dom4j for parsing XML document.

We apply the algorithm into a hospital XML document that contains 1 000 patients' information in 6 hospitals. Set minimal support and minimal strength 10 and 0.98 respectively, we extracted the following 8 approximate XFDs: ("Hs" is "Hospitals" for short, "H" is "Hospital" for short and "P" is "patient" for short)

/Hs/H/P $\overset{\approx}{\longrightarrow}$ /Hs/H/P/pName

/Hs/H/P $\overset{\approx}{\longrightarrow}$ /Hs/H/P/ward

/Hs/H/P $\overset{\approx}{\longrightarrow}$ /Hs/H/P/disease

/Hs/H, /Hs/H/P/ward $\overset{\approx}{\longrightarrow}$ /Hs/H/P/disease

/Hs/H/hName, /Hs/H/P/ward $\overset{\approx}{\longrightarrow}$ /Hs/H/P/disease

/Hs/H,/Hs/H/P/ward,/Hs/H/P/pName $\overset{\approx}{\longrightarrow}$ /Hs/H/P

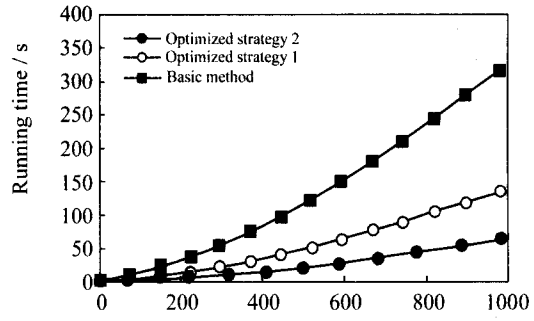
/Hs/H/P/disease,/Hs/H/P/ward, Hs/H/P/pName $\overset{\approx}{\longrightarrow}$ /Hs/H/P

/Hs/H/hName,/Hs/H/P/pName,/Hs/H/P/ward $\overset{\approx}{\longrightarrow}$ /Hs/H/P

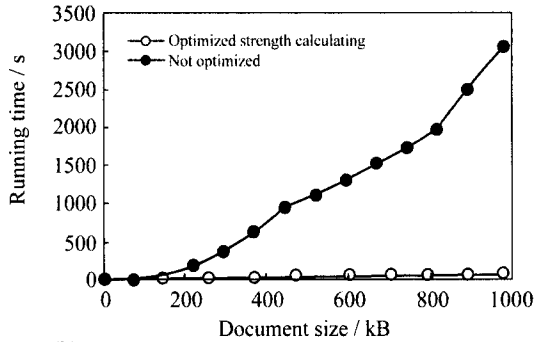
Figure 2(a) compares execution times of three algorithms with different pruning rules: basic algorithm, algorithm with optimized Strategy 1 and algorithm with optimized Strategy 2. As we can see, the one with optimized Strategy 2 is nearly 5 times faster than that of basic, and 1 time faster than that with optimized Strategy 1.

Figure 2(b) compares the effieiency of the algorithms with and without strength optimization. The algorithm adopting strength optimization is roughly 30 times faster than that without strength optimization.

We set minimal support to 10 and test the algorithm on a hospital XML document containing 1 000 patients' information of 6 hospitals. Figure 3 shows the relationship between minimal strength val and number of extracted approximate XFDs. The number of approximate XFDs decreased when minimal strength increases from 0.5 to 1. That is because, when minimal strength decreases, it loosens the restriction for approximate XFDs, therefore, extracts more approximate XFDs.



(a) The effieiency of the optimized strategies



(b) The effieiency of the optimized strength calculating

Fig. 2 The efficiency of algorithms

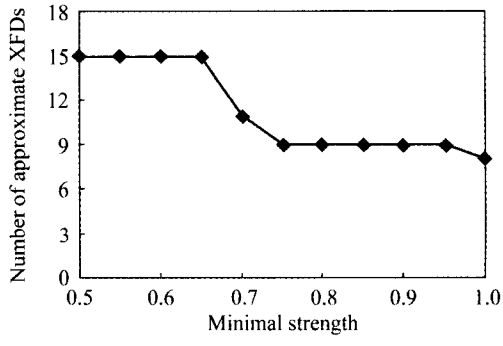


Fig. 3 The relationship between minimal support and number of approximate FDs extracted

5 Conclusion

We defined approximate XFDs based on value equality, and presented two criteria namely strength and support for measuring approximate XFDs. We proposed the concept of minimal set of approximate XFDs, a method

and two optimized strategies are proposed for extracting such a set from an XML document. The experimental results show that the proposed extracting approaches are correct and effective.

References

[1] Yang X, Li C. Secure XML Publishing without Information Leakage in the Presence of Data Inference. *Proceedings of 30th International Conference on Very Large Data Bases*. Toronto, Canada, Sept. 2004. 96-107.

[2] Yang X, Li C, Yu G. XGuard: A System for Publishing XML Documents without Information Leakage in the Presence of Data Inference. *Proceedings of International Conference on Data Engineering*. Tokyo, Japan, April 2005. 1124-1125.

[3] Kivinen J, Mannila H. Approximate Inference of Functional Dependencies from Relations. *Theor Comput Sci*, 1995, **149** (1): 129-149.

[4] Arenas M, Libkin L. A Normal Form for XML Documents. ACM Trans. *Database Syst*, 2004, **29**: 195-232.

[5] Liu J, Vincent M W, Liu C. Functional Dependencies, from Relational to XML. *Ershov Memorial Conference*. Novosibirsk, Russia, July, 2003. 531-538.

[6] Lee M L, Ling T W, Low W L. Designing Functional Dependencies for XML. *Proceedings of 8th International Conference on Extending Database Technology*. Prague, Czech Repubic, March 2002. 124-141.

[7] Yang X, Wang G. Mapping Referential Integrity Constraints from Relational Databases to XML. *Proceedings of the 2nd International Conference on Advances in Web-Age Information Management*. German: Springer-Verlag Press, 2001. 329-340.

[8] Huhtala Y, Kärkkäinen J, Porkka P, *et al*. TANE: An Efficient Algorithm for Discovering Functional and Approximate Dependencies. *Comput J*, 1999, **42**: 100-111.

[9] Mannila H, Toivonen H. Levelwise Search and Borders of Theories in Knowledge Discovery. *Data Min Knowl Discov*, 1997, **1**(3): 241-258.

[10] Ilyas I F, Markl V, Haas P J, *et al*. CORDS: Automatic Discovery of Correlations and Soft Functional Dependencies. *Proceedings of Proceeding at the ACM SIGMOD International Conference on Management of Data*. Paris, France, June 2004. 647-658.

[11] Grahne G, Zhu J. Discovering Approximate Keys in XML Data. *Proceedings of International Conference on Information and Knowledge Management*. Virginia, USA, Nov. 2002. 453-460.

[12] Buneman P, Davidson S, Fan W, *et al*. Reasoning about Keys for XML. International Workshop on Database Programming Languages. Lecture Notes in Computer Science 2397. Frascati, Italy, Sept. 2002. 133-148.

□