Efficient Associating Mining Approaches for Compressing Incrementally Updatable Native XML Databases

Chin-Feng Lee, Chia-Hsing Tsai

***Abstract*—XML-based applications widely apply to data exchange in EC and digital archives. However, the study of compressing Native XML databases has been surprisingly neglected, especially for the huge amount of data and the rapidly updatable database. These two factors give rise to our interest, and motivate us to develop an approach to efficiently compress native XML databases and dynamically maintaining a set of compression rules when insertion, deletion, or modification functions in databases. This approach is to utilize data mining technology and association rules, to mine all frequent patterns. We proposed a frequent tag and character data pattern split tree (FTCP-split tree) to fast generate the set of association patterns. Then we convert these frequent patterns into compression rules, which would be used to compress native XML databases. The question which we must consider next is how to dynamically maintain XML database compression when XML documents are inserted, deleted, or modified. We propose a compression approach with dynamic maintenance on native XML databases to deal with it. The results of preliminary experiment indicate that the compression rate of our research is higher than the common compression software as ZIP and RAR.**

***Index Terms*—Compression, Data Mining, Incremental data Mining, Native XML Database.**

I. Introduction

Due to the extensive application of XML technology in different fields, such as digital archive, geographic information system, e-commerce, and health industry, an enormous number of XML documents have been created. For the last few years, more and more manufacturers began to develop and design native XML databases that enable direct storage and management of XML document. We confronted with two difficulties. The first is the data storage capacity. The second is the data variation. In this study, we improve the problems mentioned above. Our attempts are:

Manuscript received April 6, 2006.

Chin-Feng Lee is with the Department of Information Management, Chaoyang University of Technology, No. 168, Jifong E. Rd., Wufong Township, Taichung Country 41349, Taiwan (R.O.C.)

E-mail: lcf@cyut.edu.tw

Chia-Hsing Tsai is with the Department of Information Management, Chaoyang University of Technology, No. 168, Jifong E. Rd., Wufong Township, Taichung Country 41349, Taiwan (R.O.C.)

E-mail: s9314641@cyut.edu.tw

A. Raise the efficiency of compression rate

Our main purpose is to develop and design a more efficient compression technique for native XML databases. We apply the association rule mining in this research to generate a set of compression rules. In order to reach this goal we adopt the structure of FTCP-split tree to preserve our frequent patterns. The features of this structure are fast in tree construction, which is the core phase to generate a set of compression rules. This method leads the process to efficiency.

B. Dynamically maintain the compressed databases

When insertion, deletion, or modification functions in the database, the compression rules need to be changed. For this reason, we develop an approach to dynamically maintain the compressed databases. In our research, we use two thresholds proposed by Hong *et al.* to obtain a set of pre-frequent tag and character data for maintaining the set of compression rules [[6]]. This approach will reduce compression time since we do not regenerate a set of compression rules for dynamic XML databases.

In mining process, the traditional FP-growth was only used for mining transaction items which appear one time in a transaction record. In this research, we not only improve the above drawback but to deal with the repeatable transaction items in transaction database and native XML database. Therefore, Then exploit the FTCP-split tree proposed by us to compress the data in a tree structure and enhance the mining efficiency. In this process, we can greatly reduce the time cost of rescanning database and reduce the candidate itemset.

II. LITERATURE REVIEW

A. Compress Data with Data Mining Technique

XML is widely and frequently used for data exchange, which makes the data volume growing over time. Consequently, database compression is the key to solve this problem. In recent years, some scholars apply the data mining techniques for database compression such as: Apriori algorithm & ID3 algorithm (Goh et al., 1998), Decision Tree (Babu et al., 2001), CIT algorithm (Lee et al., 2001), and FUP&FUP2 algorithm (Lee & Tang, 2004) are notable examples [[1]][[2]][[3]][[4]][[8]][[9]].

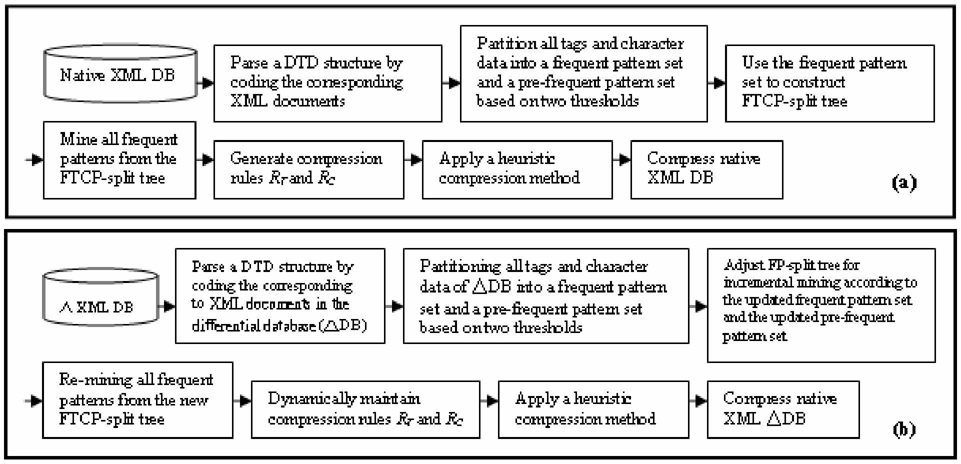

Native XML DB
Parse a DTD structure by coding the corresponding XML documents
Partition all tags and character data into a frequent pattern set and a pre-frequent pattern set based on two thresholds
Use the frequent pattern set to construct FTCP-split tree
Mine all frequent patterns from the FTCP-split tree
Generate compression rules R_T and R_C
Apply a heuristic compression method
Compress native XML DB
(a)
△ XML DB
Parse a DTD structure by coding the corresponding to XML document's in the differential database (△DB)
Partitioning all tags and character data of △DB into a frequent pattern set and a pre-frequent pattern set based on two thresholds
Adjust FP-split tree for incremental mining according to the updated frequent pattern set and the updated pre-frequent pattern set
Re-mining all frequent patterns from the new FTCP-split tree
Dynamically maintain compression rules R_T and R_C
Apply a heuristic compression method
Compress native XML △DB
(b)

Figure 1: The flow chart of the proposed compression approaches XML DB.
(a) PartⅠ: Static compression (b) PartⅡ: Dynamic compression

The first scholar to give much attention to dynamically maintain compressed databases with Data Mining Technique was Lee & Tang. In our research, we develop this idea a little further. Lee & Tang adopted FUP& FUP2 to compress databases [[2]][[3]]. Lee & Tang used an Apriori-like approach so that their approach will incurred the bottleneck in generating large amount of candidate itemset [[8]].

B. Dynamic Mining

Rescanning a changed database will consume much time and cost, to reduce the cost of database rescan, many incremental data mining are proposed to maintain a set of frequent itemsets. Those approaches can be divided into two categories – Apriori-like and Non Apriori-like approaches. Apriori-like approaches such as FUP or FUP2 need to repeat scanning the database several times [[2]][[3]]; while Non Apriori-like approaches utilize information structures, such as AFPIM adjust the FP-tree to mine frequent itemsets without repeating scanning the database [[5]][[7]]. However, the previous approaches are still suffered from the problem of [[2]][[3]].

C. FP-split tree

FP-split algorithm (Lee & Shen, 2005) constructs FP-split tree by adopting the divide-and-conquer strategy by means of intersection and difference of itemsets [[11]]. This approach could rapidly generate a set of associated patterns since the structure of FP-split tree formed by scanning the database only once. The tree construction time and I/O cost were much less than those of FP-tree (Han *et al.*, 2000), because the FP-growth approach based on FP-tree requires scanning the database twice [[5]]. FP-split approach was only applicable to static data mining. However, when a database are inserted, deleted, or modified, the updatable database should be rescanned to constructing a new FP-split tree [[11]].

D. Pre-large itemsets

Hong *et al.* proposed a novel incrementally mining algorithm using two thresholds [[6]]. This research solves the

problem of case 3 in FUP algorithm [[2]]. That is, the case 3 problem is the situation when a database is changed without saving the enough information (non-frequent itemsets). The solution is to rescan the original database frequently. However, this will reduce the database performance. The Pre-large itemsets play a buffer role to avoid rescanning the original database even if data are inserted, deleted, or modified. The results of Hong's experiment proved the maintenance time could be reduced.

III. THE PROPOSED COMPRESSION METHOD

Our research composes of the static compression phase and dynamic compression phase. It is shown in Figure 1.

A. Phase Ⅰ Static compression

Step 1 Parse a set of XML documents defined by a given DTD

The way we parse every XML documents by a given DTD is to encode these structures as follows. In a XML database for a specific DTD structure (say *D* in Figure 2), an approach is proposed to encoding all tags and character data of XML documents which is defined by *D*.

```
<!Element A (B* , C*)>
<!Element B (#PCDATA)>
<!Element C (#PCDATA)>
```

Figure 2: An example of DTD Structure (called *D*)

Approach to encoding all XML document X_k:

The approach has two kinds of cases：

Case (1) For the k-th XML document with a n-level hierarchy, the node at the first level is the root and can be encoded as k.

Case (2) From level 2 to level n-1, we can encode each tag as follows. Let the tag which we want to encode be q. Its parent tag is p. If the code of p is x then the code of q is x.y . In addition, the leaves at the n_{th} level can inherit the codes from the $(n-1)_{th}$ level.

1<A>	X_1	2<A>	X_2
1.1 <B>D</B>		2.1 <B>D</B>	
1.2 <B>D</B>		2.2 <C>E</C>	
1.3 <B>F</B>		</A>	

</A> (a)	(b)
3<A> X_3	4<A> X_4
3.1 <B>D</B>	4.1 <C>E</C>
3.2 <B>E</B>	</A>
3.3 <B>F</B>	
3.4 <C>E</C> (c)	(d)
</A>	

Figure 3: Four coded XML documents （a）X_1 （b）X_2 （c）X_3 （d）X_4

Example 1: Take Figure 3(a~d) as an example to show the result of the encoding procedure. With regard to some Collection C_i and its structure of DTD D_i, we use DFS (Depth-First-Search) to scan all XML documents defined by D_i and then encode all of them.

Step2 Extract a set of frequent element and a set of pre-frequent element of length one.

We adopt a concept of two support threshold proposed by Hong *et al.*[[6]]. The purpose is to find frequent element set more efficiently.

DEFINITION The equivalence class of elements

Given an element set Ω=<ε_1, ε_2,…, ε_k>, the term "the equivalence of elements" for Ω can be defined as EC_Ω= { $(d_1, d_2, ..., d_j) | d_k$, which is presented as code correspond to ε_i, and the code is the position where ε_i locates. If the size of Ω is *k*, and then we will call EC_Ω as an equivalence class with length of *k*.

Example 2: Take the two elements "A" and "F" from Figure 3(a~d) as an example. Ω={A, F} means that tags "A" and character data "F" occur in one XML document at the same time. The equivalence class of these two elements can be represented by $EC_{<A, F>}$= {(1, 1.3), (3, 3.3)}. It means an equivalence class with length of two.

DEFINITION An equivalence class with recurrent elements

We call an element ε is recurrent if it occurs more than one time in one document.

$d_1.d_2.\cdots.d_{i-1}.[d_{i,1}, d_{i,2}, \dots, d_{i,k}]$ is a general expression for the position code of element ε. It means ε appear in the l locations where are $d_1.d_2.\cdots.d_{i-1}.d_{i,1}$, $d_1.d_2.\cdots.d_{i-1}.d_{i,2}$, ... ,$d_1.d_2.\cdots.d_{i-1}.d_{i,l}$, respectively.

Example 3: Take the tag "B" from Figure 3(a) as an example. Tag "B" is recurrent because it appears in the first XML document three times and is coded as "1.1", "1.2" and "1.3", respectively. The position code for tag "B" can be presented as 1.[1, 2, 3].

Example 4: Take the two tags "A" and "B" from Figure 3(a~d) for example. Let Ω= {A, B}. The equivalence class of these two tags are shown as $EC_{<A, B>}$= {(1, 1.[1, 2, 3]), (2, 2.1), (3, 3.[1, 2, 3])}. Besides, the equivalence class count ($|EC_\Omega|$) for $EC_{<A, B>}$ is 3.

A procedure MSL is developed to return a list L, which indicates the element set Ω actually appear in what documents.

```
Procedure MSL（$EC_\Omega$）
{ $L = \varnothing$ ;
  While（$\forall\varepsilon$ in $EC_\Omega$ and $\varepsilon$ is NOT NULL）
    $L \leftarrow f(\varepsilon) \cup L$ ; //Given an equivalence class $EC_\Omega$ for
        element set $\Omega$. For each element x in $\Omega$, function
        f can output x's most significant locators.//
  Return（ L ）;
}
```

Example 5: Take the character data "F" from Figure 3(a~d) for example. Given $EC_{<F>}$= {1.3 , 3.3}. Therefore, the *MSL($EC_{<F>}$)*= {1 , 3}.

DEFINITION Count

Given an element ε_i, c_{ik} is the occurrence times in the k-th document. So, **C**= $\sum_{k=1}^{n} c_{ik}$ is the total count for element ε_i in database XDB.

Example 6: Take the tag "B" from Figure 3(a~d) for example. Given $EC_{<B>}$= {1.[1, 2, 3] , 2.1 , 3.[1, 2, 3]}. Therefore, the count of $EC_{<B>}$ returns 3 in the first document, 1 in the second document, and 3 in the third document. The count of $EC_{<B>}$=3+1+3=7.

DEFINITION Support

The support of Ω can be defined as $\sum_{k=1}^{n} \min(c_{ik}, c_{jk})$ which states the occurrence times for the set Ω of elements.

Example 7: The equivalence class for Ω= {A, E} is $EC_{<A, E>}$= {(2 , 2.2) , (3 , 3.[2, 4]) , (4 , 4.1)}. Therefore, the support of $EC_{<A, E>}$ =3.

DEFINITION F_k (Frequent element set)

F_k means the set of frequent element of length k from XDB. So, F_k= {(ε_1, ε_2,⋯, ε_k)}. That is, for given a set of *k*- element Ω, Their support is greater than S_u.

DEFINITION PF_k (Pre-Frequent element set)

A Pre-Frequent element set is not really frequent, but is promising to be frequent in the future. Therefore, for given a set of *k*- element Ω, and Ω is said to be pre-frequent, then their support is less than or equal to S_u and greater than or equal to S_l.

According to the above definitions, we can extract a set of frequent element of length one. In the following step, we will store all the F_1 into a head table by the order of their support. In addition, we also store the set of PF_k for dynamic compression rules maintenance.

Step3. Construct FTCP-split tree

An FP-split algorithm is developed by Lee *et al.* which improves the FP growth algorithm in tree construction time [[11]]. FP-split algorithm is based on the FP-split tree, which can compress the database by representing frequent tags and character data into the FP-split tree, but retain the set association information, and then divide such a compressed database into a set of conditional databases (a special kind of projected database), each associated with one frequent item, and mine each such database separately.

In our research, we adapt a tree called FTCP-split tree

(Frequent Tags and Character data Pattern-split tree) for extract frequent association patterns which future can be used to be compressed together.

The nodes in the proposed FTCP-split tree have several fields. The Content field records the content name of tag or character data Ω. The field, Count, records the real amount of which in each documents. The field, Child_link, is a pointer which links to its child nodes. The field, Split_link, is also a pointer linking a split node which has the same Content as Ω. The field, List, stores EC_{Ω}.

We build a FTCP-split tree according to the F_1 obtained from Step 2. We preserve all frequent sets on this tree. We utilize a head table as an index of each node needed in mining period. In head table, it includes three fields, Item, Link, and Bit. The field, Item, records the content name of tags or character data sorted by Support. The field, Link, records a pointer linked each node on this tree. The field, Bit, records the node discriminated between tag and character data (1 or 0). If this bit equal to 1, then it represents a tag. If this bit equal to 0, then it represents a character data.

Approach to construct tree as follows:

First, we generate a root node. This node is a dummy node. If p is a root node and then we execute Case I; else, if p isn't a root node and then we execute Case Ⅱ.

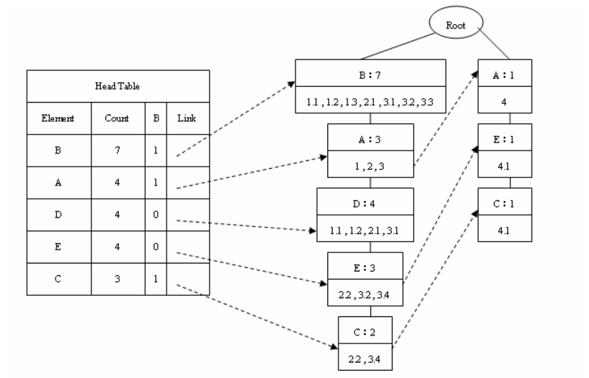
```
CaseI.   IF  p's child node=NULL
         THEN  p.child_link ← n ;
         ELSE Compare (p.child_link, n) ;
CaseII.  IF  p's child node=NULL
         THEN  Compare (p, n) ;
         ELSE Compare (p.child_link, n) ;

Compare(x,y)
CaseI.   IF  MSL(y.List) ⊂ MSL(x.List)
         THEN  x.child_link ← y ;
CaseII.  IF  MSL(x.List) ⋂ MSL(y.List)= ∅
         THEN  IF p is a root
                 THEN p.child_link ← n ;
                 ELSE Compare (x.parent_link, y) ;
CaseIII. IF  MSL(x.List)  ⋂  MSL(y.List)  ≠  ∅  and
         MSL(x.List) ≠ MSL(y.List)
         THEN  Split y into two nodes, n_1, and n_2 ;
                 α ← MSL(x.List) ⋂ MSL(y.List) ;
                 β ← MSL(x.List) – MSL(y.List) ;
            ∀$\varepsilon_i$ ∈ y.List
                 IF  MSL( $\varepsilon_i$ ) ⊆ α
                 THEN Set  $\varepsilon_i$ → n_1.List
                 IF  MSL( $\varepsilon_i$ ) ⊆ β
                 THEN Set  $\varepsilon_i$ → n_2.List
                        n_1.split_link ← n_2.split_link;
                        x.child_link ← n_1 ;
                 Compare (x.parent_link, n_2) ;
```

First, generate a virtual root node, and then, by the order of their support, generate nodes from FTC. Secondly, compare the MSL of table List in the new node N with the node p in the old tree.

CaseI. If p does not exist, make n under the root node.
CaseII. If the MSL of table List in n is included in p, then make n under p.
CaseIII. If the INTERSECT of the MSL of table List in n and the one in p is $\varnothing$, then check the parent node of p.
CaseIV. If the INTERSECT of the MSL of table List in n and the one in p is not $\varnothing$ but n_1 and n_2, use split link to connect these two nodes. Update the INTERSECT of the MSL of table List in n and the one in p to n_1, then Update the DIFFERENCE of the MSL of table List in n and the one in p to n_2. Make n_2 under p. Compare n_2 with the parent node of p again.

According to this approach, the frequent tag and character data can be stored in FTCP-split tree.

Example 8: Take Figure 3(a~d) for example. We can obtain a FTCP-split tree as Figure 4. Character data are too less to build on the FTCP-split tree because the number of documents isn't enough in this example.



Head Table			
Element	Count	B	Link
B	7	1	
A	4	1	
D	4	0	
E	4	0	
C	3	1	

Figure 4: FTCP-split tree

Step4. Mining association patterns

In this step, we adapt FP-growth algorithm for mine all frequent tag and character data with length of *k* (also called k-patterns) from the FTCP-split tree [[5]]. We defined k-patterns = $(\varepsilon_1, \varepsilon_2, \cdots, \varepsilon_k)$. We adapt the calculation of threshold, Support. It was defined as above.

Example 9: Take Figure 4 for example. Let S_u = 75% = (4*0.75 = 3) and S_l = 50% = (4*0.50 = 2). We will mine a set of 2- tags and character data ｛A, E｝= {(2 , 2.2), (3 , 3.[2, 4]), (4 , 4.1)}. The support of ｛A, E｝ is 3. Therefore, ｛A, E｝=FTC_2 = $(\varepsilon_1, \varepsilon_2)$

In the above step, we only preserve the FTCP-split tree and a set of pre-frequent tag and character data. The purpose of preserving FTCP-split tree and PFTC lies in the maintenance of data insertion, deletion, and modification. The information, FTCP-split tree and PFTC, would be updated when the time goes by. When we dynamically maintain the updatable XML database, we only need to rescan the FTCP-split tree. Since we do not need to rescan the original database, our process time is greatly reduced.

The FTCP-split tree is a remarkably thin upper-layer and fat lower-layer shape. For given a XML database, we can find that most significantly frequent tags are stored in the upper-layer because they have greater support. Therefore, the tags stands in the upper-layer part of FTCP-split tree will be compressed most.

Step5. Generating compression rules and calculating compression space

We apply the previously explored frequent tags and character data sets to establish compression rules of tags and character data sets, respectively, and to calculate spaces for compression. In addition, the utilized compression rules of character data sets and tag sets and corresponding Lists are stored in the metarule, respectively, and the used compression rules will be marked.

In our research, we adopt the compression types from the research of Lee et al to translate all length=1, 2, ..., K frequent sets into compression rules [[8]]. Next, we calculate the compression space with the compression rules.

DEFINITION Utilize frequent element set to generate compression rules

According to the FTCP-split tree, we generate a set of compression rules. This set of compression rules has their length of k ($k \geq 1$). They can be shown as follows:

$t\,(P_1, \cdots, P_{j1\text{-}1}, \varepsilon_1, P_{j1+1}, \cdots, P_{j2\text{-}1}, \varepsilon_2, P_{j2+1}, \cdots, P_{jk\text{-}1}, \varepsilon_k, P_{jk+1}, \cdots, P_n)$ $\leftarrow t'(P_1, \cdots, P_{j1\text{-}1}, P_{j1+1}, \cdots, P_{j2\text{-}1}, P_{j2+1}, \cdots, P_{jk\text{-}1}, P_{jk+1}, \cdots, P_n)$, Γ

The $P_1, \cdots, P_{j1\text{-}1}, P_{j1+1}, \cdots, P_{j2\text{-}1}, P_{j2+1}, \cdots P_{j\,k\text{-}1}, \cdots P_{jk+1}, P_n$ among the structure of the DTD represent the variables tag or variables character data. The ε_i represents the compressed element, for i=1, 2, ⋯, k. The Γ represents the EC_Ω of the frequent element set. The compress space of the rule can be shown as: $\sum_{i=1}^{k} B(\varepsilon_i) * \mathbf{C}$. The Count is the total number of equivalence class correspond to the total number of times of the ε_i appear in XML database. The information of count was stored in the tree. The B(ε_i) is the memory space of the ε_i. For i =1, 2, ⋯, n. The ε_i is a set in this rule. Because there would be more than two character data can hold same tags. For example: Two character data, D and E, hold same tag, <C >.

Example 10: To follow Example 9. We take the element A and E for example. Given $EC_{<\text{A, E}>}$= { (2 , 2.2) , (3 , 3.[2, 4]) , (4 , 4.1) } , We can generate a compression rule as follows:

t$_1$ (A ，*B* ，*C* ，*D* ，E) ←t$_1$**'** (*B* ，*C* ，*D*) ， { (2 , 2.2) , (3 , 3.[2, 4]) , (4 , 4.1) }

This compression rule represent the element A and E appear in four documents according to the information of Γ = { (2 , 2.2) , (3 , 3.[2, 4]) , (4 , 4.1) } and The Count of "A" equals to 1, 1, and 1 in 2nd, 3rd, and 4th documents. The Count of "E" equals to 1, 2, and 1 in 2nd, 3rd, and 4th documents. In the compression rule, "*B*", "*C*", "*D*" represent variables tag or variables character data. They are shown as italic type. "A" and "E" represent constant tag or constant character data and are shown as regular type. We can compress the frequent tag "A" and character data "E" with this rule.

The compression space can be calculated as follows:

$\sum_{i=1}^{k} B(\varepsilon_i) * \mathbf{C} \rightarrow$ 〔 (5) * (1+1+1) 〕+〔 (1) * (1+2+1) 〕=19 (Bytes) if A's memory space is 5, E's memory space is 1.

Step6~7. Selecting effective compression rules

A heuristic compression method is developed to assist selecting effective compression rules. The conflicts of redundancy compressing have to be resolved by a heuristic compression method. During the process of compression, several compression rules may apply to the same data. Once a rule is chosen because of their potential to reach the greatest compression ratio, the remainder of these conflict rules should be re-adjust because some of them might not be enough strong to compress the data sets. Even if some of the remainder of the conflict rule still can used for compression rules, they can not compress so many capacity as they used to be. Therefore, the compress space should be recalculated.

Example 11: If The memory space of D is 5, E is 8, and G is 2. The information of count was stored in the tree.

t$_1$(*a* , *b* , D , E , *f*) ← t$_1$'(*a* , *b* , *f*) ，(1 , 2)

The count of D separately equals to 2, and 1 in 1st, and 2nd documents.

The count of E separately equals to 1, and 3 in 1st, and 2nd documents.

So, the compression space of t$_1$ → 〔5 * (2+1) +8 * (1+3) 〕=47

t$_2$(*a* , *c* , *d* , E , G) ←t$_2$'(*a* , *c* , *d*) ，(2 , 3)

The count of E separately equals to 1, and 3 in 1st, and 2nd documents.

The count of G separately equals to 1, and 2 in 2nd, and 3rd documents.

So, the compression space of t$_2$ → 〔8 * (1+3) +2 * (1+2) 〕=38

We first choose t$_1$ compression rule. When we choose t$_2$ compression rule, we need to consider the conflict problem. It has to revise the compression space of t$_2$ compression rule. Rule t$_1$ and t$_2$ both compress E, which makes E be counted twice when it proceeds to Rule t$_2$. That is, E [8 * (1 + 3)] has to be eliminated, and the compressible space of t$_2$ becomes [2 * (1 + 2)] = 6.

B. Phase Ⅱ *An Approach to Dynamic Compression*

An incrementally updatable native XML database means that the amount of XML documents will change over time when the database occurs insertion, deletion, or modification, which makes it an important task to dynamically maintain compressed databases. To solve that, we proposed an approach to dynamically maintain a set of compression rules. The proposed approach is based on an adjusted FTCP-split algorithm. This approach can deal with data variation efficiently.

Since the database is incrementally updatable, a ping-pong effect will occur and incur the compression performance degression, for a set of compression rules will be brought-in and then removed-out. Document insertion can possibly bring some tags or character data into the set of frequent patterns. On the other hand, document deletion can possibly remove some tags or character data out from the non-frequent sets. For addressing this ping-pong problem, we adopt the concept of two thresholds proposed by Hong *et al* [[6]]. In the phase of dynamic compression we adopt the concept of safety number proposed by Hong *et al* [[13]][[14]]. It is precisely on such grounds that our research would efficiently reduce the cost of data maintenance. For example: *time* and *memory space*.

The following procedure is proposed for adjusting the hierarchical positions of tag and character data in the FTCP-split tree. when a number of XML documents are inserted, deleted, or modified.

Procedure : Preprocessing data
Input: a differential XML database ΔXDB with the size of t, the original XML database size d, safety number f, the upper threshold Su and the lower threshold S_l.

Step1 : Parse ΔXDB to obtain a ΔC, which is a set of codes corresponding to the tags and character data in ΔXDB.

Step2 : Partition ΔC into C^F, if C^F comes from the FTCP-split tree.

C^P, if C^P is a set of pre-frequent tag and character data in the original database. and this set is created and stored in the second step of Phase I.

C^N, if C^N is a set of non-frequent tag and character data set in the original database.

Step3 : Recalculate the support of C^F and the support of C^P.

Step4 : For all patterns in $C^F \cup C^P$

Adjust the FTCP-split tree according to the new supports calculated by Step3.

Step5 : Calculate a safety number *S*, where $S = \frac{(S_u - S_l)d}{1 - S_u}$, if document insertion

$S = \frac{(S_u - S_l)d}{S_u}$, if document deletion

$S = (S_u - S_l)d$, if document modification.

Step6 : For all patterns in C^N

Check if $t + c \le f$ then reset $c = t + c$ and do nothing;

Else rescan the original database and reset $c = 0$, $d = d + t + c$.

Procedure : Adjust the FTCP-split tree as follows:

CaseI. If x and $c_1[x]$ have identical list contents and $c_1[x]$ has no child. (x.List= $c_1[x]$.List and $c_1[x]$.child=null)
Then call SWAP (x, $c_1[x]$) to alter the hierarchical order of both nodes, i.e., make $c_1[x]$ become parent p[x] of x, as Figure 5.

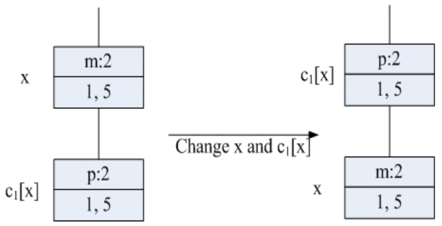


Figure 5: Case 1 of Adjust FTCP-split tree

CaseII. If x and $c_1[x]$ have identical list contents and $c_1[x]$ has children $\{c_1[c_1[x]], c_2[c_1[x]], \ldots, c_m[c_1[x]]\}$ (x.List= $c_1[x]$.List and $c_1[x]$.child=$\{c_1[c_1[x]], c_2[c_1[x]], \ldots, c_m[c_1[x]]\}$)
Then call SWAP (x, $c_1[x]$) to exchange the hierarchical order of both nodes, i.e., make $c_1[x]$ become parent p[x] of x and $\{c_1[c_1[x]], c_2[c_1[x]], \ldots, c_m[c_1[x]]\}$ become children of x $\{c_1[x], c_2[x], \ldots, c_m[x]\}$, as Figure 6.

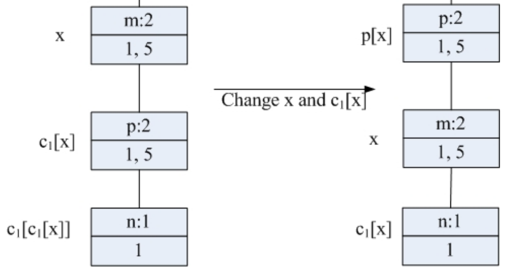
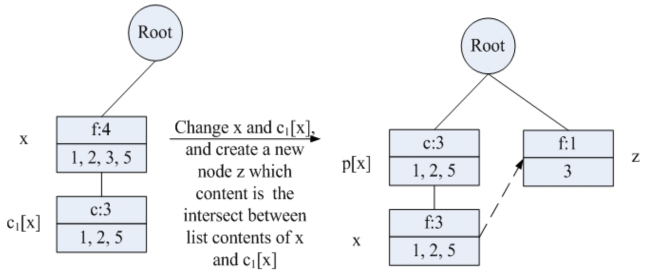


Figure 6: Case 2 of Adjust FTCP-split tree

CaseIII. If x and c1[x] have similar list contents and $c_1[x]$ has no child nodes except for $c_1[x]$. (x.child=$\{c_1[x]\}$and x.List$\cap c_1[x]$.List$\neq \varnothing$)
Then set α as the difference between list contents of x and c1[x], i.e., α=x.List - c1[x].List, create a child z under the root, and create a point of sibling between nodes x and z. Set β as the intersect between list contents of x and $c_1[x]$, as Figure 7.



Root
x
f:4
1, 2, 3, 5
$c_1[x]$
c:3
1, 2, 5
Change x and $c_1[x]$, and create a new node z which content is the intersect between list contents of x and $c_1[x]$
Root
p[x]
c:3
1, 2, 5
f:1
3
z
x
f:3
1, 2, 5

Figure 7: Case 3 of Adjust FTCP-split tree

CaseIV. If x and $c_1[x]$ have similar list contents and x other children. (α=x.List-$c_1[x]$.List $\neq \varnothing$ and x.child=$\{c_1[x], c_2[x], \cdots, c_m[x]\}$) and α$\supseteq c_j[x]$.List，for j$\neq$1)
Then set β as the intersect between list contents of x and $c_1[x]$. Create a new child Z under the root, and create a point of sibling between nodes x and z. Set $c_j[x]$ as child of z, i.e., z.chlid=$\{c_j[x]\}$. Then, make x.List=β and call SWAP (x, $c_1[x]$) to exchange the hierarchical order of the two nodes, as Figure 8.

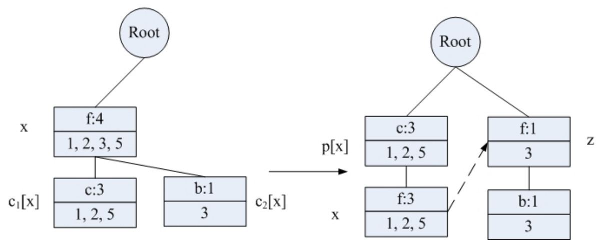


Figure 8: Case 4 of Adjust FTCP-split tree

Example 12: There are four original XML documents (d=4) in Example 1. Let a differential XML database ΔXDB {X_5, X_6}. So, t is two. Let S_u=75%=6*0.75=5，S_l=50%=6*0.5=3. The first step, we will parse the two XML documents and obtain the corresponding code for tags and character data as shown in the following figure. Therefore, the set of equivalence classes for the generated ΔC is $EC_{<A>}$= {5, 6}, $EC_{<B>}$= {6.1}, $EC_{<C>}$= {5.1, 5.2 , 5.3 , 6.2} , $EC_{<D>}$= {5.1 , 6.1}, $EC_{<F>}$= {5.2 , 5.3 , 6.2}.

5<A> X_5	6<A> X_6
5.1 <C>D</C>	6.1 <B>D</B>
5.2 <C>F</C>	6.2 <C>F</C>
5.3 <C>F</C> (a)	</A> (b)
</A>	

Figure 9: Two encoded XML documents, which are (a) X_5 and (b) X_6, respectively.

The second step, we can partition ΔC into C^F ={A , B , C}, C^P ={F} and $C^N = \varnothing$. For each element in C^F and C^P , we recalculated their corresponding supports. The following step is to adjust the FTCP-split tree which is according to the new supports calculated as above. After that, we calculate the safety number as $f = \left\lfloor \frac{(S_u - S_l)d}{1-S_u} \right\rfloor = \left\lfloor \frac{(0.75-0.5)4}{1-0.75} \right\rfloor = 4$.

Owing to the number of inserted XML documents（t＝2）and the initial parameter（c＝0）, we find that t+c (=2) is less than the safety number (f=4). Thus it doesn't need rescan the original database.

From the above steps, we just analyze the differential XML (ΔXDB) database and need not to rescan the updated database ($XDB + \Delta XDB$), we can also obtain the adjusted FTCP-split tree for late compression rule generation. By using the new created set of compression rules, we can efficiently compressing the incremental updatable native XML database.

IV. EXPERIMENT ANALYSIS

In this initial experiment, we will proceed with experimental evaluation for compression rate of compressing Native XML Database. We have completed compression analysis of the character data sets with a length of 1 and details are given as follows:

The develop platform is Java programming language (J2SDK 1.4.2) to develop this experiment. The hardware is Intel P4-2.8G, with memory of 2GB, and the operating system is the Microsoft Windows 2000 Professional. Furthermore, Assoc.gen is available from the IBM official website, and is used to generate the transaction database for this experimental analysis.

Analysis of experiment analysis

The transaction records are generated by Assoc.gen, and then converted into XML documents for the experiment, where each transaction record stands for one XML document and the DTD structure is as shown in Figure 2.

Time of building tree

As a result of our research adapted the structure of split tree from Lee [[11]]. We can preserve whole original XML documents in FTCP-split tree without destroying the tree structure of original XML documents. It will be clear from the experiment of Lee that the time of building tree is less than traditional FP-tree approach [[11]].

The compression effectiveness of documents with different sizes

In this experiment, there are 2500, 5000, 7000, and 10000 XML documents simulated for evaluating the compression rate with different document sizes and we set the parameter such as: the average length of documents as 20, the average length of frequent sets as 10, the number of items in the database as 100, and the relationship of frequent sets as 1. Figure 10 shows tag, character data, and whole compression rate under the different amount of documents. Compression ratio is defined by the expression of A / B, where A represents the data volume for compressed tag or character data or total (tag + character data), and B represents the data volume for XML documents and compression rules.

Compare our compression rate with ZIP and RAR

Figure 11 shows the compression rate compare bar chart among our research, ZIP, and RAR under the minimum support as 30%. We adopt XML database belonged to the structure of the Figure 2 DTD D. This figure indicates that the compression rate is higher than ZIP and RAR when we compress the length=2 character data sets and the length=1 tag sets. (It can be shown as curve B).

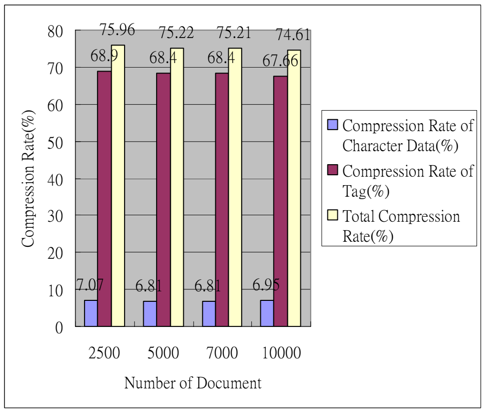


Figure 10: The effectiveness of min-sup10 (%)

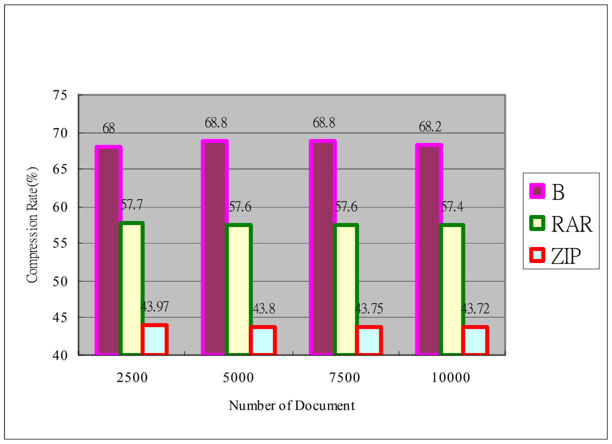


Figure 11: The compression rate compare bar chart among our research, ZIP, and RAR.

V. CONCLUSION

The use of the Internet to exchange electronic business documents (eBusiness) is growing exponentially. XML acts as the best way to exchange information because it is a standard defined language by the World Wide Web Consortium (W3C), and is a simple, easy-to-grasp method of encoding information in plain text. In addition to being a means for moving data over the Internet, XML files provide a good way of moving data among applications. Thus, the capacity for storing XML documents is growing fast. Database compression can address this problem.

The first purpose of our proposed approach is to utilize technology of association rule mining to extract out all frequent patterns. The frequent patterns can be stored in a frequent tag and character data pattern split tree (FTCP-split tree) to fast generate the set of association patterns. Then we convert these frequent patterns into a set of compression rules. The compression rules can be used to compress native XML databases. Moreover, we also can use the association patterns to generate a set of association rules which usually are reliable and valuable.

The second purpose of our proposed approach is to solve the problem of data maintenance when compressed database occur variation. We proposed an efficient approach named as Adjust FTCP-split algorithm for incremental mining to solve it. The features of this method are fast mining and high compression rate. First, since the Adjust FTCP-split approach do not generate large amount of candidate set, it can be quicker than traditional the traditional Apriori-like approaches. Second, the experiment results show that the compression rate of our proposed approach is higher than the common compression tools such as ZIP and RAR. Third, our proposed approach can dynamically maintain compression rules when database occur variation such as insertion, deletion, or modification. According to the above three features, our proposed approach can reach the goal of reducing compression cost and raising compression rate.

REFERENCES

[1] Babu, S., Garofalakis, M., & Rastogi, R. 2001, 'SPARTAN: A Model-Based Semantic Compression System for Massive Data Tables', *Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD'01)*, pp. 283-294.

[2] Cheung, David W., Han, Jiawei, Ng, Vincen T., & Wong, C.Y. 1996, 'Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique', *Proceedings of International Conference on Data Engineering*, New Orleans, Louisiana, pp.106-114.

[3] Cheung, David W., Lee, S. D., & Kao, Benjamin 1997, 'A General Incremental Technique for Maintaining Discovered Association Rules', *Proceedings of the 5th International Conference on Database Systems for Advanced Applications (DASFAA)*, pp.185-194.

[4] Goh, C. L., Aisaka, K., Tsukamoto, M., Harumoto, K., & Nishio, S. 1998, 'Database Compression with Data Mining Methods', *Proceedings of 5th International Conference on Foundations of Data OrganiPation (FODO'98)*, Kobe, Japan, pp. 97-106.

[5] Han, J., Pei, J., and Yin, Y. 2000, 'Mining Frequent Patterns without Candidate Generation', *Proceedings of the ACM SIGMOD International Conference on Management of Data(SIGMOD'00)*, pp. 1-12.

[6] Hong, T. P., Wang, C. Y., Tao, Y. H. 2000, 'Incremental Data Mining Based on Two Support Thresholds', Proceedings of the 4th International Conference on Knowledge-Based Intelligent Engineering Systems and Allied Technologies, pp.436-439.

[7] Hsieh, S. F. 2002, 'An Efficient Approach for Maintaining Association Rules Based on Adjusting FP-tree Structure.', Thesis for The Degree for Master, The Graduate Institute of Information and Computer Education, National Taiwan Normal University.

[8] Lee, C. F. and Tang, C. M. 2004, 'A Compression Approach with Dynamic Maintenance on Native XML Database via Incremental Updating Techniques', Proceedings of the ACME International Conference on DB,DSS&EIS.

[9] Lee, C. F., Changchien, S. W., & Wang, W.T. 2001, 'Using Generated Association Mining for Object-Oriented Database Compression', *National Computer Symposium—Database & Software Engineering*, pp. 151-162.

[10] Lee, C. F., Changchien, S. W., & Wang, W.T. 2003, 'Association Rules Mining for Native XML Database', *Department of Information Management, Chaoyang University of Technology*, Taichung, Taiwan, CYUT-IM-TR-2003-011.

[11] Lee, C. F., Shen, T.H. 2005, 'A FP-split Method for Fast Association Rules Mining', Proceedings of the 3rd International Conference on Information Technology: Research and Education.

[12] Lee, C. F., Tang, C.M. 2005, 'Dynamically Compressing XML Tags and Data Characters via Incremental Updating Mining', *Proceedings of the International Association for Computer Information Systems Conference*.

[13] Wang, C. Y., Hong, T. P., Tseng, S. S. 2001, 'Maintenance of Sequential Patterns for Record Deletion', *Proceedings of IEEE International Conference in Data Mining*, pp.536-541.

[14] Wang, C. Y., Hong, T. P., Tseng, S. S. 2002, 'Maintenance of Sequential Patterns for Record Modification Using Pre-large Sequences', *Proceedings of IEEE International Conference in Data Mining*, pp.693-696.