IMTAR: Incremental Mining of General Temporal Association Rules

Anour F.A. Dafa-Alla*, Ho Sun Shon*, Khalid E.K. Saeed*, Minghao Piao*, Un-il Yun**, Kyung Joo Cheoi** and Keun Ho Ryu*

Abstract—Nowadays due to the rapid advances in the field of information systems, transactional databases are being updated regularly and/or periodically. The knowledge discovered from these databases has to be maintained, and an incremental updating technique needs to be developed for maintaining the discovered association rules from these databases. The concept of Temporal Association Rules has been introduced to solve the problem of handling time series by including time expressions into association rules. In this paper we introduce a novel algorithm for Incremental Mining of General Temporal Association Rules (IMTAR) using an extended TFP-tree. The main benefits introduced by our algorithm are that it offers significant advantages in terms of storage and running time and it can handle the problem of mining general temporal association rules in incremental databases by building TFP-trees incrementally. It can be utilized and applied to real life application domains. We demonstrate our algorithm and its advantages in this paper.

Keywords—Incremental Mining of General Temporal Association Rules, Incremental TFP-Tree

1. INTRODUCTION

In the field of data mining, association rules play an important role and have been applicable in many areas. The problem of mining association rules can be divided into two sub-problems 1) finding frequent itemsets. 2) Using those itemsets to generate the rules. After the frequent itemsets have been recognized, the corresponding rules may be derived easily.

Agrawal's pioneering work [1] has led to many proposals of mining association rules such as incremental mining approaches [2], updating approaches [8], tree based approaches such as [9], [10] and [11] and various formations of rule patterns such as temporal patterns [3, 4]. In general, due to the recent developments in information science, most transactional databases accumulate a small size of incremental databases and are being appended into the main database regularly. Thus, designing efficient incremental mining algorithms is becoming particularly urgent.

※ This research was supported by the Maritime Affairs and by the Ministry of Education, Science Technology (MEST) and Korea Industrial Technology Foundation (KOTEF) through the Human Resource Training Project for Regional Innovation, and by the grant of the Korean Ministry of Education, Science and Technology (The Regional Core Research Program / Chungbuk BIT Research-Oriented University Consortium).

Manuscript received February 2, 2010; accepted April 6, 2010.

Corresponding Author: Ho Sun Shon

* Database/Bioinformatics Lab., Chungbuk National University, Cheongju, South Korea {anwarking, abolkog, bluemhp, khryu}@dblab.chungbuk.ac.kr

** School of Electrical & Computer Engineering, Chungbuk National University, Cheongju, South Korea

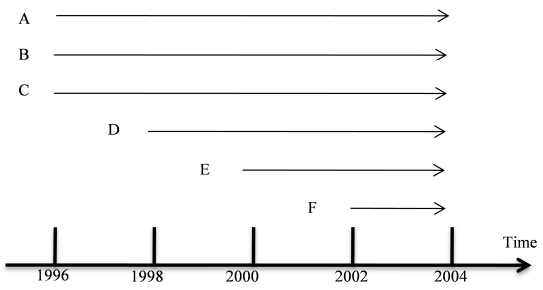

Fig. 1. An Example of a publication database

In this paper we introduce a new algorithm for mining general temporal association rules in publication databases called IMTAR (Incremental Mining of Temporal Association Rules). A publication database is a transactional database where each item involves an individual exhibition period [5]. Consider the publication database in Fig. 1, items A, B and C are exhibited from 1996 to 2004. However, item F is exhibited from 2002 to 2004. Traditional mining techniques tend to ignore the exhibition period of each item and instead opt to use a unique minimum support threshold which is unfair for newly added transactions.

General temporal association rules are of the form $(X \Rightarrow Y)^{t,n}$, where *t* denotes the *starting exhibition period* of both items *X and Y*, and *n* denotes the *end of publication database*. An association rule (X⇒Y) is said to be a *frequent general temporal association rule* $(X \Rightarrow Y)^{t,n}$, if and only if its probability is larger than the *minimum support required*, i.e., $P(X^{t,n} \cup Y^{t,n}) > min\_sup$, and the conditional probability is larger than the *minimum confidence* i.e., $P(X^{t,n} \mid Y^{t,n}) > min\_conf$. Using the *absolute minimum support threshold* can be unfair for some items in the publication database as shown in Fig. 1. Instead a *relative minimum support* is used, $min\_sup_R = |D_X| \times min\_sup$ where $|D_X|$ denotes the amount of partial transactions in the exhibition period of itemset X. This is provided to deal with the mining of temporal association rules.

F. Conen et al. introduced a new technique for mining association rules using TFP-trees (Total from Partial tree) [6]. We extended the structure of the TFP-tree to handle the problem of mining general temporal association rules. Our proposed algorithm inherits the significant advantages of the TFP-tree and introduces new important capabilities of handling temporal association rules incrementally.

The rest of this paper is organized as follows: Section 2 introduces some of the related work on incremental mining of association rules. Section 3 describes the proposed algorithm IMTAR. Conclusions and future work are discussed in section 4.

2. RELATED WORK

D.W Cheung introduced a new algorithm called the *Fast Update algorithm (FUP)* [2], for efficient maintenance of discovered association rules when new transactions are added. It's similar in its frame work to that of Apriori [1] and *Direct Hash and Pruning DHP* [12]. It contains a number of iterations starting at *size-one* itemsets; at each iteration, all the large itemsets of the same size are founded. The candidate sets at each iteration are generated based on the large itemsets founded in the previous iteration.

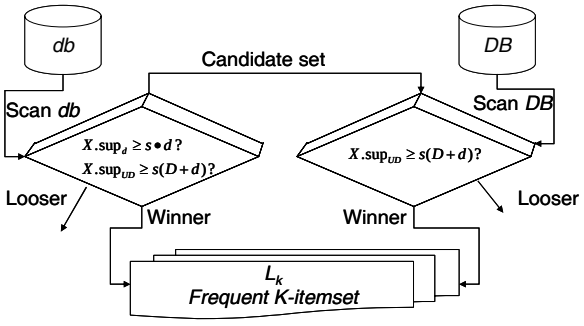

Fig. 2. The first iteration of FUP

FUP introduces the following feature to handle the incremental database and to update the discovered rules. At each iteration the support of *size-k* frequent itemsets are updated against the *incremental database* to filter out the *loser* (i.e., those that are not longer frequent in the updated database). Only the incremental database has to be scanned to do this filtering. While scanning the incremental database, a set of candidates is extracted along with their support from the incremental database.

Lemma 1: *A k-itemset not in the original large k-itemset can become a winner in the updated database if, and only if, its support is greater than or equal to the minimum support threshold in the incremental database.*

The proof to this lemma is shown in [2]. According to this lemma, many sets can be pruned away simply by checking their support against the incremental database before the update against the original database. The size of the updated database is reduced at each iteration by pruning away some itemsets from some transactions in the updated database. Fig. 2 shows the first iteration of an *FUP* algorithm. Due to the limited length of this paper, we will show only the first iteration, the second iteration and beyond can be thought of as a generalization of the first iteration.

3. INCREMENTAL MINING OF GENERAL TEMPORAL ASSOCIATION RULES

In this section we will describe our proposed algorithm *IMTAR* (Incremental Mining of General Temporal Association Rules) inspired by *Apriori-TFP* [6], a successful structure to mine frequent patterns. We use a tree structure called the TFP-tree as the basis for our method. A TFP-tree is a set enumeration tree structure [7], storing the quantitative information about itemsets. The TFP-tree in its current structure is not suitable for mining temporal frequent itemsets; therefore we extended and modified the tree structure to suit the need of mining temporal frequent itemsets. We will now describe the major modifications to the P-tree and T-tree of the Apriori TFP-method to make it suitable for mining temporal patterns. Some of the notations used in the rest of this paper are shown in Table 1.

The partial support tree (P-tree) is a compressed set enumeration tree which is used to store

Table 1. Notations description

Notation	Description
DB	Original database
db	Incremental database
UD	Updated database
$TFP\text{-}Tree_{DB}$	TFP-Tree of the original database
$TFP\text{-}Tree_{db}$	TFP-Tree of the incremental database
$TFP\text{-}tree_{UD}$	TFP-Tree of the updated database
$X.supp_{DB}$	Support count of item X in the original database
$X.supp_{db}$	Support count of item X in the incremental database
$X.supp_{UD}$	Support count of item X in the updated database

partial support for itemsets. The top level is comprised of an array of nodes, each index describing 1-itemset, with child reference to P-tree nodes. The top nodes consist of a support field for the itemset and a child node link to a sub node, we extended one more fields to handle these at which time this node starts to appear i.e., at the start of the exhibition period.

The P-tree node contains support fields – an array of short integers- for the itemset that the node represents and a child and a sibling link to further sub nodes and one field to store at which time the item represented by this node starts to appear. The modified P-tree is shown in Fig. 3.

The total support tree (T-tree) is a set enumeration tree for storing itemsets information. Levels in each sub-branch of the tree are defined using arrays. This thus permits indexing at all levels and consequently offers computational advantages. To assist this indexing the tree is built in "reverse." Each branch is founded on the last element of the frequent sets to be stored. This allows direct indexing by attribute number rather than first applying some offset. The same modification done to the P-tree was applied to the T-tree to handle temporal association rules. This is

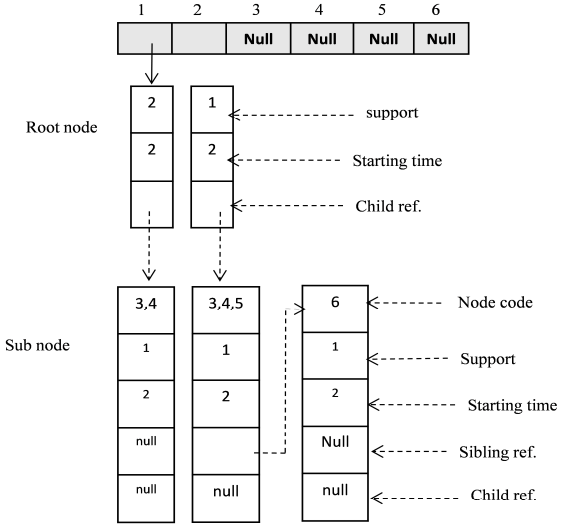

Fig. 3. P-tree structure for mining general temporal association rules

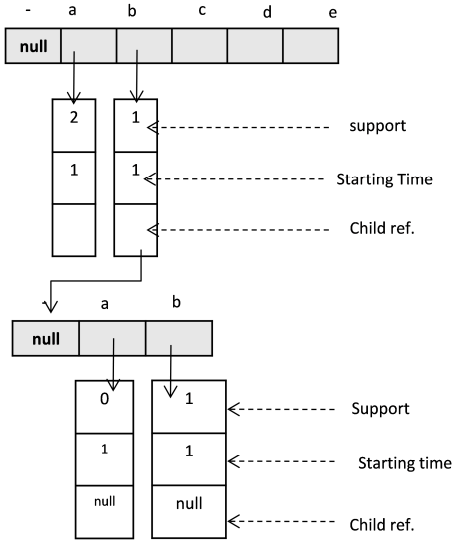

Fig. 4. T-tree structure for mining general temporal association rules

represented in the modified T-tree shown in Fig. 4.

Now we will present the details of our proposed algorithm for mining general temporal association rules. To better illustrate our algorithm, consider the data presented in Fig. 5 and assume that the given minimum support threshold is 30%. This figure shows an example of a publication database where each item is associated with a publication date. The Date field corresponds to the start of the publication date for the itemset, and the TID is the transaction id that contains the set of items.

We start by partitioning the publication database according to time interval into three partitions namely, P_1, P_2 *and* P_3, as shown in Fig. 5, and processing each partition one at a time. Par-

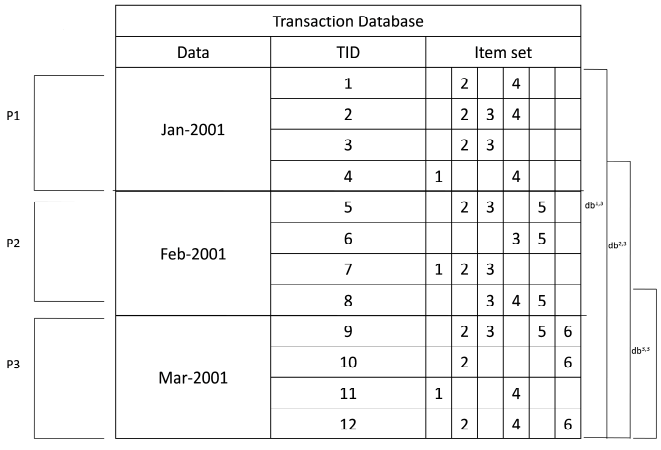

Fig. 5. P-tree structure for mining general temporal association rules

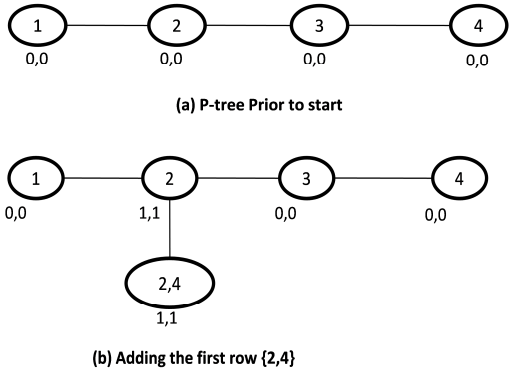

Fig. 6. Constructing a P-tree$_{DB}$ for partition one, P_1

tition one can be traded as the original database and each successive partition (P_2 & P_3) can be thought of as the incremental part that will be appended to the first partition P_1. Considering the first partition p_1, we construct a TFP-tree for this partition *TFP-tree*$_{DB}$, ignoring the pruning step as per the original Apriori-TFP. Ignoring the pruning step is essential – as we will see later – in the incremental mining procedure for finding frequent itemsets.

If we follow the original algorithm some of the items which were not frequent but can be frequent later after adding the incremental database will be lost. The first step in constructing the *TFP-tree*$_{DB}$ is to construct a *P-tree*$_{DB}$ of the first partition p_1. The *P-tree*$_{DB}$ prior to start is shown in Fig. 6.a, and after adding the first transaction {2, 4} is shown in Fig. 6.b, where the two numbers below the node denote the support count and the starting exhibition period respectively *{support, starting exhibition period}*, for example if we look at node 2 we will see that its support count is 1 and starting exhibition period is 1 *{1, 1}*, note that the parent node is also incremented by one.

After storing the first transaction in the *P-tree*$_{DB}$, we now process the next transactions and add them to the *P-tree*$_{DB}$ in a manner similar to adding the first transaction. Fig. 7, shows the final *P-tree*$_{DB}$ after processing partition one. Note that nothing is lost if we don't store part of any particular node code which is duplicated in the parent code.

On completion of the generation process, the *P-tree*$_{DB}$ itself is thrown away and stored in a *P-tree*$_{DB}$ table. The advantages offered by the P-tree table are: 1) Reduced storage requirements (Particularly where the data set contained duplicated rows). 2) Faster run time because the de-

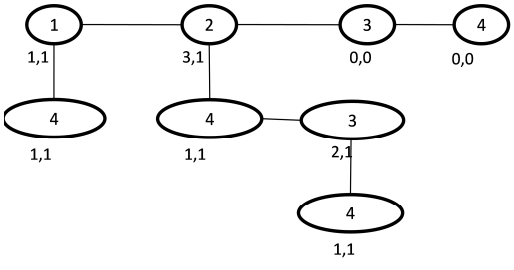

Fig. 7. P-tree$_{DB}$ after processing partition one, P_1

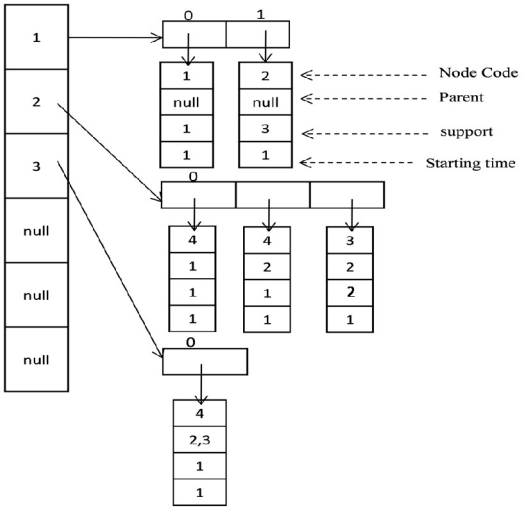

Fig. 8. P-tree$_{DB}$ table for partition one

sired total support counts had already been partially calculated. Fig. 8, shows the *P-tree$_{DB}$* table for partition one.

The *T-tree$_{DB}$* is then constructed from the *P-tree$_{DB}$* table shown in Fig. 8, by processing the P-tree table level by level. Each level in the P-tree table represents the *set cardinality for the item-sets i.e.*, level l represents size *1-itemsets*, level 2 represents size *2-itemsets* and so on.

The *T-tree$_{DB}$* construction is as follows: Starting with an empty top level *T-tree$_{DB}$*, passing through the *P-tree$_{DB}$* table level by level, starting at level 1, updating the top level of the *T-tree$_{DB}$* according to the hierarchical nature of the *P-tree$_{DB}$* table. The first level in the P-tree table has two records, representing nodes {1} and {2}, thus we updated the support count and the starting exhibition period for elements 1 and 2 in the top level of the *T-tree$_{DB}$* as shown in Fig. 9.

We now pass down to the second level in the *P-tree$_{DB}$* table (index 2), which represents doubles (set cardinality =2). There are three records at this level; the first one has parent *node* {1} and *node code 4*. The contribution of this node to element 1 (the parent node) in the *T-tree$_{DB}$* has already been considered. Thus, as before we only consider the node code and update the support and the starting exhibition period for element 4 in the T-tree$_{DB.}$ The second one has *parent node* {2}, and *node code* {4}, so as before we only update element 4. The last record has *parent node* {2} and *node code* {3} thus we update element 3 in the T-tree$_{DB}$.

Then we drop down to level 3 in the *P-tree$_{DB}$* table, there is one record in this level , this has the node code of {4} and parent node code {2,3}, like before we only consider the node code and thus we update element 4 in the *T-tree$_{DB}$*. Figure 10 shows the T-tree$_{DB}$ after the first pass of

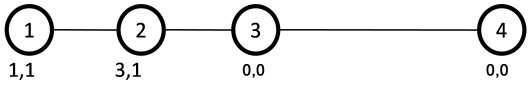

Fig. 9. T-tree$_{DB}$ generation process, passing through P-tree$_{DB}$ table level 1

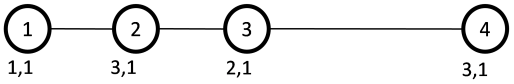

Fig. 10. T-tree$_{DB}$ generation process, passing through P-tree$_{DB}$ table level 2 and level 3

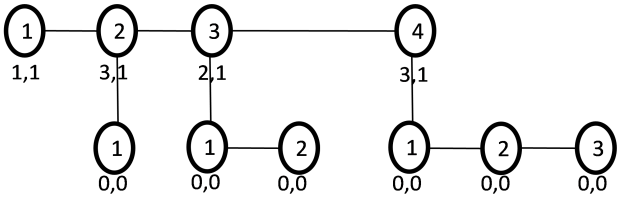

Fig. 11. T-tree$_{DB}$ generation, level 2 prior to start

the P-tree$_{DB}$.

We have now completed one pass of the *P-tree$_{DB}$* and we can remove the singles from the *P-tree$_{DB}$* table i.e., index 1, and generate the next *T-tree$_{DB}$* level as shown in Fig. 11.

Continuing on to level two in the *T-tree$_{DB}$,* we start from set cardinality 2 in the *P-tree$_{DB}$* table. There are three cardinality nodes at this level in the table; the first one has parent node {1} and node code {4}. Therefore we search the *T-tree$_{DB}$* branches emerging from element {4} only (the node code) and do not consider the branches emerging from element {1} (the parent code). Thus we pass down branch {4} to the second level and attempt to update elements {1} and {4} if necessary.

The second record encountered has node code {4} and parent node code {2}, so we also search the branch emerging from element 4 and drop down to the appropriate level, and attempt to update element {2,4} if necessary. The last record has parent node {2} and node code {3} so we search the branch emerging from element 3 , dropping down to level two and try to update element {2,3} if necessary.

We then move to the cardinality 3 nodes of the *P-tree$_{DB}$* table, there is only one record in this level, this has parent node {2,3} and node code {4}, like before we search the branches emerging from element {4} and attempt to update elements {2},{3} and {4} if necessary. After finish-

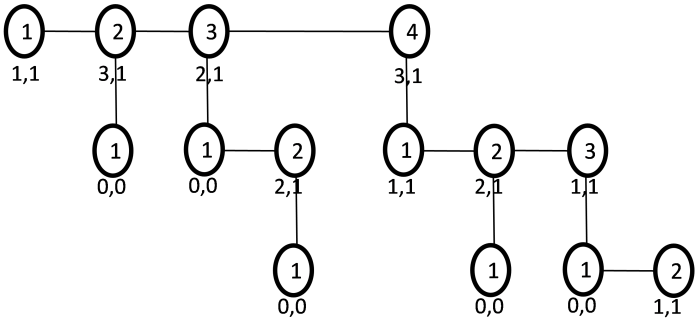

Fig. 12. T-tree$_{DB}$ after processing the first partition, p_1

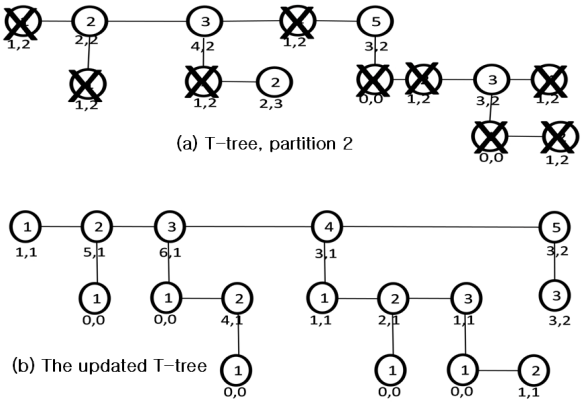

Fig. 13. (a) T-tree, partition 2 (b) updated T-tree: combined trees from partition 1 and 2

ing this pass we can remove the set cardinality 2 (i.e., index 2) from the *P-tree*$_{DB}$ table and generate the next level in the *T-tree*$_{DB}$. Similarly, updates take place for the third level of the *P-tree*$_{DB}$ table. Fig. 12 shows the final *T-tree*$_{DB}$ after processing partition one p_1.

After processing the first partition p_1, for each remaining partition, we try to find the frequent items in that partition following the original construction of the *TFP-tree*. According to *Lemma 1* introduced earlier in section two, an itemset can be frequent in the updated database if and only if it is frequent in the incremental database. So by pruning the non-frequent itemsets, we can reduce the size of the incremental part before appending to the original database. After processing the current partition we have a compressed *T-Tree*$_{db}$ due to the pruning technique, we try to add the new *T-Tree*$_{db}$ to the tree constructed from the first partition *T-tree*$_{DB}$. For each item in the *T-Tree*$_{DB}$ we update its support count, $X.supp_{UD} = X.supp_{db} + X.supp_{DB}$. This process is iterated for each remaining partition. After we found the frequent items from all partitions and combined them with the tree constructed from the first partition we ended up having the tree for the whole updated database, *T-tree*$_{UD}$, then we pruned the *T-tree*$_{UD}$ to filter out the items that were inadequately supported. Fig. 13.a shows the tree of the incremented database *db* constructed from partition p_2. The tree from updated database *UD* after combining *T-tree*$_{db}$ to it is shown in Fig. 13.b.

After all partitions have been processed we attempt now to prune the tree to filter out those items that are inadequately supported. Fig. 14 shows the final *T-tree*$_{UD}$ after the pruning step for

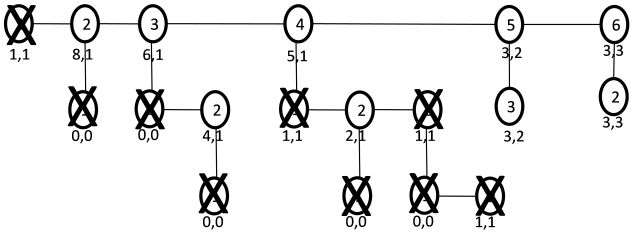

Fig. 14. The final T-tree$_{UD}$

Table 2. Pseudo code for the IMTAR algorithm

Input: Set of temporal transactions, Minimum Support threshold. **Output:** General frequent patterns.	
1	Partition the database *DB* into P_i partitions according to time interval.
2	Scan partition P_1 and construct $T\text{-}tree_{DB}$ without pruning
3	For each remaining partition P_i :
4	Construct T-tree_{db} for P_i following the original TFP with pruning
5	// Combine the $T\text{-}tree_{DB}$ and $T\text{-}tree_{db}$ to get the updated $T\text{-}tree_{UD}$:
6	$\text{T-tree}_{UD} = \text{T-tree}_{DB} + \text{T-ree}_{db}$.
7	if (current item exists in $T\text{-}tree_{DB}$) //update its support
8	$X.supp_{UD} = X.supp_{DB} + X.supp_{db}$;
9	else //i.e., new item
10	add the item to the tree, update its support and start time
11	End
12	Prune T-tree_{UD} to filter outer inadequately supported items.
13	Return $\text{T-tree}_{UD.}$
14	END

the itemset along with their appearing times.

After all partitions have been processed we attempt now to prune the tree to filter out those items that are inadequately supported. Fig. 14 shows the final $T\text{-}tree_{UD}$ after the pruning step for the itemset along with their appearing times.

Algorithm 1 shows the proposed algorithm IMTAR for incremental mining of temporal association rule.

4. EXPERIMENTAL RESULTS

Our experiments were conducted on a personal computer Pentium 4, 2.4 MHz with 2GB RAM, and we implemented the proposed algorithm using java programming language.

To evaluate the proposed algorithm we conducted several experiments using synthetic data generated in a manner similar to [1, 5], and a real life dataset obtained from Korea Electrical Power Research Institute (KEPRI). This data contains 11,228 transactions obtained in a time interval of 15 min for one day. We generalized this time interval and used a window of 12 hours (half a day) to represent the original transaction and the incremental and thus we have two partitions in our data. Fig. 15 shows the running time performance of IMTAR using both the real life dataset and the synthetic data. Since the power load data is from the real world, it is consequently more complicated than the synthetic data. Therefore, when the given support is small, the number of frequent patterns from the power load data is much higher than from the synthetic data set. Because of this, the running cost of IMTAR on power load data will be significantly higher than tests on synthetic data when the support is small. However, the cost will equalize as the support is increased.

We also compare the number of frequent patterns generated by our algorithm IMTAR against FP-growth, as can be seen in Fig. 16. Since IMTAR ignores some patterns that are possibly frequent when combining the *db* and *DB*, the number of mined frequent patterns is smaller than the

result from the FP-growth algorithm.

For our last experiment, we evaluated the scale-up performance of the proposed algorithm IMTAR. In this experiment we used a different transactional dataset as shown in Fig. 17, where D_X denotes the size of the dataset considered, we changed the support threshold value from 1% to 4%. As we can see in Fig. 17, the execution time slightly increases as the size of the data increase, showing good scalability of the proposed algorithm *IMTAR.*

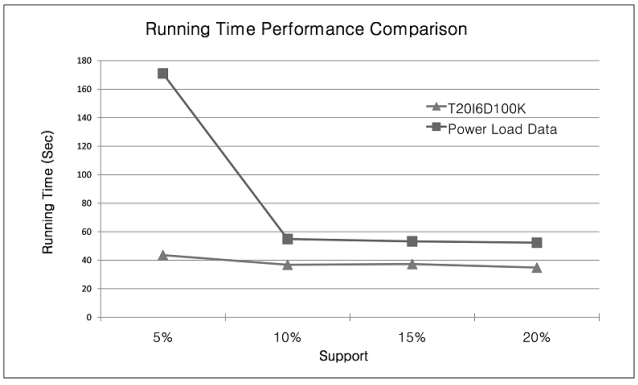

Fig. 15. Performance of IMTAR using real life dataset and synthetic dataset

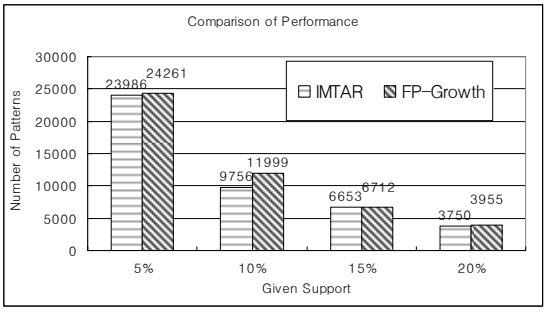

Fig. 16. Number of frequent patterns

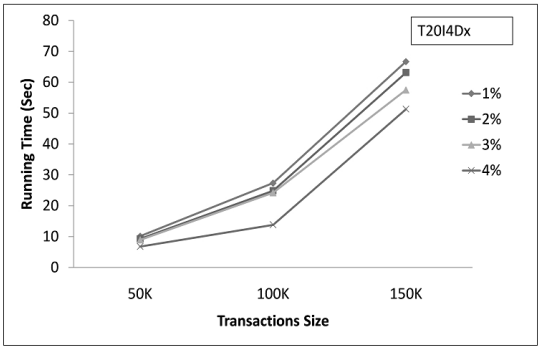

Fig. 17. Number of frequent patterns

5. CONCLUSION

Rapid advances in the field of information science have led to large size databases. Databases are now frequently and periodically updated. This brings about the need for algorithms to handle and maintain the information obtained from these databases. The concept of Temporal Association Rule was introduced to handle time series in association rules.

In this paper we introduced a new algorithm called, *IMTAR* for mining general temporal association rules in publication databases based on the *TFP-Apriori* tree. Our algorithm has the advantage of building the tree incrementally from the incremental databases without the need to rescan the original database. Furthermore our proposed algorithm extends the structure of the TFP-tree so that it can handle general temporal association rules. Our experiments showed that the proposed algorithm IMTAR has good performance with synthetic data and real life data as well as good scalability.

We conclude that IMTAR is an efficient algorithm for incremental mining of general temporal association rules, and it can be utilized within real life application domains.

REFERENCES

[1] R. Agrawal, T. Imielinski, A. Swan, “Mining Association Rules Between Set of Items in Large Databases”, In ACM SIGMOD International Conference on Management of Data, pp.207-216, 1995.

[2] D. W., Cheung, J. Han, V. Neg, Y. Wong, “Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique”, In the International Conference on Data Engineering, pp.106-114, 1996.

[3] W. Wang, Y. Yang, R. Muntz, “Temporal Association Rules with Numeric Attribute”, In NCLA CSD Technical Report. 1999.

[4] W. Wang, Y. Yang, R. Muntz, R., “Temporal Association Rules on Evolving Numerical Attribute”, In the 17th International Conference on Data Engineering, pp.283-292, 2001.

[5] C.H. Lee, M.S. Chen, C.R. Lin, C, R., “Progressive Partition Miner: An Efficient Algorithm for Mining General Temporal Association Rules”, In IEEE Transaction on Knowledge Engineering, pp.1004-1017, 2003.

[6] F. Conen, P. Leng, S. Ahmed, “Data Structure for Association Rules Mining: T-tree and P-tree”, In IEEE Transaction on Knowledge Engineering, pp.774-778, 2004.

[7] R. Rymon, “Searching through systematic set enumeration”, In the 3rd International Conference on Principles of Knowledge and Reasoning, 1993.

[8] C. Zheng, “An Incremental updating technique for mining indirect association rules,” Proceedings of the Seventh International Conference on Machine Learning and Cybernetics, Kunming, pp.217-221, 2008.

[9] X. Li, Z.H. Deng and S. Twang, “A Fast Algorithm for Maintenance of Association Rules in Incremental Databases” Adnaced Data Mining and Application, pp.56-63, 2006.

[10] M. Adnan, R. Alhajj, and K. Barker “Constructing Complete FP-Tree for Incremental Mining of Frequent Patterns in Dynamic Databases”, Advances in Applied Artificial Intelligence, 19th International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE, pp.363-372, 2006.

[11] W. Cheung, and O. R. Zaiane, “Incremental Mining of Frequent Patterns without Candidate Generation or Support Constraint”, in Proc. IDEAS’03, 2003, vol. 1098-8068/03, p.111.

[12] J. S. Park, M. S. Chen and P.S. Yu, “An Effective Hashed-Based Algorithm for Mining Association Rules”, in proc. 1995 ACM-SIGMOD Int. Conf. Management of Data, San Joe, CA. May, 1995.

Anour F.A. Dafa-Alla

He received a Bsc. From the Arab Academy for Science and Technology, Egypt in 2003. And he obtained his MS Degree in Computer Science from Chungbuk National University, South Korea in 2006. Currently he is a PhD candidate at the same school focusing on problems of privacy preserving data mining, data publishing among other areas of database related issues.

Ho Sun Shon

She received a Bsc. in Statistics from the Sung Shin Women University of Nature Science, South Korea in 1986. And she obtained her MS Degree in Statistics from Sung Shin Women University, South Korea in 1992. And she obtained her PhD Degree in Computer Science from Chungbuk National University, South Korea in 2010. Her research interests are in the area of bioinformatics, data mining and pattern recognition.

Khalid E.K. Saeed

He received his BS in Computer Science on October 6 from the University of Egypt in 2008 and he obtained his MS Degree in Computer Science from Chungbuk National University, South Korea in 2010. His research interests are in the area of Data Mining, he has been focusing on the problems of Classification and Association Rule mining. Including topics such as Emerging Patterns Based Classifiers, Incremental Classification and Association Rules techniques.

Minghao Piao

He received a Bsc. in Computer Science from the Yanbian University of Science and technology, China in 2007. And he obtained his MS Degree in Bio Informatics from Chungbuk National University, South Korea in 2009. Currently he is a PhD candidate of Computer Science at the same school. His research interests are in the area of incremental frequent patterns mining, incremental decision tree induction and temporal patterns mining.

Un-il Yun

He received an MS degree in Computer Science and Engineering from Korea University, Republic of Korea, in 1997, and a PhD degree in Computer Science from Texas A&M University, Texas, USA, in 2005. He worked at the Multimedia Laboratory, Korea Telecom, from 1997 to 2002. After receiving a PhD degree, he worked as a post-doctoral associate for almost I year at the Computer Science Department of Texas A&M University. Afterwards, he worked as a senior researcher at the Electronics and Telecommunications Research Institute (ETRI). Currently, he is an assistant professor at the School of Electrical & Computer Engineering, Chungbuk National University. His research interests include data mining, database systems, information retrieval, artificial intelligence, and digital libraries.

Kyung Joo Cheoi

She received an MS degree in Computer Science from Yonsei University, Republic of Korea, in 1998, and a PhD degree in Computer Science from Yonsei University, Republic of Korea, in 2002. After receiving the PhD degree, she worked as a research engineer at the LG CNS, from 2002 to 2005. Afterwards, she worked as full-time lecturer at Chungbuk National University, from 2005 to 2006. Now, she is an assistant professor at the School of Electrical & Computer Engineering, Chungbuk National University. Her research interests include computer vision, image processing, pattern recognition, brain science, cognitive science and biometrics.

Keun Ho Ryu

Keun Ho Ryu is a professor at Chungbuk National University and a leader of database and bioinformatics laboratory in Rep. of Korea. He received the Ph.D degree from Yonsei University, Rep. of Korea, in 1988. He served Korean Army as ROTC. He worked not only at University of Arizona as Post-doc and research scientist in U.S.A but also at Electronics& Telecommunications Research Institute in Rep. of Korea. He has served on numerous program committees including a demonstration co-chair of the VLDB, a panel and tutorial co-chair, a co-chair of the ADMA conference, the PC committee member of APWeb, the AINA, and so on. His research interests are included in temporal databases, spatiotemporal database, temporal GIS, ubiquitous computing and stream data processing, knowledgebase information retrieval, database security, data mining, bioinformatics and biomedical. He is a member of the IEEE as well as a member of the ACM since 1983