An Efficient Technique for Indexing Temporal Databases

Mohammed M. Fouad
Faculty of Computing and Information Technology
King Abdulaziz University
Jeddah, Saudi Arabia

Mostafa G.M. Mostafa
Faculty of Computers and Information Sciences
Ain Shams University
Cairo, Egypt

ABSTRACT
Temporal databases added a new dimension to traditional transaction databases. This dimension is the life time of each item, i.e. exhibition period, starting from the partition when this item appears in the transaction database to the partition when this item no longer exists. Mining temporal association rules became very interesting topic in many applications nowadays. In this paper, an efficient technique is proposed for indexing temporal databases in order to facilitate support counting process during mining operation. Some experiments were conducted using well-known real datasets to show the performance of the proposed indexing technique with respect to index size and running time of the mining algorithm. The results show that the proposed indexing technique saves a lot of running time and works efficiently with different databases characteristics.

Keywords
Indexing Temporal Databases, Apriori Algorithm, Temporal Association Rules (TAR).

1. INTRODUCTION
Extracting useful information from huge amount of data becomes very essential research topic these days. As the size and complexity of data increase, the need of efficient data analysis algorithms is very critical. Mining association rules is very interesting topic in the data mining and analysis fields. In its simplest form, it is interested in finding hidden relations between items in transactional databases, which were very useful in some applications like market analysis, decision making and business management [1].

Many algorithms were presented to solve the association rules mining from transactional databases. These algorithms adopted different approaches including: Level-wise Apriori and its modified versions, Partitioning and FP-Tree and FP-Growth algorithm [2, 3, 4].

The transactional databases become more complicated with extra information added on which required different set of algorithms to deal with the new information. For example, frequent weighted itemsets mining topic appeared when items in transactional database have different weights based on their significance [5]. Hence, a broad range of algorithms proposed to solve the problem of finding frequent weighted itemsets.

The research work in this paper is interested in working with temporal databases with extra temporal dimension added on. Temporal mining becomes an important topic in many areas such as weather forecasting, economics and communications [6]. In temporal database, each item has life time period, starting from the partition when this item appears in the transaction database to the partition when this item no longer exists. This life time period is called Exhibition Period which could be different from one item to another based on its availability in the database [7].

The main objective of this paper is to propose a new data structure for indexing temporal databases for fast support counting. The first proposed index, called TIndex, has one drawback which is the extra memory space needed to store the index. This drawback is solved in its modified version, called TIndex2, which requires less memory than the older version. The experimental results show that using the proposed indexing structure helps greatly in reducing the overall running time for mining frequent temporal itemsets.

The rest of the paper is organized as follows. Related studies in temporal association rules mining are discussed briefly in section 2. Section 3 includes the detailed steps of the proposed TIndex data structure with an illustrative example. Experimental results and discussion are presented in Section 4, and conclusions are finally drawn in Section 5.

2. RELATED WORK
There are different types of transactional databases. They differ on the extra information provided with each transaction. Simple, or Traditional, transactional database are simply list of transactions, where each transaction holds group of items with no extra details. Temporal databases included extra information about the life time of this transaction in the database. Many algorithms were proposed to deal with the mining process of temporal association rules (TAR) in general or incremental temporal databases.

Progressive Partition Miner (PPM) algorithm was proposed by Lee et al. [8, 9] to discover all the frequent temporal itemsets. The algorithm was designed to work on static databases, but it could be utilized to work with incremental databases. Firstly, the input database is partitioned into some parts based on time granularity. Secondly, the algorithm scans all these parts one by one and accumulates the level-2 candidate itemsets. Then using scan reduction technique, all the candidates are generated level by level and stored in the memory to be pruned. After generating all the candidates, the algorithm scans the databases transactions (sequentially) to calculate the support of each candidate and remove the infrequent ones. The experiments were conducted to compare PPM algorithm running time with Apriori+ (modified Apriori for temporal mining [2]) on synthetic datasets. The results showed a big performance gap difference between both algorithms especially in low minimum support values because Aprioi is very basic algorithm with no pruning or optimization techniques

Chang et al. [7] proposed Segmented Progressive Filter (SPF) algorithm for mining frequent temporal itemsets. It also can be utilized to handle incremental databases. The input database is partitioned into some parts based on time granularity and parts with similar items exhibition periods are processed together as segments. The level-2 candidates from each segment are generated and merged together to obtain the final level-2 candidate itemsets. Again, using scan reduction technique, all the candidates are generated level by level and stored in the memory to be pruned. After generating all the candidates, the algorithm scans the databases transactions (sequentially) to calculate the support of each candidate and remove the infrequent ones (similar to PPM algorithm). The authors compared the running time of SPF algorithm with Apriori^{IP} (modified version of Apriori for temporal mining [2]) using one synthetic dataset. The results show that SPF overcomes Apriori^{IP} especially in low minimum support values. SPF should improve the performance slightly than PPM (although no comparison found) due to segmenting the database parts based on shared items exhibition periods.

Huang et al. [10] presented Twain algorithm for mining frequent temporal itemsets mining. The proposed idea is much similar to PPM and SPF algorithms. Twain algorithm starts by processing the database parts one by one. In each part, the level-2 candidates are found and checked against minimum support threshold. The frequent ones are added directly to output frequent itemsets. Again, starting from level-2 candidate itemsets, the algorithm uses scan reduction technique to generate the candidate itemsets in all levels and stores them in the memory. After that, the algorithm scans the database transactions (sequentially) to calculate the support of each candidate (starting from level-3 candidate itemsets) and remove the infrequent ones (similar to PPM and SPF algorithm). Some experiments were conducted to compare the running time of Twain algorithm with SPF and Apriori^{IP}. The results show slight enhancement in running time when compared to SFP algorithm.

These algorithms have two main problems. The first problem is using scan reduction technique which is not efficient in case of large number of candidates because all these candidates must fit in the memory at the same time. The second problem is in the support counting process which is performed sequentially (for each candidate, all the transactions are scanned). This is not efficient in case of very large or dense databases.

Discovered temporal association rules can be used in different data analysis based on required application analysis. As an example, Linag et al. [11] modified Apriori algorithm to work with temporal databases and proposed a new algorithm called T-Apriori. After that, they used their proposed algorithm in discovering temporal association rules from the red tide monitoring data in Dapeng bay. Recently, Khairudin et al. [12] utilized temporal association rules mining in web log data. They investigated the effect of adding temporal information into mining operation and its effect on output association rules rather than traditional association rules. Their experimental results showed that temporal association rules mining outputs smaller number of rules rather than Apriori and FP-Growth algorithms. Also, the results showed that the generated rules have better quality and more meaningful than traditional ones.

3. INDEXING TEMPORAL DATABASES

In this section, TIndex data structure is presented for indexing temporal databases and its drawbacks. Then the new TIndex2 data structure is introduced that solves some of these drawbacks.

3.1 TIndex Data Structure

A new Tree-based data structure, called TIndex, is proposed for indexing transactions in temporal database. The TIndex improved Trie data structure to include temporal information of each item.

The TIndex contains two main components: Tree and Header Table. The Tree structure stores the transactions in the database and Header Table holds the information about each item in the database and a link to its first node in the tree. Each node in the tree is a tuple <item, part, support>, where *item* is item name, *part* is current part number and *support* is item frequency in this part. In the final TIndex, the transaction is mapped into a path from root to leaf and each node contains one item from this transaction and its support in this path. For example, consider the temporal database shown in Table 1 with two parts P_1 and P_2.

Table 1. Example Temporal Database

	TID	Transaction					
P_1	1		B		D		
	2		B	C	D		
	3		B	C			
	4	A			D		
P_2	5		B	C		E	
	6				D	E	
	7	A	B	C			
	8			C	D	E	
P_3	9		B	C		E	F
	10		B				F
	11	A			D		
	12		B		D		F

Starting from transaction TID-1, a new node <B, 1, 1> is added as child node to root and another new node <D, 1, 1> as child to node B as shown in Figure 1-a. For TID-2, the root already has child node <B, 1> so its support is incremented to be 2. Then add new child node <C, 1, 1> as node B has no matching child. For last item D, add a new node <D, 1, 1> as child to node C as in Figure 1-b. For TID-3, just increment support of node <B, 1> and its child <C, 1> as they already in the same path. For last transaction in part P_1, TID-4, root has no child labeled <A, 1>, so a new node <A, 1, 1> is added as child to root node. Then add new node <D, 1, 1> as child to <A, 1>. Figure 1-c shows the final TIndex after indexing all transactions in part P_1.

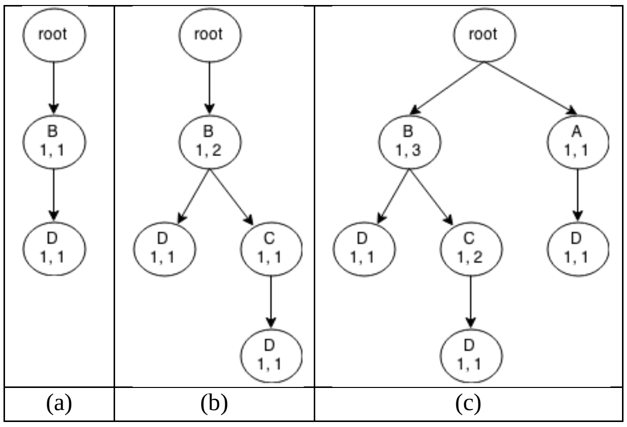

Fig. 1: TIndex for Part P_1 in Example Database.

The same procedure is applied to parts P_2 and P_3, but change current part to be 2 and 3 respectively. For transaction TID-5, a new node <B, 2, 1> is added as child to root node. A new node is added to the root node because it is not matched with node <B, 1> as it is from different part. After adding all transactions in the example database, the final TIndex for is shown in Figure 2. Note that there is a connection, dashed lines, between nodes that have same item to facilitate traversing in tree for fast support counting.

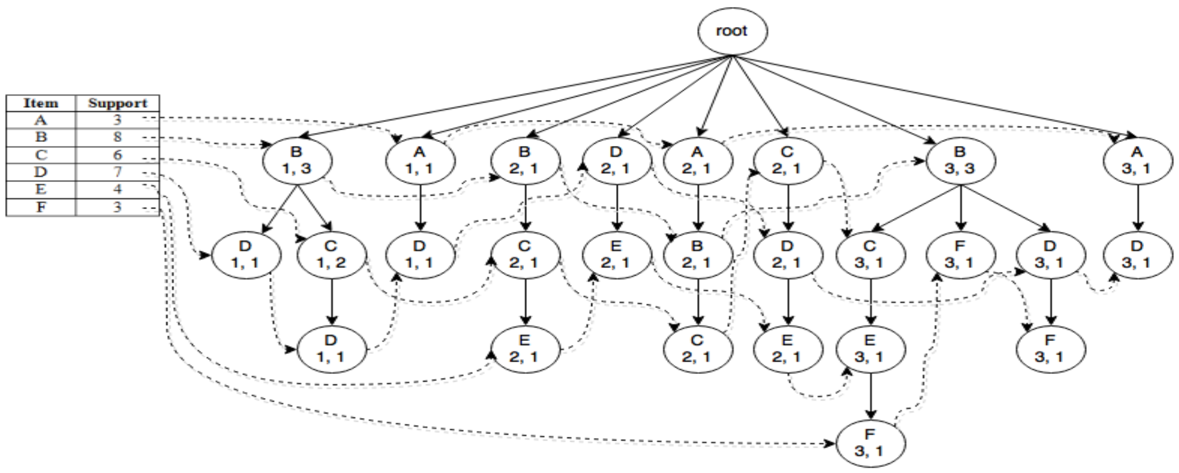

Fig. 2: Complete TIndex for the Example Database in Table 1

In order to calculate the support of given itemset, start searching the Header table with first item to determine the starting nodes. Then search within each subtree about the remaining items in this itemset. For example, consider the support of itemset {B D} to be calculated. First, Header table is examined to get the first node for item {B} that is <B, 1, 3> and consider it root of first subtree to search for item {D}. It is found in two paths (B-D) and (B-C-D) each with support 1. Then use next link to find next node for item {B} that is <B, 2, 1>, but item {D} cannot be found in this new subtree. Then use next link to find next node for item {B} that is <B, 3, 3> and found {D] in one path (B-D-F). The final support is calculated by adding the support of item {D}, last item in given itemset, in all the found paths. In this case the support of itemset {B D} is 3 because {D} has support 1 in each path of the three paths found.

The main objective of the proposed TIndex is to minimize search space when calculating support for candidate itemsets. As noticed in previous example, rather than searching all database transactions for itemset {B D}. The search space is reduced to only 3 transactions, which allows fast support counting especially in large and/or condensed databases. One drawback of the proposed TIndex is that it requires extra memory space to store the index tree and header table.

3.2 TIndex2 Data Structure

TIndex has a major drawback in storing transactions that share same prefix. If these transactions are in different parts, then a new branch will be created for each part. This will duplicate many nodes especially in dense databases that will result in extra memory space. For example, there are 4 transactions with prefix {B-C}. In TIndex tree shown in Figure 2, there are three different branches for these transactions because they are distributed over the three parts of the example database.

To solve this issue, TIndex2 is proposed to handle common prefix transactions in different parts. Some modifications occurred over the proposed TIndex data structure. First, the node in TIndex2 can store the support of each item in different parts by adding list of parts in each node. For example, node {B} will contain list of its support in the different parts as (1, 3), (2, 1) and (3, 3). The complete TIndex2 is shown in Figure 3.

As shown in Figure 3, TIndex2 is much more condensed than TIndex because it combines shared prefixes from all parts rather than create separate branch for each part. This will reduce the overall required memory space for the final index especially in the dense databases because many transactions have common prefix.

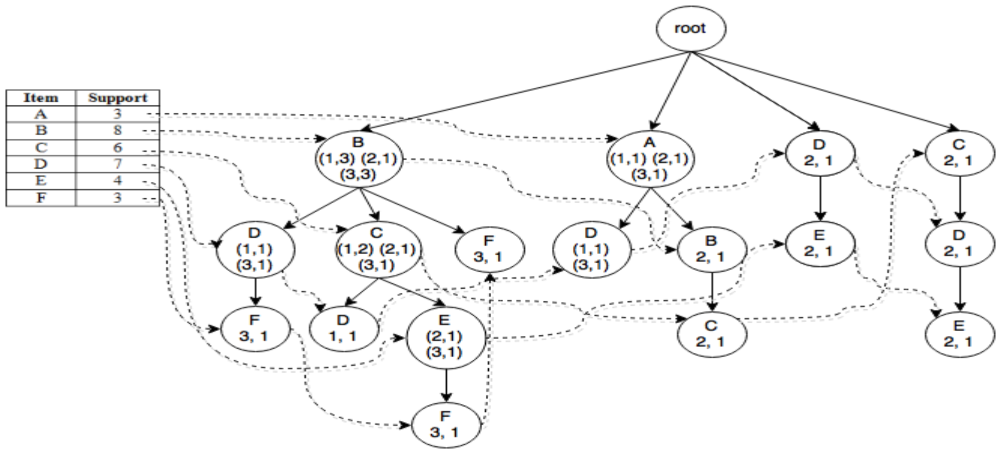

Fig. 3. Complete TIndex2 for the Example Database in Table 1

4. EXPERIMENTAL RESULTS

Some experiments were conducted to evaluate the performance of TIndex and TIndex2 data structures. Four real datasets were used with different sizes and characteristics varying from small to very large databases and for both low-density and high-density databases. First the required memory space is measured for each index to show the reduction ratio in each database. Then Apriori algorithm [2] is used in the mining operation to compare the required running time in case of using and not using the indexing technique.

4.1 Datasets

Four real databases (Chess, Mushroom, Retail and Accidents) obtained from frequent itemset mining, data set repository [13] are used in the experiments. Table 2 shows the statistical information for the different databases used in the experiments. For each database, it shows the number of transactions (#Trans), number of distinct items (#Items) and average number of items per transaction (TransSize). This information gives a clear view about the density of the database. For example, Retail database has 57 distinct items with average transaction size of 13 items. This means that Retail dataset will produce large number of candidates while most of them will not be frequent because they are distributed over the dataset with low density.

Table 2. Databases Information [13]

Dataset	#Trans (DB)	#Trans (db)	#Items	TransSize
Chess	2,696	500	75	37
Mushroom	7,125	1000	119	23
Retail	80,000	8,163	57	13
Accidents	306,183	34,000	468	33.8

4.2 Memory Usage

As illustrated earlier in section 3, TIndex and TIndex2 indexing techniques should reduce the required memory for storing the temporal database. In this experiment, the required memory space for each dataset is measured. The corresponding TIndex and Tindex2 sizes are shown in Table 3 and Table 4 respectively.

Table 3. Memory Usage Analysis - TIndex

Dataset	Dataset size (MB)	TIndex size (MB)	Reduction Ratio (%)
Chess	18.443	10.465	43.3
Mushroom	9.169	6.887	24.9
Retail	92.372	64.284	30.4
Accidents	677.814	511.631	24.5

As shown in Table 3, TIndex reduces the required memory to store each database because transactions with shared prefix have the same path. The reduction ratio is affected by database characteristics. For example, Chess dataset is very dense and most of its transactions share the same prefix, so TIndex achieves 43.3% reduction ratio. On the other hand, Accidents dataset is sparse one, so TIndex achieves only 24.5% reduction ratio in this case.

As shown in Table 4, TIndex2 achieves better reduction ratio than TIndex because it combines the transactions with shared prefix from all parts. This works efficiently with dense databases such as Chess and Retail as TIndex2 achieves 60.2% and 46.3% reduction ratio respectively. In case of Mushroom and Accidents datasets, TIndex2 requires only about one third of the overall dataset size.

Table 4. Memory Usage Analysis – TIndex2

Dataset	Dataset size (MB)	TIndex2 size (MB)	Reduction Ratio (%)
Chess	18.443	7.331	60.2
Mushroom	9.169	5.696	37.9
Retail	92.372	49.627	46.3
Accidents	677.814	435.398	35.8

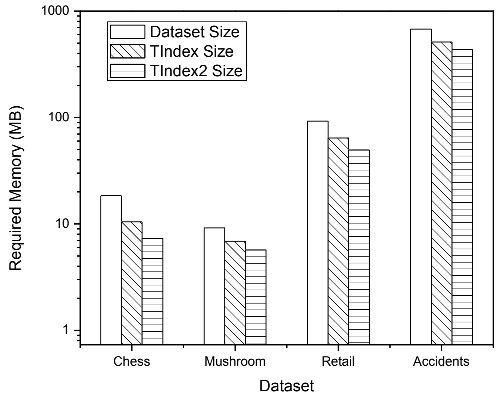

Fig. 4. Memory Space for TIndex and TIndex2

Figure 4 shows overall comparison between TIndex and TIndex2 required memory. It is clear that TIndex2 saves extra memory space than TIndex and also reduce the required memory to store input temporal databases.

4.3 Running Time

The objective of this experiment is to show the performance of the proposed TIndex2 indexing technique in the mining operation. Traditional Apriori algorithm is used and a modified version of Apriori, called Apriori-TIndex, that is used TIndex data structure in support counting process. Figures 5-8 show the measured running time with different minsupp threshold values in the four real datasets.

Chess dataset is very dense dataset that generates huge number of candidates that requires most of the running time to calculate the support of these candidates. As shown in Figure 5, using TIndex with Apriori algorithm reduces the overall running time and achieves about 2.86 speed-up ratio. The speed-up ration is not large because building TIndex for such database adds extra running time before mining algorithm begins working.

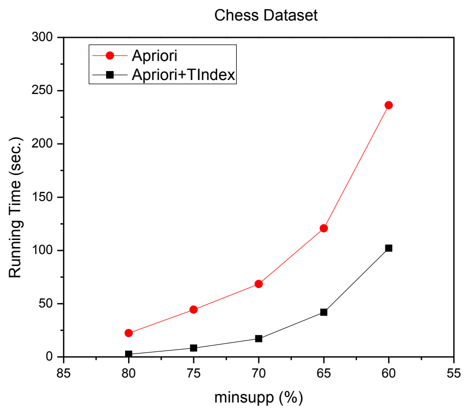

Fig. 5. Running Time Comparison – Chess Dataset

As shown in Figure 6, the running time of Apriori+TIndex algorithm is very low compared to Apriori algorithm alone. At low minsupp values, Apriori generates huge number of candidates that requires extra running time to calculate their support and remove infrequent ones. For example, at 30% minsupp, Apriori algorithm needs about 90 seconds, while adding TIndex requires only 15 seconds, which achieves speed-up ratio about 6 times.

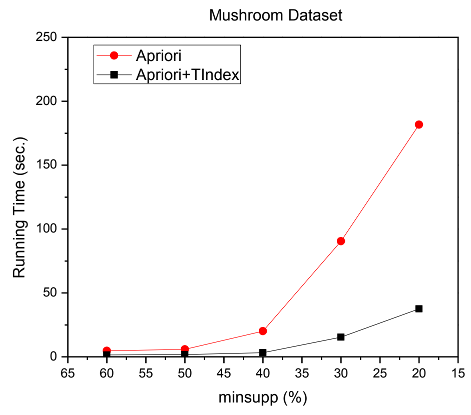

Fig. 6. Running Time Comparison – Mushroom Dataset

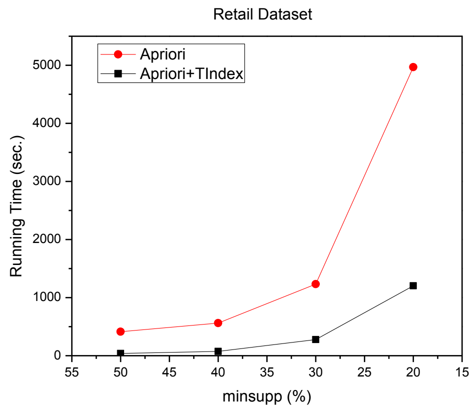

Fig. 7. Running Time Comparison – Retail Dataset

In Figure 7, TIndex works efficiently with Apriori algorithm because there are many numbers of candidates that requires extra running time for support counting. This extra running time covers the required time for building TIndex because dataset is also dense one. This allows Apriori+TIndex algorithm to achieve about 4.5 speed-up ratio on average than traditional Apriori algorithm alone.

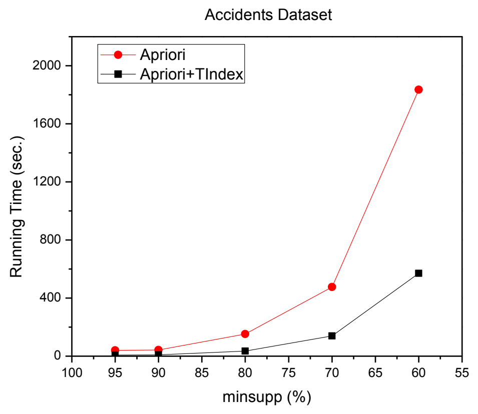

Fig. 8. Running Time Comparison – Accidents Dataset

Accidents dataset is very sparse, which produces small number of candidates. On the other hand, the dataset size is large, which needs extra time when calculating support of these candidates. As shown in Figure 8, using TIndex improves the overall mining running time by one third on average.

5. CONCLUSION

In this paper, the problem of mining temporal association rules is addressed. The problem mainly interested in finding frequent temporal itemsets in temporal databases. A new indexing technique, called TIndex, is proposed for indexing temporal database. The proposed TIndex data structure suffers from high memory usage problem because it separates transactions from each part. To solve this problem, TIndex2 data structure is proposed. It combines the transactions from different parts as they share same prefix items. The experimental results on real datasets show that TIndex2 requires less memory than TIndex that achieves better reduction ratio.

To check the effect of the proposed indexing technique in the mining process, the running time of Apriori algorithm alone is compared with the modified one that includes TIndex data structure. The experimental results show that using the proposed indexing technique achieves great performance in the mining process because support counting of candidate itemsets is the bottleneck of the mining algorithm which requires the big share of overall running time.

6. REFERENCES

[1] Q. Zhao and S. Bhowmick, “Association Rule Mining: A Survey,” Technical Report, Center for Advanced Information Systems (CAIS), Nanyang Technological University, Singapore, 2003.

[2] R. Agrawal and R. Srikant, “Fast algorithms for mining association rules,” in Proc. Int. Conf. Very Large Data Bases VLDB’94, 1994, pp. 487–499.

[3] J. Han, J. Pei, Y. Yin and R. Mao, “Mining frequent patterns without candidate generation: a frequent-pattern tree approach,” Data Mining and Knowledge Discovery, vol. 8, no. 1, 2004, pp. 53–87.

[4] G. Grahne and J. Zhu, “Fast algorithms for frequent itemset mining using FP-trees,” IEEE Trans. on Knowledge and Data Eng., vol. 17, no. 10, 2005, pp. 1347–1362.

[5] B. Vo, F. Coenen and B. Le, “A new method for mining Frequent Weighted Itemsets based on WIT-trees,” Expert Systems with Applications, vol. 40, no. 4, 2013, pp.1256-1264.

[6] H. Ning, H.Yuan and S. Chen, “Temporal association rules in mining method,” The 1st International Multi-Symposiums on Computer and Computational Sciences (IMSCCS'06), Zhejiang, China, vol. 2, 2006, pp. 739–742.

[7] C-Y. Chang, M-S. Chen and C-H. Lee, “Mining general temporal association rules for items with different exhibition periods,” in Proc. IEEE Int. Conf. Data Mining, 2002, pp. 59–66.

[8] C-H. Lee, C-R. Lin and M-S. Chen, “On Mining General Temporal Association Rules in a Publication Database,” in Proc. IEEE Int. Conf. Data Mining, 2001, pp. 487–499.

[9] C-H. Lee, M-S. Chen and C-R. Lin, “Progressive partition miner an efficient algorithm for mining general temporal association rules,” IEEE Trans. Knowledge and Data Eng., vol. 15, no. 4, 2003, pp. 1004–1017.

[10] J-W. Huang, B-R. Dai and M-S. Chen, "Twain Two-end association miner with precise frequent exhibition periods," ACM Trans. on Knowledge Discovery from Data (TKDD), vo. 1, no. 2, 2007.

[11] Z. Linag, T. Ximming, L. Lin and J. Wenliang, "Temporal Association Rule Mining Based on T-Apriori Algorithm and its Typical Application," International Symposium on Spatio-temporal Modeling, Spatial Reasoning, Analysis, Data Mining and Data Fusion , 2005

[12] N. Khairudin, A. Mustapha and M. Ahmad, "Effect of Temporal Relationships in Associative Rule Mining for Web Log Data," The Scientific World Journal, vol. 2014, Article ID 813983, 10 pages, 2014.

[13] Frequent Itemset Mining Dataset Repository (Accessed 1 Spetember 2015). Available online at: http://fimi.ua.ac.be/data/