International Journal of Computer Information Systems and Industrial Management Applications.
ISSN 2150-7988 Volume 9 (2017) pp. 103-113
© MIR Labs, www.mirlabs.net/ijcisim/index.html

IndxTAR: An Efficient Algorithm for Indexed Mining of Incremental Temporal Association Rules

Mohammed M. Fouad[1] and Mostafa G.M. Mostafa[2]

[1] Faculty of Computing and Information Technology, King Abdulaziz University,
Jeddah, Saudi Arabia
mmfouad@kau.edu.sa

[2] Faculty of Computers and Information Sciences, Ain Shams University,
Cairo, Egypt
mgmostafa@cis.asu.edu.eg

***Abstract*: Mining temporal association rules is a very interesting topic, which is applied in many applications nowadays. In temporal databases, each item has its own lifetime period, called exhibition period, which is different from other items. With the rapid increase of databases and new transactions added, the incremental mining is introduced to solve the problem of maintaining association rules in updated databases. The existing algorithms did not utilize the previously discovered rules when the database is updated .This paper presents an efficient algorithm for mining incremental temporal association rules. The proposed algorithm, called IndxTAR, utilizes two major components: a relatively new data structure and previously discovered frequent temporal itemsets to improve the performance of mining the incremental temporal association rules. Experiments on both real and synthetic datasets are conducted to compare IndxTAR algorithm performance with recently cited incremental temporal mining algorithms. The results show that IndxTAR algorithm overcomes other algorithms by many orders of magnitude and can efficiently process large databases with linear scalability.**

***Keywords*:** Temporal Association Rules, Incremental Mining, Database Indexing.

I. Introduction

Extracting useful information from huge amount of data becomes very essential research topic in these days. As the size and complexity of data increase, the need of efficient data analysis algorithms is very critical. Association Rules Mining (ARM) is very interesting topic in data mining. In its simplest form, ARM is interested in finding hidden relations between items in transactional databases, which are very useful in some applications like market analysis, decision-making and business management [1, 2].

Many algorithms were presented to solve the association rules mining for transactional databases including level-wise Apriori and its variations [3], partitioning and FP-Tree based algorithms [4, 5].

The transactional databases become more complicated with extra information added on which required a different set of algorithms to deal with the new information. For example, frequent weighted itemsets mining topic appeared when items in a transactional database have different weights based on their significance [6]. Hence, a broad range of algorithms is proposed to solve the problem of finding frequent weighted itemsets.

In this paper, we are interested in working with temporal databases that have extra temporal dimension added to each transaction. Issues of temporal databases mining are important in many applications such as weather forecasting, economics and communications [7]. Temporal databases are extensions of the traditional transactional databases. Each database is split into parts according to the time granularity based on the application data. For example, publications temporal database is split into several parts where each part contains the papers that were published in certain date/volume.

In temporal databases, each item has lifetime interval, starting from the partition when this item appears in the transactional database to the partition when this item no longer exists. This lifetime period is called Exhibition Period, which could be different from one item to another based on its availability in the database [8].

As temporal databases are always increasing, incremental mining of temporal association rules became popular research problem. Since adding new transactions to the original database may invalidate some existing rules and generate new rules, some algorithms were proposed to solve the problem of mining incremental temporal association rules [9, 10, 11]. As discussed later, the main problem of these algorithms is they did not utilize the previously discovered frequent itemsets from the original database. They mainly tend to re-discover all frequent itemsets from the updated database [12, 13, 14].

The main objective of this paper is to propose IndxTAR algorithm for efficiently mining incremental temporal association rules. The proposed algorithm utilized previously discovered frequent itemsets in the mining process. In addition, we proposed a new indexing approach for indexing incremental temporal databases for fast support counting.

The contributions of this work are summarized as follows:

1- A new algorithm for mining incremental temporal association rules (IndxTAR) is proposed for mining frequent itemsets in large updated temporal databases.
2- New data structure is introduced, TIndex, for indexing temporal transactional databases for fast support counting.
3- Minimum disk I/O requests by scanning incremental part only once and utilizing previously discovered frequent itemsets from original database.
4- Using TIndex to early prune infrequent candidates reduces the number of generated candidates over scan reduction technique that is implied in recently cited TAR algorithms.
5- Many experiments were conducted on both real and synthetic datasets to compare the performance of the proposed algorithm with recently cited algorithms. Experimental results show a significant performance improvement in running time with slight memory increase due to index size. In addition, the experiment shows that IndxTAR algorithm is linearly scalable to run on large temporal databases even at low minimum support values.

The rest of the paper is organized as follows. Related studies in temporal association rules mining are discussed briefly in section II. Section III includes the detailed steps of the proposed IndxTAR algorithm with an illustrative example. Experimental results and discussion are presented in Section IV, and conclusions are finally drawn in Section V.

II. Related Work

There are different types of transactional databases. They differ from the extra information provided with each transaction. Simple, or Traditional, transactional database is simply a list of transactions, where each transaction holds a group of items with no extra details. Temporal databases included extra information about the lifetime of this transaction in the database. In this case, some transactions may be invalid after their lifetime range ends while new transactions are added to the database with new lifetime range in incremental parts. Many algorithms were proposed to deal with the mining process of temporal association rules (TAR) in general or incremental temporal databases [15, 16].

Lee et al. [17, 18] proposed the Progressive Partition Miner (PPM) algorithm 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) 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

Change et al. [8] 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 re-move the infrequent ones (similar to PPM algorithm). The authors compared the running time of SPF algorithm with AprioriIP (modified version of Apriori for temporal mining) using one synthetic dataset. The results show that SPF overcomes AprioriIP 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. [9] presented Twain algorithm for mining frequent temporal itemsets. 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 algorithm discovers all level-2 candidates and checks them against global 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 re-move the infrequent ones (similar to PPM and SPF algorithm). Some experiments were conducted to compare the running time of Twain algorithm with SPF and AprioriIP. The results show slight enhancement in running time when compared to SFP algorithm.

Gharib et al. [10] proposed Incremental Temporal Association Rules Mining (ITARM) algorithm for mining frequent temporal itemsets from incremental databases. The algorithms starts with the level-2 candidates generated from the previous mining process on original database. The first step is to scan the incremental database to find the level-2 candidates in it, and then use Sliding Window Filtering (SWF) technique to merge both lists to obtain the final level-2 candidate itemsets. The algorithm then uses scan reduction technique to generate the candidate item-sets in all levels and stores them in the memory. Finally, the algorithm scans the updated database transactions (sequentially) to calculate the support of each candidate and remove the infrequent ones (similar to the previous algorithms). The experiments were conducted to compare the running time of ITARM algorithm with SPF and Twain algorithms. The results showed small performance improvement of ITARM algorithm against other algorithms.

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.

Temporal FP-Trees (TFP-Trees) is proposed firstly by Jin et al. [19] to discover frequent temporal itemsets. They simply added temporal information to existing FP-tree and used FP-Growth algorithm [4] in the mining process. Later, Dafa-Alla et al. [11] proposed Incremental Mining of General Temporal Association Rules (IMTAR) algorithm. They utilized TFP-Apriori tree proposed to be used in incremental temporal mining with extra pruning technique applied. In their experiments, they did not compare IMTAR algorithm with any incremental temporal mining algorithm, only show the running time for both datasets along with min. support value. They also compared the number of frequent patterns generated by IMTAR and FP-Growth algorithms which seem to be the same except they added extra pruning technique that made IMTAR had less number than FP-Growth. The main drawback of IMTAR algorithm is it did not utilize the previously discovered frequent temporal itemsets.

Discovered temporal association rules can be used in different data analysis based on required application analysis. As an example, Linag et al. [20] 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, Nazli et al. [21] 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. In addition, the results showed that the generated rules have better quality and more meaningful than traditional ones.

III. Mining Incremental Temporal Association Rules

In this section, we present the proposed Indexed Temporal Association Rules Mining (IndxTAR) algorithm for efficient mining of incremental temporal association rules. We propose a new data structure, TIndex, for indexing transactions in temporal databases. IndxTAR algorithm uses the proposed TIndex in calculating the support of the generated candidate itemsets in an efficient way. Subsection A presents TIndex data structure with detailed steps. Later in this section, the proposed IndxTAR algorithm is illustrated with a detailed example.

A. TIndex Data Structure

We propose a new tree-based data structure, called TIndex, for indexing transactions in temporal databases. 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 the 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 the item name, *part* is the current part number and *support* is the item frequency in this part. For example, in Figure 1-c, node <B, 1, 3> means that item B appeared 3 times in part 1 of the example database shown in Table 1. In the final TIndex, the transaction is mapped into a path from the root node to a leaf and each node contains one item from this transaction and its support in this path.

The proposed TIndex has the same structure of the FP-tree (i.e. Tree structure and Header table); however, TIndex is designed to index temporal databases for fast support counting. The main differences between both structures are:

1- FP-tree works only on traditional transactional databases without any support for temporal databases. In other words, FP-tree stores only the frequency of each item but not the temporal information such as exhibition period for each item.

2- FP-tree stores only frequent items in each transaction. Therefore it cannot be used to index the incremental database, because some itemsets may be infrequent in the original database (will not be included in the FP-tree) but after the increment, they became frequent ones.

Table 1. Example Temporal Database

TID		Transaction
Part P_1	1	B D
	2	B C D
	3	B C
	4	A D
Part P_2	5	B C E
	6	D E
	7	A B C
	8	C D E

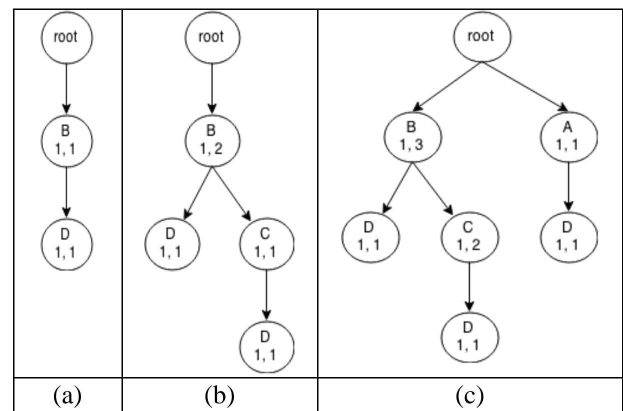

Figure 1. TIndex for part P_1 in the example database shown in Table 1.

For example, consider the temporal database shown in Table 1 with two parts P_1 and P_2. Starting from transaction TID-1, a new node <B, 1, 1> is added as a child node to root and another new node <D, 1, 1> as a child to node B as shown in Figure 1-a. For TID-2, the root already has child node <B, 1> so we just increase the support of it to be 2. Then add new child node <C, 1, 1> as node B has no matching child. For the last item D, add a new node <D, 1, 1> as a child to node C as in

Figure 1-b. For TID-3, just increase support of node <B, 1> and its child <C, 1> as they already in the same path. For the 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 a child to root node. Then add new node <D, 1, 1> as a child to <A, 1>. Figure 1-c shows the final TIndex after indexing all transactions in part P_1.

The same steps are applied to part P_2, but change current part to be 2. For transaction TID-5, a new node <B, 2, 1> is added as a child to root node. We add new node because it is not matched with node <B, 1> as it has a different part. After adding all transactions in part P_2, the final TIndex for the example database is shown in Figure 2. Note that there is a connection, represented by dashed lines, between some nodes. These nodes have the same item to facilitate traversing in the tree for fast support counting.

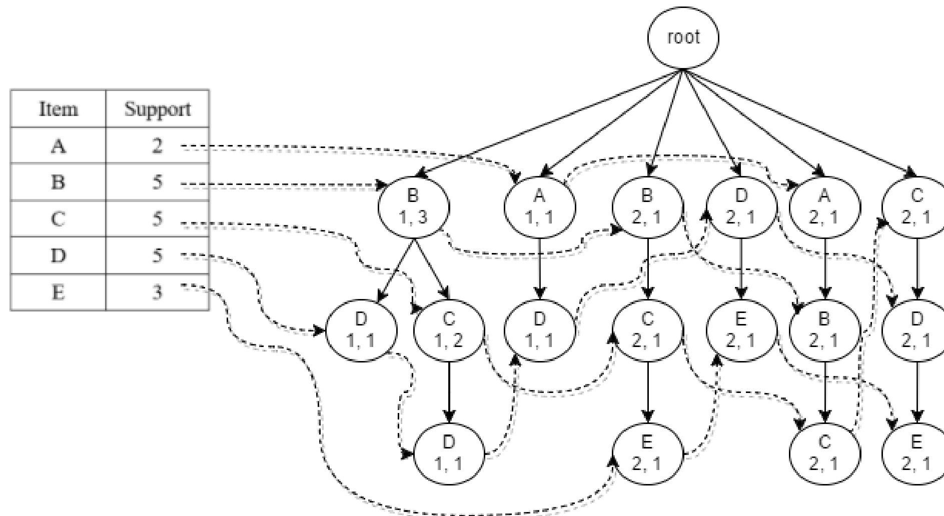

Figure 2. The complete TIndex for the example temporal database in Table 1.

In order to calculate the support of a given itemset, we start from the header table with the first item in the itemset to determine the starting nodes. Then search within each subtree about the remaining items in this itemset. For example, if we need to calculate the support of itemset {B D}. First, we examine Header table to get the first node for item {B} that is <B, 1, 3> and consider it as a root of the first subtree to search for item {D}. We found it 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 we cannot find item {D} in this new subtree. 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 2 because there are only two paths where {D} has support 1 in each path.

The main objective of the proposed TIndex is to minimize search space when calculating support for candidate itemsets. As we noticed in the previous example, rather than searching all the transactions for itemset {B D}, the search space is reduced to only 3 transactions. This proves that the proposed TIndex allows fast support counting, especially in the large and/or condensed databases.

B. The Proposed IndxTAR Algorithm

In this section, we describe a detailed example of the proposed IndxTAR algorithm for incremental mining of temporal databases. The symbols used in the proposed algorithm are listed in Table 2.

The main idea of the proposed algorithm is to utilize the frequent itemsets that were previously discovered from the original database (DB). This could be achieved by considering only level-2 frequent itemsets from the original database as initial candidates. Then, these candidates are updated with new candidates from the incremental database to form the initial candidates for the updated database. In this case, we only scan incremental database once and did not scan original database again at all. The proposed TIndex data structure is used then for indexing the updated database for fast support counting as discussed in subsection A.

Table 2. The Used Symbols in IndxTAR Algorithm

Symbol	Meaning
DB	Original database with /*DB*/ transactions
db	Incremental database with |*db*| new transactions
DB+db	Updated database, i.e. *DB* U *db*
minsupp	Minimum support threshold value
P	Number of parts of updated database
F_k^{DB}	Frequent Level-k itemsets in database *DB*
C_k^{DB}	Candidate itemsets in database *DB* with k items
MCP(X)	Maximal Common exhibition Period of itemset *X*
X.start	The starting part for itemset *X*
X.end	The last part for itemset *X*
X.Supp	Support of itemset *X*
SX	The set of all subsets of given itemset *X*
F_{DB+db}	Final frequent itemsets in updated database

The input of IndxTAR algorithm includes the original and incremental databases (divided into *P* parts), the frequent level-2 itemsets from original database and the minimum support threshold. The output of the algorithm is the final frequent itemsets in the updated database. The detailed steps of the algorithm are illustrated in Figure 3.

The proposed algorithm has five main steps as follows:

- Step 1: Build TIndex of the updated database (*DB+db*). This index will be used later in calculating the support of the generated candidates.

- Step 2: Generate the initial level-2 candidates for updated database by merging level-2 frequent itemsets from the original database and level-2 candidates from the incremental part.
- Step 3: Filter initial level-2 candidates to find level-2 frequent itemsets by removing the ones with relative support lower than minimum support threshold.
- Step 4: Use Apriori join operator to generate all frequent itemsets starting from level-3, and use TIndex to calculate support of generated candidates at each level to remove infrequent ones before advancing to next level.
- Step 5: Generate level-1 frequent itemsets by generating all subsets of level-2 frequent itemsets and check the support of each one against minimum support threshold.

C. Illustrative Example

Consider the temporal database shown in Table 3. The original database contains 2 parts (P_1 and P_2) and last part (P_3) is the increment part. The algorithm input includes also the frequent level-2 itemsets from original database, F_2^{DB}, generated at ***30%*** minimum support value as shown in Table 4. For each itemset, the start part, the end part and the occurrences count are mentioned along the algorithm steps.

The algorithm starts by building TIndex for the updated database as shown in Figure 4. The header table and inter-connections between nodes are removed and only the tree is shown for better viewing. For step 2, the algorithm scans the incremental part to find candidate level-2 itemsets (C_2^{db}) as shown in Table 5. Both lists are merged together to form the initial candidate level-2 itemsets for the updated database (C_2^{DB+db}) as shown in Table 6.

The relative support of each candidate is calculated based on the updated interval of the merged candidates. For example, itemsets {BC} is found in both lists, so its interval end is expanded to be 3 and its new support is the sum of both supports in original and incremental parts to be 4 + 1 = 5. The relative support of itemset {BC} is calculated now over the 3 parts in which {BC} interval include as to be 5 / 12 = 41.67%. Some itemsets could not be found in both lists, so their count stay as they are and relative support is calculated with respect to their interval.

Table 3. Incremental Temporal Databases (DB + db)

		TID	Transaction
Original Database (DB)	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
Increment (db)	P_3	9	B C E F
		10	B F
		11	A D
		12	B D F

Table 4. Original Frequent level-2 Itemsets F_2^{DB}.

Itemset	Start	End	Count
BC	1	2	4
CE	2	2	2
DE	2	2	2

Inputs:

Original database (DB) divided into (P-1) parts,
Incremental database (db),
Number of parts (P),
Frequent Level-2 itemsets in DB (F_2^{DB}),
Minimum support threshold (minsupp).

Output:

Frequent itemsets in updated database (F_{DB+db}).

Method:

IndxTAR Algorithm

1. *//build index for the updated database DB+db*
   TI = **Build_TIndex**($DB+db$)

2. *//Generate initial Candidate Level-2 itemsets in DB+db*
   $C_2^{DB+db} = F_2^{DB}$
   C_2^{db} = Candidate Level-2 itemsets in db
   Foreach itemset $X \in C_2^{db}$
   If $X \in F_2^{DB}$ then
   $X.end = P$
   $X.Supp = Support(X, C_2^{db}) + Support(X, F_2^{DB})$
   Else
   Add X to C_2^{DB+db}
   If $X.Supp \geq minsupp*|DB+db|_{(X.start,\ X.end)}$ then
   While $X.start \geq 1$
   $X.start = X.start - 1$
   $X.Supp$ = **Calculate_Support**(TI, X)
   If $X.Supp \geq minsupp*|DB+db|_{(X.start,\ X.end)}$ then
   Add X to F_2^{DB+db}
   Else
   Break

3. *//Filter initial C_2^{DB+db} to obtain F_2^{DB+db}*
   $F_{DB+db} = \emptyset$
   Foreach itemset $X \in C_2^{DB+db}$
   If $X.Supp \geq minsupp * |DB+db|_{(X.start,\ X.end)}$ then
   Add X to F_2^{DB+db}
   $F_{DB+db} = F_{DB+db} \cup F_2^{DB+db}$

4. *//Generate all frequent itemsets using Apriori*
   Let $k = 2$, $C_k^{DB+db} = F_2^{DB+db}$
   While($C_k^{DB+db} \neq \emptyset$)
   $C_{k+1}^{DB+db} = C_k^{DB+db} \times C_k^{DB+db}$ *//Apriori join operator*
   Foreach itemset $X \in C_{k+1}^{DB+db}$
   $X.Supp$ = **Calculate_Support**(TI, X)
   If $X.Supp < minsupp * |DB+db|_{(X.start,\ X.end)}$ then
   Remove X from C_{k+1}^{DB+db}
   $F_{DB+db} = F_{DB+db} \cup C_{k+1}^{DB+db}$
   $k = k + 1$

5. *//Generate Frequent Level-1 itemsets (F_1^{DB+db})*
   $F_1^{DB+db} = \emptyset$
   Foreach itemset $X \in F_2^{DB+db}$
   $SX = \{Z^{MCP(X)} \mid Z \subset X\}$
   $SX.Supp$ = **Calculate_Support**(TI, SX)
   Add SX to F_1^{DB+db}
   $F_{DB+db} = F_{DB+db} \cup F_1^{DB+db}$

Figure 3. The Proposed IndxTAR Algorithm

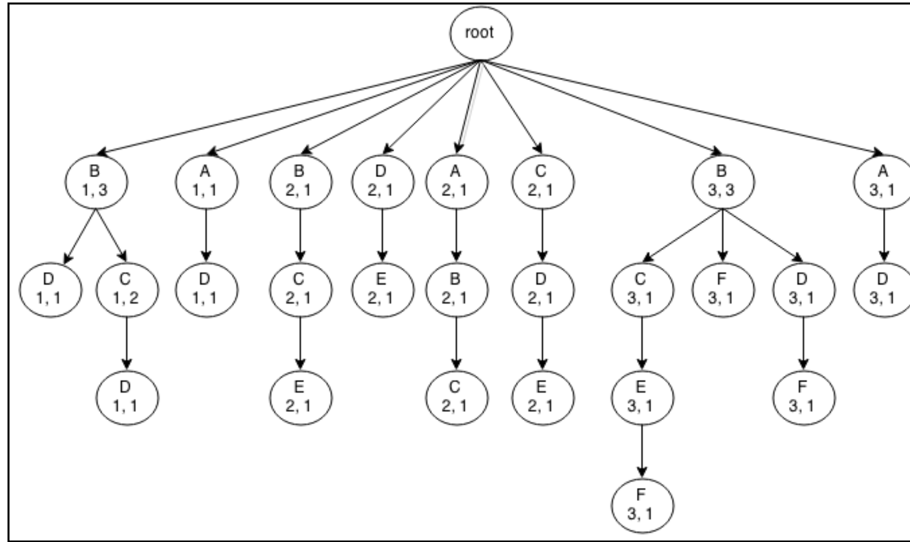

Figure 4. TIndex for the Updated Database (*DB+db*) in Table 3.

Some itemsets could be infrequent in the original database but may be frequent in the updated database. This could only happen if they are frequent in the incremental part. To overcome this issue, the support of the frequent level-2 itemsets in the incremental part is checked in the previous parts one by one. If they found frequent in one part they added to F_2^{DB+db} and proceeds to the previous part and so on. For example, itemset $\{BF^{3,3}\}$ is frequent in the incremental part, but not shown in F_2^{DB}. The algorithm checks its support in the previous part to check if it is frequent or not. The support of itemset $\{BF^{2,3}\}$ is calculated to be 3 / 8 = 38% which is frequent and added to F_2^{DB+db}. Then, check the support of itemset $\{BF^{1,3}\}$ which is found to be not frequent as its support is 3 / 12 = 25%.

Table 5. Candidate level-2 Itemsets C_2^{db} – Incremental Part.

Itemset	Start	End	Count
AD	3	3	1
BC	3	3	1
BD	3	3	1
BE	3	3	1
BF	3	3	3
CE	3	3	1
CF	3	3	1
DF	3	3	1
EF	3	3	1

Table 6. Generating level-2 Itemsets C_2^{DB+db}.

Itemset	Start	End	Count	Relative Support
AD	3	3	1	1 / 4 = 25%
BC	1	3	4 + 1 = 5	5 / 12 = 42%
BD	3	3	1	1 / 4 = 25%
BE	3	3	1	1 / 4 = 25%
BF	3	3	3	3 / 4 = 75%
CE	2	3	1 + 2 = 3	3 / 8 = 38%
CF	3	3	1	1 / 4 = 25%
DE	2	2	2	2 / 4 = 50%
DF	3	3	1	1 / 4 = 25%
EF	3	3	1	1 / 4 = 25%

For Step 3, the relative support of the candidate itemsets in C_2^{DB+db} is checked against the minimum support threshold (30%). Only itemsets that satisfies this threshold are considered frequent. In our example, only $\{BC^{1,3}\}$, $\{BF^{3,3}\}$, $\{BF^{2,3}\}$, $\{CE^{2,3}\}$ and $\{DE^{2,2}\}$ are frequent and added to F_2^{DB+db} list. This will be the initial level-2 list that will be used in step 4 to generated candidates for next levels.

Apriori join operator is used then to generate candidate itemsets starting from initial C_2^{DB+db} for next levels. The process goes as following: first generate level-k+1 candidates (C_{k+1}^{DB+db}) from level-k frequent itemsets, then filter these candidates to find frequent level-k+1 itemsets (F_{k+1}^{DB+db}) and remove infrequent ones. In our example, no more candidates will be produced because there is no common prefix and interval for any pair of in F_2^{DB+db} list.

Table 7. Frequent Itemsets in Updated Database F_{DB+db}.

Itemset	Count	Relative Support
$B^{1,3}$	8	8 / 12 = 67%
$C^{1,3}$	6	6 / 12 = 50%
$B^{3,3}$	3	3 / 4 = 75%
$F^{3,3}$	3	3 / 4 = 75%
$C^{2,3}$	4	4 / 8 = 50%
$E^{2,3}$	4	4 / 8 = 50%
$D^{2,2}$	2	2 / 4 = 50%
$E^{2,2}$	3	3 / 4 = 75%
$B^{2,3}$	5	5 / 8 = 63%
$F^{2,3}$	3	3 / 8 = 38%
$BC^{1,3}$	5	5 / 12 = 42%
$BF^{3,3}$	3	3 / 4 = 75%
$CE^{2,3}$	3	3 / 8 = 38%
$DE^{2,2}$	2	2 / 4 = 50%
$BF^{2,3}$	3	3 / 8 = 38%

In the final step, level-1 frequent itemsets F_1^{DB+db} are generated by finding all the subsets of each itemset in the level-2 frequent itemsets F_2^{DB+db}. The support of these itemsets is calculated using the TIndex. For example, frequent itemset $\{BC^{1,3}\}$ will produce two frequent level-1 itemsets $\{B^{1,3}\}$ with support 8 and $\{C^{1,3}\}$ with support 6. Table 7 shows

the final output frequent temporal itemsets for the example database shown in Table 3.

IV. Experimental Results

In this section, we will present some experiments that were conducted to evaluate the performance of IndxTAR algorithm. There are three main aspects to investigate; running time, average memory usage and number of generated candidates. The main objective of these experiments is to compare the performance of the proposed algorithm with recently incremental temporal mining algorithms; TWAIN [9] and ITARM [10]. 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. Also, three synthetic datasets are used to test the performance of IndxTAR algorithm when dealing with very large databases. All the experiments were conducted using a machine with Intel® Core™ i7-3770 CPU @ 3.40GHz and 8.00 GB memory, running 64-bit Windows 7 Enterprise Edition®.

A. Datasets

Four real databases (Chess, Mushroom, Retail and Accidents) are used in the experiments, which obtained from frequent itemset mining data set repository [22]. In addition, three more large synthetic databases are included with different sizes (100K, 200K, and 400K transactions) generated using the data generator IBM Almaden Quest research group. This generator is not available now on their website, but another implementation can be downloaded from [23]. These datasets are split into two parts as original and incremental parts to run the incremental mining algorithms over them.

The synthetic dataset generator used some parameters to maintain the characteristics of the generated dataset. For simplicity, we added some notation to dataset name to define its parameters. The generated dataset is named by (Syn-Tx,Nm,Ln) where, x is the number of transactions (in thousands), m is the number of distinct items (in thousands) and n is the number of maximal potentially frequent itemsets (in thousands).

Table 8. Database Information [22].

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
Syn-T100,N1,L10	90,000	4,664	1000	10.4
Syn-T200,N1,L10	220,000	18,193	1000	10.8
Syn-T400,N1,L10	360,000	32,518	1000	11.3

Table 8 shows the statistical information for the different databases used in the experiments. For each database, it shows the number of transactions (#Trans) in both original and incremental databases, the number of distinct items (#Items) and the average number of items per transaction (TransSize). This information gives a clear view of the density of the database. For example, Retail database has 57 distinct items with an average transaction size of 13 items. This means that Retail dataset will produce a large number of candidates while most of them will not be frequent because they are distributed over the dataset with low density.

B. Number of Candidates

TWAIN and ITARM algorithms use scan reduction technique in candidate generation phase. In this technique, all candidates from all levels are generated first then support of all these candidates is counted to remove infrequent ones. The main advantage of this technique is reducing database scans but the main disadvantage is it produces a huge number of candidates because there is no pruning at each level.

The proposed IndxTAR uses TIndex to count the support of the generated candidates. This enables pruning infrequent candidates at each level which reduce the total number of generated candidates. In this experiment, we counted the average number of generated candidates from ITARM and IndxTAR algorithms to compare the performance of both techniques as shown in Table 9.

Table 9. Average Number of Generated Candidates.

Dataset	ITARM	IndxTAR	Reduction Ratio (%)
Chess	196,947	132,787	32.6
Mushroom	68,833	35,765	48.1
Retail	671,545	74,389	88.9
Accidents	16,132	1,229	92.4
Syn-T100,N1,L10	326,875	33,399	89.8
Syn-T200,N1,L10	281,151	33,021	88.3
Syn-T400,N1,L10	302,826	32,963	89.2

As shown in Table 9, IndxTAR produces few numbers of candidates for all datasets because it prunes infrequent ones at each level before advancing to next level. This will affect overall running time and required memory space greatly which will be discussed briefly in next subsections.

C. Memory Usage

The proposed IndxTAR algorithm uses TIndex data structure to index both original and incremental database transactions for faster support counting. As illustrated earlier in section III.A, TIndex indexing technique should reduce the required memory for storing the updated database. In this experiment, we measure the memory space for each dataset and its corresponding TIndex as shown in Table 10.

Table 10. Memory Usage Analysis – TIndex Size.

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
Syn-T100,N1,L10	25.298	19.733	21.9
Syn-T200,N1,L10	64.235	51.892	19.2
Syn-T400,N1,L10	122.766	90.114	26.6

As shown in Table 10, 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.

In addition, TIndex requires extra memory space to store the index. In this experiment, we measure required average memory usage for the three algorithms over the different datasets as shown in Table 11. It is clear that IndxTAR algorithm needs extra memory space than other two algorithms to discover frequent temporal itemsets, which is the main drawback of the algorithm. On the other hand, the improvement in running time covers memory space issue as discussed in next subsection.

Table 11. Comparison of Average Memory Usage (MB).

Dataset	TWAIN	ITARM	IndxTAR
Chess	13.097	13.786	18.252
Mushroom	6.966	7.224	12.242
Retail	37.383	44.977	128.586
Accidents	270.489	272.528	891.539
Syn-T100,N1,L10	14.035	14.869	37.609
Syn-T200,N1,L10	25.192	27.369	86.131
Syn-T400,N1,L10	36.166	38.024	120.544

D. Running Time

In this experiment, the running time (in seconds) of IndxTAR algorithm is compared with TWAIN and ITARM algorithms. Figures 5-8 show the measured running time with different *minsupp* threshold values in the four real datasets.

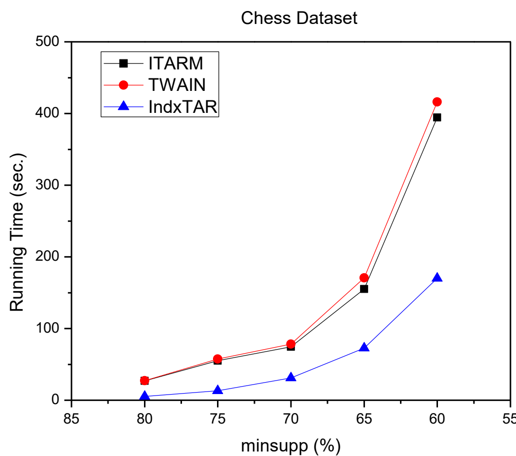

Figure 5. Running Time Comparison – Chess Dataset.

In Figure 5, IndxTAR algorithm overcomes the other two algorithms in running time. It requires only 170 seconds at 60% *minsupp* while TWIAN and ITARM need about 420 and 395 seconds, respectively. This huge difference is because Chess dataset is very dense and produces a huge number of candidates. ITARM needs extra time to check the support of these candidates while IndxTAR generated less number of candidates (about 0.67x) and uses TIndex to find their support faster.

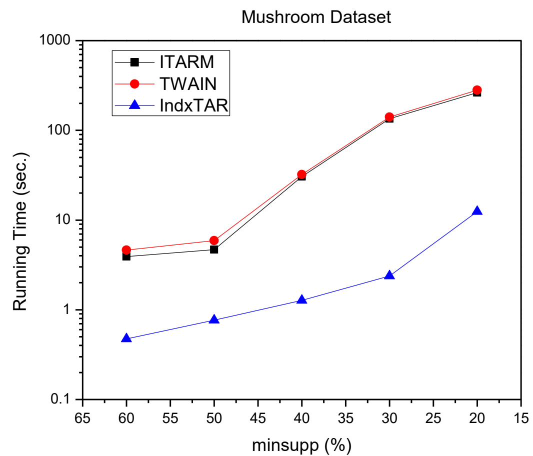

Figure 6. Running Time Comparison – Mushroom Dataset.

As shown in Figure 6, the running time of the proposed IndxTAR algorithm is very low compared to other algorithms. At low *minsupp* values (30% and 20%), ITARM and TWAIN algorithms generate nearly double the candidates than IndxTAR, which leads to a huge gap in running time (i.e. ITARM needs about 134 seconds while IndxTAR needs only 2.4 seconds at 30% *minsupp*). On the other hand, IndxTAR requires extra memory space to store the index (about 1.8x of ITARM).

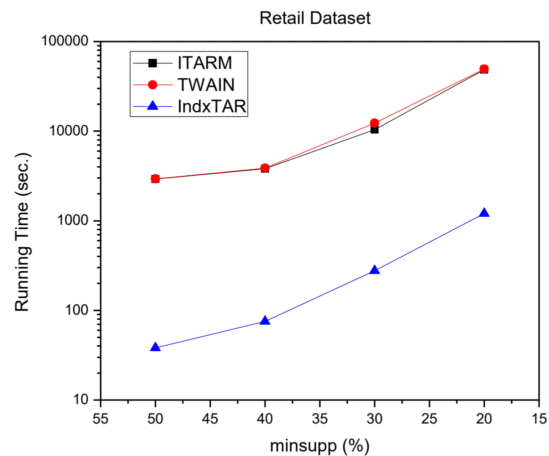

Figure 7. Running Time Comparison – Retail Dataset.

In Figure 7, IndxTAR works efficiently with Retail dataset because of the number of the generated candidates. TWAIN and ITARM algorithms generate a huge number of candidates (about 671K candidates on average) while IndxTAR generates about 74K only on average. This huge difference in the number of candidates leads to the shown big gap in running time, especially at low *minsupp* values. The cost of this performance was in memory usage. IndxTAR needs about 3.4x extra memory than other algorithms.

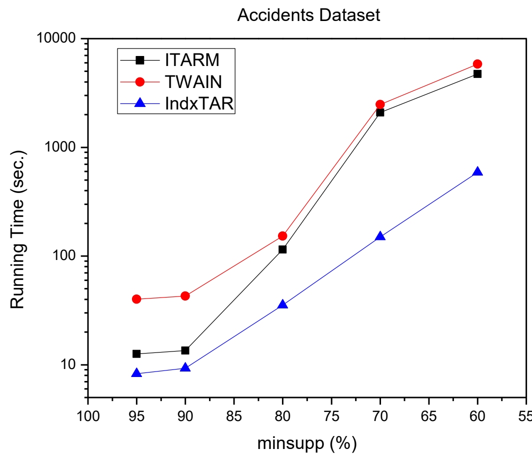

Figure 8. Running Time Comparison – Accidents Dataset.

Accidents dataset is very sparse, which produces a small number of candidates as shown in Table 9. IndxTAR needs extra memory space in this case (about 3.3x than ITARM) because dataset itself is large (contains about 300K transactions). On the other hand, using TIndex leads to very good running time speedup, about 8.8x faster than other algorithms.

Extra three synthetic datasets with different sizes (about 100K, 200K and 400K) were generated using The IBM Synthetic Dataset Generator [23]. The running time of the three algorithms is recorded as shown in Figures 9-11.

In general, IndxTAR algorithm needs very low running time compared to other algorithms. This huge gap occurred due to the large database size and its sparse feature. ITARM and TWAIN produce a huge number of candidates with scan reduction technique and need more running time to calculate their support by scanning database transactions with minimum memory space. On the other hand, IndxTAR requires extra memory space (about 3.3x on average) to store the index of these large databases but generates a very low number of candidates as shown in Table 9.

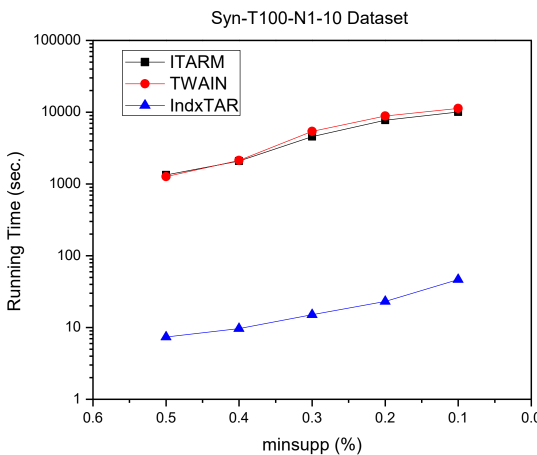

Figure 9. Running Time Comparison – Syn-T100-N1-L10 Dataset.

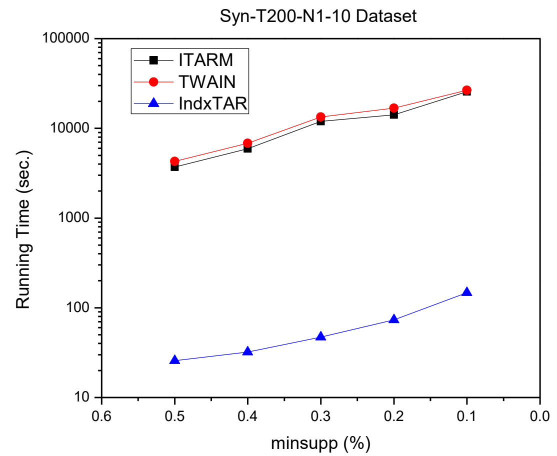

Figure 10. Running Time Comparison – Syn-T200-N1-L10 Dataset.

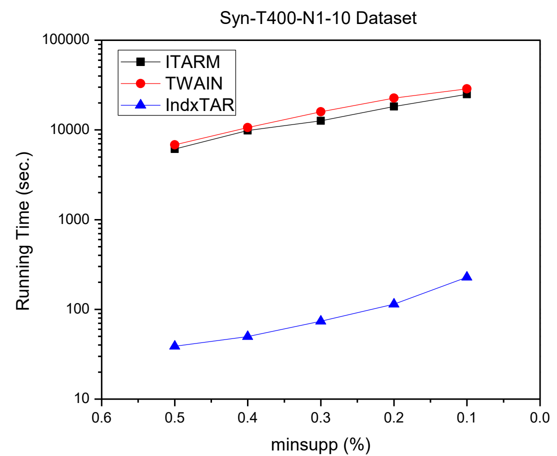

Figure 11. Running Time Comparison – Syn-T400-N1-L10 Dataset.

E. Scalability

We examined the scalability of the proposed algorithm with respect to large database sizes. We used the generated synthetic databases in this experiment due to their large sizes (100K, 200K and 400K). The running time of IndxTAR algorithm is measured (in seconds) at three *minsupp* values 0.1%, 0.3% and 0.5% as shown in Table 12. The results show that IndxTAR algorithm running time is scalable and nearly linear with respect to database size at different *minsupp* values as shown in Figure 12.

Table 12. IndxTAR running time with respect to database size.

No of Transactions (in thousands)	minsupp (0.5%)	minsupp (0.3%)	minsupp (0.1%)
100	7.403	15.076	46.775
200	25.67	47.195	147.276
400	38.9	73.89	227.73

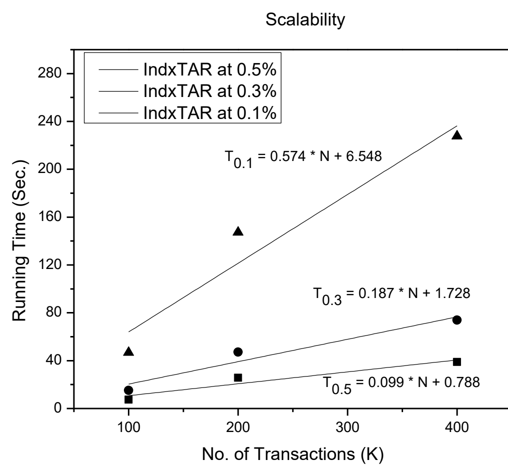

Figure 12. Scalability of the IndxTAR Algorithm. Points are computation presented in Table 12, and lines are linear fitting for such computations.

Figure 12 shows the measured running time shown in Table 12 and linear fitting equations for such measures. The results show that IndxTAR algorithm can deal efficiently with large databases even at low *minsupp* values with linear scalability.

V. Conclusions

In this paper, we addressed the problem of incremental mining of temporal association rule. The problem is mainly interested in finding frequent temporal itemsets in updated databases in which new transactions are added based on a certain time schedule. We proposed IndxTAR algorithm that introduced a new data structure to index temporal databases for efficient itemset support counting. The proposed algorithm utilized the previously discovered frequent temporal itemsets during the mining process of the updated database. Many experiments were conducted on both real and synthetic datasets to compare IndxTAR algorithm's performance with recently cited algorithms.

Experimental results showed that IndxTAR overcomes other algorithms by many orders of magnitude. This occurs because IndxTAR algorithm generates few candidates due to the early pruning using fast support counting from the TIndex index. Results also showed that IndxTAR algorithm requires extra memory space to store TIndex for updated temporal database. On the other hand, IndxTAR algorithm showed linear scalability running time when mining large databases even at low minsupp threshold values.

For future work, the proposed TIndex should be revised and adapted to be more efficient and more space preserving. This improvement will solve the extra memory space for the TIndex and improves the overall performance of the IndxTAR algorithm.

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] E. Yafi, A.S. Al-Hegami, M.A. Alam and R. Biswas, "YAMI: Incremental Mining of Interesting Association Patterns". *International Arab Journal of Information Technology*, 9(6), pp.504-510, 2012.

[3] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules". In *Proceedings of the International Conference on Very Large Data Bases (VLDB'94)*, pp. 487–499, 1994.

[4] 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*, 8(1), pp. 53–87, 2004.

[5] G. Grahne and J. Zhu, "Fast algorithms for frequent itemset mining using FP-trees", *IEEE Transactions on Knowledge and Data Engineering*, 17(10), pp. 1347–1362, 2005.

[6] B. Vo, F. Coenen and B. Le, "A new method for mining Frequent Weighted Itemsets based on WIT-trees", *Expert Systems with Applications*, 40(4), pp.1256-1264, 2013.

[7] H. Ning, H.Yuan and S. Chen, "Temporal association rules in mining method". In *Proceedings of the 1st International Multi-Symposiums on Computer and Computational Sciences (IMSCCS'06)*, Zhejiang, China, pp. 739–742, 2006.

[8] C-Y. Chang, M-S. Chen and C-H. Lee, "Mining general temporal association rules for items with different exhibition periods". In *Proceedings of the IEEE International Conference on Data Mining*, pp. 59–66, 2002.

[9] J-W. Huang, B-R. Dai and M-S. Chen, "Twain Two-end association miner with precise frequent exhibition periods", *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(2), 2007.

[10] T.F. Gharib, H. Nassar, M.Taha and A. Abraham, "An efficient algorithm for incremental mining of temporal association rules", *Data & Knowledge Engineering*, 69(8), pp. 800–815, 2010.

[11] A. Dafa-Alla, H-S. Shon, K. Saeed, M. Piao, U. Yun, K-J. Cheoi and K-H. Ryu, "IMTAR: Incremental Mining of General Temporal Association Rules", *Journal of Information Processing Systems*, 6(2), pp. 163–176, 2010.

[12] R. Chan, Q. Yang, and Y.D. Shen, "Mining high utility itemsets". In *Proceedings of the 3rd IEEE International Conference on Data Mining*, pp. 19–26, 2003.

[13] C.W. Lin, G.C. Lan and T.P. Hong, "An incremental mining algorithm for high utility itemsets", *Expert Systems with Applications*, 39(8), pp. 7173–7180, 2012.

[14] C.C. Change, Y.C. Li and J.S. Lee, "An Efficient Algorithm for Incremental Mining of Association Rules". In *Proceedings of the 15th International Workshop Research Issues in Data Engineering: Stream Data Mining and Applications (RIDE-SDMA'05*, pp. 1–8*)*, 2005.

[15] B. Nath, D.K. Bhattacharyya and A. Ghosh, "Incremental association rule mining: a survey". *Data Mining and Knowledge Discovery*, 3(3), pp.157-169, 2013.

[16] A.S. Al-Hegami, "A Framework for Incremental Mining of Interesting Temporal Association Rules". *International Journal of Computer Applications*, 131(8), pp.28-33, 2015.

[17] C-H. Lee, C-R. Lin and M-S. Chen, "On Mining General Temporal Association Rules in a Publication Database".

In *Proceedings of the IEEE International Conference on Data Mining*, pp. 487–499, 2001.
[18] C-H. Lee, M-S. Chen and C-R. Lin, "Progressive partition miner an efficient algorithm for mining general temporal association rules", *IEEE Transactions on Knowledge and Data Engineering*, 15(4), pp. 1004–1017, 2003.
[19] L. Jin, Y. Lee, S. Seo and K-H. Ryu, "Discovery of Temporal Frequent Patterns using TFP-Tree", *Advances in Web-Age Information Management (LNCS 4016)*, pp 349-361, 2006.
[20] Z. Linag, T. Ximming, L. Lin and J. Wenliang, "Temporal Association Rule Mining Based on T-Apriori Algorithm and its Typical Application". In *Proceedings of the International Symposium on Spatio-temporal Modeling, Spatial Reasoning, Analysis, Data Mining and Data Fusion*, 2005.
[21] N. Khairudin, A. Mustapha and M. Ahmad, "Effect of Temporal Relationships in Associative Rule Mining for Web Log Data", *The Scientific World Journal*, Article ID 813983, 10 pages, 2014.
[22] Frequent Itemset Mining Dataset Repository. Available online at: http://fimi.ua.ac.be/data/ (Accessed 1 May 2016)
[23] IBM Synthetic Dataset Generator. Available online at: http://miles.cnuce.cnr.it/~palmeri/datam/DCI/ (Accessed 1 May 2016)

Author Biographies

Mohammed M. Fouad is a Lecturer in Information Technology Department at the Faculty of Computing and Information Technology, King Abdulaziz University, Jeddah, Saudi Arabia. He received his Ph.D. and M.Sc. in Computer Science from Ain shams University, Cairo, Egypt in 2016 and 2009 respectively. His research interests are in data mining area especially in association rules discovery, text and web mining, and natural language processing.

Mostafa Gadal-Haqq M. Mostafa is a Professor of Computer Science at the Faculty of Computer and Information Sciences, Ain Shams University. He received a B.Sc. (Honor) in 1984 in Physics, a M.Sc. in 1989 in Computational Physics from the Faculty of Science, Ain Shams University, Cairo, Egypt, and a Ph.D. in 1996 in Computational Physics through joint supervision between Ain Shams University and Oak Ridge National Lab (ORNL), USA, in the period from 1993 to 1995. His research interests includes Computer Vision, Pattern Recognition, Data Mining, Medical Image Analysis, Arabic Optical Character Recognition, Speech Processing, Bioinformatics, and Information Security.