Incremental Mining on Association Rules

Wei-Guang Teng and Ming-Syan Chen

Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan, ROC

Summary. The discovery of association rules has been known to be useful in selective marketing, decision analysis, and business management. An important application area of mining association rules is the market basket analysis, which studies the buying behaviors of customers by searching for sets of items that are frequently purchased together. With the increasing use of the record-based databases whose data is being continuously added, recent important applications have called for the need of incremental mining. In dynamic transaction databases, new transactions are appended and obsolete transactions are discarded as time advances. Several research works have developed feasible algorithms for deriving precise association rules efficiently and effectively in such dynamic databases. On the other hand, approaches to generate approximations from data streams have received a significant amount of research attention recently. In each scheme, previously proposed algorithms are explored with examples to illustrate their concepts and techniques in this chapter.

1 Introduction

Due to the increasing use of computing for various applications, the importance of data mining is growing at rapid pace recently. It is noted that analysis of past transaction data can provide very valuable information on customer buying behavior, and thus improve the quality of business decisions. In essence, it is necessary to collect and analyze a sufficient amount of sales data before any meaningful conclusion can be drawn therefrom. Since the amount of these processed data tends to be huge, it is important to devise efficient algorithms to conduct mining on these data. Various data mining capabilities have been explored in the literature [2, 4, 5, 12, 10, 11, 21, 39, 51, 52]. Among them, the one receiving a significant amount of research attention is on mining association rules over basket data [2, 3, 16, 22, 25, 33, 36, 41, 44, 50]. For example, given a database of sales transactions, it is desirable to discover all associations among items such that the presence of some items in a transaction will imply the presence of other items in the same transaction, e.g., 90%

of customers that purchase milk and bread also purchase eggs at the same time.

Recent important applications have called for the need of incremental mining. This is due to the increasing use of the record-based databases whose data is being continuously added. Examples of such applications include Web log records, stock market data, grocery sales data, transactions in electronic commerce, and daily weather/traffic records, to name a few. In many applications, we would like to mine the transaction database for a fixed amount of most recent data (say, data in the last 12 months). That is, in the incremental mining, one has to not only include new data (i.e., data in the new month) into, but also remove the old data (i.e., data in the most obsolete month) from the mining process.

A naive approach to solve the incremental mining problem is to re-run the mining algorithm on the updated database. However, it obviously lacks of efficiency since previous results are not utilized on supporting the discovering of new results while the updated portion is usually small compared to the whole dataset. Consequently, the efficiency and the effectiveness of algorithms for incremental mining are both crucial issues which will be discussed in details in this chapter.

The rest of the chapter is organized as follows. Preliminaries including the mining of association rules and the need for performing incremental mining are given in Section 1. Algorithms for mining association rules incrementally from transactional databases that generate precise results are described in Section 2. On the other hand, algorithms dealing with data streams (i.e., online transaction flows) that generate approximate results are explored in Section 3. In each scheme, the algorithms can be further categorized according to their major principles. Namely, the Apriori-based algorithms, the partition-based algorithms and the pattern growth algorithms. In Section 4, remarks on the relationship between two schemes are made, giving the summary of the state of the art for incremental mining on association rules.

1.1 Mining Association Rules

Mining association rules was first introduced in [2], where the goal is to discover interesting relationships among items in a given transactional dataset.

A mathematical model was proposed in [2] to address the problem of mining association rules. Let $\mathcal{I}$=$\{i_1, i_2, ..., i_m\}$ be a set of literals, called items. Let D be a set of transactions, where each transaction T is a set of items such that $T \subseteq \mathcal{I}$. Note that the quantities of items bought in a transaction are not considered, meaning that each item is a binary variable representing if an item was bought. Each transaction is associated with an identifier, called TID. Let X be an itemset, i.e., a set of items. A transaction T is said to contain X if and only if $X \subseteq T$. An association rule is an implication of the form $X \Longrightarrow Y$, where $X \subset \mathcal{I}$, $Y \subset \mathcal{I}$ and $X \bigcap Y = \phi$. The rule $X \Longrightarrow Y$ has *support* s in the transaction set D if $s\%$ of transactions in D contain $X \bigcup Y$.

The rule $X \Longrightarrow Y$ holds in the transaction set D with *confidence* c if $c\%$ of transactions in D that contain X also contain Y. By utilizing the notation of probability theory, the concepts of support and confidence for an association rule can be formulated as

$$support(X \Longrightarrow Y) = P(X \cup Y) \text{ and,}$$
$$confidence(X \Longrightarrow Y) = P(Y|X).$$

For a given pair of support and confidence thresholds, namely the minimum support S_{min} and the minimum confidence C_{min}, the problem of mining association rules is to find out all the association rules that have confidence and support greater than the corresponding thresholds. Moreover, the mining of association rules is a two-step process:
1. *Find all frequent itemsets*: Frequent itemsets are itemsets satisfying the support threshold, i.e., $\{X|X.support \geq S_{min}\}$. In some earlier literatures, frequent itemsets are also termed as *large* itemsets and the set of frequent k-itemsets (which are composed of k items) is thus commonly denoted by L_k. Consequently, the goal in this step is to discover the set $\{L_1, ...L_k\}$.
2. *Generate association rules from the frequent itemsets*: For any pair of frequent itemsets W and X satisfying $X \subset W$, if $\frac{X.support}{W.support} \geq C_{min}$, then $X \Longrightarrow Y (= W - X)$ is identified as a valid rule.

The overall performance of mining association rules is in fact determined by the first step. After the frequent itemsets are identified, the corresponding association rules can be derived in a straightforward manner [2] as shown in the second step above. A numerous prior works including the Apriori [2], the DHP [41], and partition-based ones [32, 44] are proposed to solve the first subproblem efficiently. In addition, several novel mining techniques, including TreeProjection [1], FP-tree [24, 25, 43], and constraint-based ones [23, 27, 42, 50] also received a significant amount of research attention. To clarify the idea of the mining of association rules before formally introducing the incremental mining, some representative algorithms along with illustrative examples are provided in following sections.

Apriori-Based Algorithms

Algorithm Apriori [2] is an influential algorithm for mining association rules. It uses *prior knowledge* of frequent itemset properties to help on narrowing the search space of required frequent itemsets. Specifically, k-itemsets are used to explore (k+1)-itemsets during the *levelwise* process of frequent itemset generation. The set of frequent 1-itemsets (L_1) is firstly found by scanning the whole dataset once. L_1 is then used by performing join and prune actions to form the set of candidate 2-itemsets (C_2). After another data scan, the set of frequent 2-itemsets (L_2) are identified and extracted from C_2. The whole process continues iteratively until there is no more candidate itemsets which can be formed from previous L_k.

D	T_1	A B C
	T_2	A F
	T_3	A B C E
	T_4	A B D F
	T_5	C F
	T_6	A B C
	T_7	A B C E
	T_8	C D E
	T_9	B D E

Fig. 1. An illustrative transaction database

Example 1: Consider an example transaction database given in Fig. 1. In each iteration (or each pass), algorithm Apriori constructs a candidate set of large itemsets, counts the number of occurrences of each candidate itemset, and then determine large itemsets based on a predetermined minimum support S_{min}. In the first iteration, Apriori simply scans all the transactions to count the number of occurrences for each item. The set of candidate 1-itemsets, C_1, obtained is shown in Fig. 2. Assuming that S_{min}=40%, the set of large 1-itemsets (L_1) composed of candidate 1-itemsets with the minimum support required, can then be determined. To discover the set of large 2-itemsets, in view of the fact that any subset of a large itemset must also have minimum support, Apriori uses L_1*L_1 to generate a candidate set of itemsets C_2 where * is an operation for concatenation in this case. C_2 consists of

$$\binom{|L_1|}{2}$$

2-itemsets. Next, the nine transactions in D are scanned and the support of each candidate itemset in C_2 is counted. The middle table of the second row in Fig. 2 represents the result from such counting in C_2. The set of large 2-itemsets, L_2, is therefore determined based on the support of each candidate 2-itemset in C_2.

The set of candidate 3-itemsets (C_3) is generated from L_2 as follows. From L_2, two large 2-itemsets with the same first item, such as {AB} and {AC}, are identified first. Then, Apriori tests whether the 2-itemset {BC}, which consists of their second items, contributes a frequent 2-itemset or not. Since {BC} is a frequent itemset by itself, we know that all the subsets of {ABC} are frequent and then {ABC} becomes a candidate 3-itemset. There is no other candidate 3-itemset from L_2. Apriori then scans all the transactions and discovers the large 3-itemsets L_3 in Fig. 2. Since there is no candidate 4-itemset to be constituted from L_3, Apriori ends the process of discovering frequent itemsets.

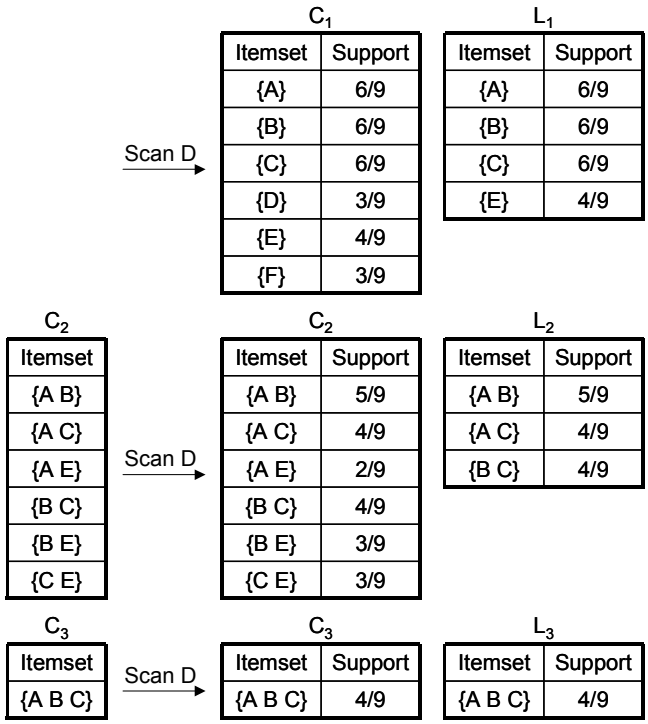

Fig. 2. Generation of candidate itemsets and frequent itemsets using algorithm Apriori

Similar to Apriori, another well known algorithm, DHP [40], also generates candidate k-itemsets from L_{k-1}. However, DHP employs a hash table, which is built in the previous pass, to test the eligibility of a k-itemset. Instead of including all k-itemset from L_{k-1}*L_{k-1} into C_k, DHP adds a k-itemset into C_k only if that k-itemset is hashed into a hash entry whose value is larger than equal to the minimum transaction support required. As a result, the size of candidate set C_k can be reduced significantly. Such a filtering technique is particularly powerful in reducing the size of C_2.

Example 2: The effect of utilizing the hash table on helping to reduce the size of C_2 is provided in Fig. 3. It is noted that since the total counts for buckets 1 and 3 cannot satisfy the minimum support constraint, itemsets in these buckets, e.g., {A E}, should not be included in C_2. This improve the computing efficiency while the number of candidate itemsets to be checked is reduced.

DHP also reduces the database size progressively by not only trimming each individual transaction size but also pruning the number of transactions in the database. We note that both DHP and Apriori are iterative algorithms

Bucket Address	0	1	2	3	4	5	6
Bucket Count	4	3	8	2	4	6	5
Bucket Contents	{C E} {A D} {C E} {C E}	{A E} {C F} {A E}	{B C} {A F} {B C} {A F} {B C} {B C} {D E} {D E}	{B D} {B D}	{B E} {D F} {B E} {B E}	{A B} {A B} {A B} {B F} {A B} {A B}	{A C} {A C} {A C} {A C} {C D}

Fig. 3. An example hash table for reducing the size of C_2 in previous example. The corresponding hash function is h(x,y)=[(order of x)*10+(order of y)]mod 7 where the order of item A is 1, the order of item B is 2, and so on.

on the frequent itemset size in the sense that the frequent k-itemset are derived from the frequent (k-1)-itemsets.

Most of the previous studies, including those in [2, 7, 13, 14, 41, 45, 48], belong to Apriori-based approaches. Basically, an Apriori-based approach is based on an anti-monotone Apriori heuristic [2], i.e., *if any itemset of length k is not frequent in the database, its length ($k+1$) super-itemset will never be frequent.* The essential idea is to iteratively generate the set of candidate itemsets of length $(k+1)$ from the set of frequent itemsets of length k (for $k \geq 1$), and to check their corresponding occurrence frequencies in the database. As a result, if the largest *frequent* itemset is a j-itemset, then an Apriori-based algorithm may need to scan the database up to $(j+1)$ times.

In Apriori-based algorithms, C_3 is generated from $L_2 * L_2$. In fact, a C_2 can be used to generate the candidate 3-itemsets. This technique is referred to as scan reduction in [10]. Clearly, a C'_3 generated from $C_2 * C_2$, instead of from $L_2 * L_2$, will have a size greater than $|C_3|$ where C_3 is generated from $L_2 * L_2$. However, if $|C'_3|$ is not much larger than $|C_3|$, and both C_2 and C_3 can be stored in main memory, we can find L_2 and L_3 together when the next scan of the database is performed, thereby saving one round of database scan. It can be seen that using this concept, one can determine all L_ks by as few as two scans of the database (i.e., one initial scan to determine L_1 and a final scan to determine all other frequent itemsets), assuming that C'_k for $k \geq 3$ is generated from C'_{k-1} and all C'_k for $k > 2$ can be kept in the memory. In [11], the technique of scan-reduction was utilized and shown to result in prominent performance improvement.

Partition-Based Algorithms

There are several techniques developed in prior works to improve the efficiency of algorithm Apriori, e.g., hashing itemset counts, transaction reduction, data sampling, data partitioning and so on. Among the various techniques, the data partitioning is the one with great importance since the goal in this chapter is on the incremental mining where bulks of transactions may be appended or discarded as time advances.

The works in [32, 38, 44] are essentially based on a partition-based heuristic, i.e., *if X is a frequent itemset in database D which is divided into n partitions p_1, p_2, ..., p_n, then X must be a frequent itemset in at least one of the n partitions.* The partition algorithm in [44] divides D into n partitions, and processes one partition in main memory at a time. The algorithm first scans partition p_i, for $i = 1$ to n, to find the set of all local frequent itemsets in p_i, denoted as L^{p_i}. Then, by taking the union of L^{p_i} for $i = 1$ to n, a set of candidate itemsets over D is constructed, denoted as C^G. Based on the above partition-based heuristic, C^G is a superset of the set of all frequent itemsets in D. Finally, the algorithm scans each partition for the second time to calculate the support of each itemset in C^G and to find out which candidate itemsets are really frequent itemsets in D.

Example 3: The flow of algorithm Partition is shown in Fig. 4. In this example, D is divided into three partitions, i.e., P_1, P_2 and P_3, and each partition contains three transactions. The sets of locally frequent itemsets L^{P_i} are discovered based on transactions in each partition. For example, $L^{P_2} = \{\{\text{A}\}, \{\text{B}\}, \{\text{C}\}, \{\text{F}\}, \{\text{A B}\}\}$. As is also shown in Fig. 4, the set of global candidate itemsets C^G is then generated by taking the union of L^{P_i}. Finally, these candidate itemsets are verified by scanning the whole dataset D once more.

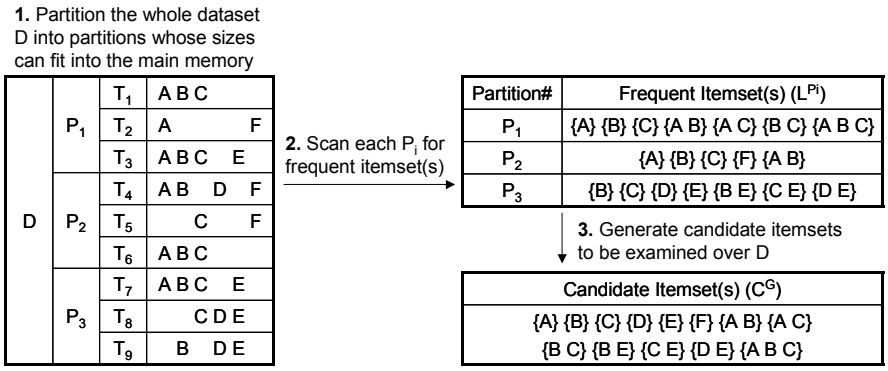

Fig. 4. Generation of global candidate itemsets using algorithm Partition.

Instead of constructing C^G by taking the union of L^{p_i}, for $i = 1$ to n, at the end of the first scan, some variations of the above partition algorithm are proposed in [32, 38]. In [38], algorithm SPINC constructs C^G incrementally by adding L^{p_i} to C^G whenever L^{p_i} is available. SPINC starts the counting of occurrences for each candidate itemset $c \in C^G$ as soon as c is added to C^G. In [32], algorithm AS-CPA employs prior knowledge collected during the mining process to further reduce the number of candidate itemsets and to overcome the problem of data skew. However, these works were not devised to handle incremental updating of association rule.

Pattern Growth Algorithms

It is noted that the generation of frequent itemsets in both the Apriori-based algorithms and the partition-based algorithms is in the style of candidate generate-and-test. No matter how the search space for candidate itemsets is narrowed, in some cases, it may still need to generate a huge number of candidate itemsets. In addition, the number of database scans is limited to be at least twice, and usually some extra scans are needed to avoid unreasonable computing overheads. These two problems are nontrivial and are resulted from the utilization of the Apriori approach.

To overcome these difficulties, the tree structure which stores projected information of large datasets are utilized in some prior works [1, 25]. In [1], the algorithm TreeProjection constructs a lexicographical tree and has the whole database projected based on the frequent itemsets mined so far. The transaction projection can limit the support counting in a relatively small space and the lexicographical tree can facilitate the management of candidate itemsets. These features of algorithm TreeProjection provide a great improvement in computing efficiency when mining association rules.

An influential algorithm which further attempts to avoid the generation of candidate itemsets is proposed in [25]. Specifically, the proposed algorithm FP-growth (frequent pattern growth) adopts a *divide-and-conquer* strategy. Firstly, all the transactions are projected into an FP-tree (frequent pattern tree), which is a highly compressed structure, based on the frequent 1-itemsets in descending order of their supports. Then, for each frequent item, the conditional FP-tree can be extracted to be mined for the generation of corresponding frequent itemsets. To further illustrate how algorithm FP-growth works, an example is provided below.

Example 4: Consider the example transaction database given in Fig. 1. After scanning the whole database once, the frequent 1-itemsets discovered are {A:6}, {B:6}, {C:4} and {E:4} where the numbers are occurrence counts for the items. Note that the ordering of frequent items, which is the ordering of item supports in descending order, is crucial when constructing the FP-tree.

The construction of the FP-tree is shown in Fig. 5. The first transaction T_1:{A B C} is mapped to the single branch of the FP-tree in Fig. 5(a). In Fig.

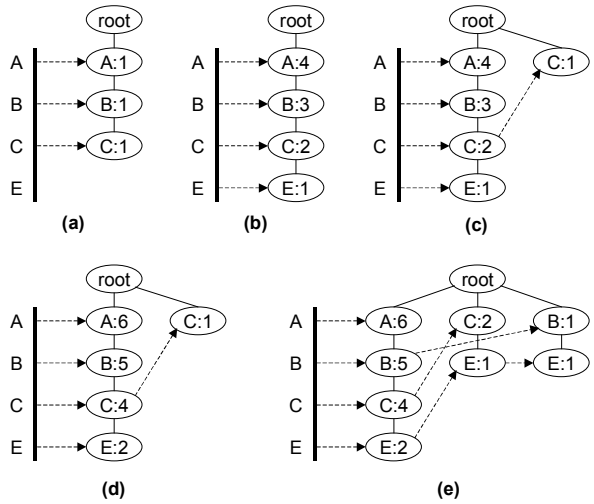

Fig. 5. Building the FP-tree based on the dataset in Fig. 1: (a) T_1 is added; (b) T_2, T_3 and T_4 are added; (c) T_5 is added; (d) T_6 and T_7 are added; (e) T_8 and T_9 are added (final FP-tree)

5(b), the next three transactions are added by simply extending the branch and increasing the corresponding counts of existing item nodes since these transactions can fit into that branch when infrequent items, i.e., items D and F, are discarded. However, the transaction T_5 contains only one frequent item C and a new branch is thus created. Note that the node-link (composing of arrows with dotted line) for item C is extended in Fig. 5(c) for tracking the information of frequent itemsets containing item C. Specifically, all the possible frequent itemsets can be obtained by following the node-links for the items. In Fig. 5(d), transaction T_6 and T_7 are added in a similar way as the addition of transactions T_2 to T_4. Finally, the complete FP-tree in Fig. 5(e) is constructed by adding the last two transactions T_8 and T_9. It can be noted from this example that since the heuristic that popular items tend to occur together works in most cases, the resulting FP-tree can achieve a very high data compression ratio, showing the compactness of algorithm FP-growth.

To discover all the frequent itemsets, each frequent item along with its conditional FP-tree are mined separately and iteratively, i.e., the *divide-and-conquer* strategy. For example, by firstly tracking the node-link of item A, the 3-itemset {A B C} is found to frequent if S_{min}=40%. Its subsets which contain the item A, i.e., {A B} and {A C}, can then be discovered with correct supports easily. In addition, by tracking the node-links of both items B and C, only one more frequent itemset {B C} is generated, showing the completeness of algorithm FP-growth.

1.2 Incremental Mining Primitives

The mining of association rules on transactional database is usually an offline process since it is costly to find the association rules in large databases. With usual market-basket applications, new transactions are generated and old transactions may be obsolete as time advances. As a result, incremental updating techniques should be developed for maintenance of the discovered association rules to avoid redoing mining on the whole updated database.

A database may allow frequent or occasional updates and such updates may not only invalidate existing association rules but also activate new rules. Thus it is nontrivial to maintain such discovered rules in large databases. Note that since the underlying transaction database has been changed as time advances, some algorithms, such as Apriori, may have to resort to the regeneration of candidate itemsets for the determination of new frequent itemsets, which is, however, very costly even if the incremental data subset is small. On the other hand, while FP-tree-based methods [24, 25, 43] are shown to be efficient for small databases, it is expected that their deficiency of memory overhead due to the need of keeping a portion of database in memory, as indicated in [26], could become more severe in the presence of a large database upon which an incremental mining process is usually performed. Consequently, ordinary approaches for mining association rules are closely related to solving the problem of incremental mining. However, these algorithms cannot be applied directly without taking the incremental characteristics into consideration.

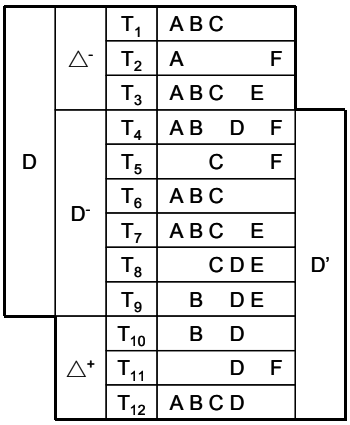

Fig. 6. An illustrative transactional database for incremental mining

The concept of incremental mining on transaction databases in further illustrated in Fig. 6. For a dynamic database, old transactions ($\triangle^-$) are deleted from the database D and new transactions ($\triangle^+$) are added as time advances.

Naturally, $\triangle^- \subseteq D$. Denote the updated database by D', $D' = (D - \triangle^-) \cup \triangle^+$. We also denote the set of unchanged transactions by $D^- = D - \triangle^-$.

Generally speaking, the goal is to solve the efficient update problem of association rules after a nontrivial number of new records have been added to or removed from a database. Assuming that the two thresholds, minimum support S_{min} and minimum confidence C_{min}, do not change, there are several important characteristics in the update problem.

1. The update problem can be reduced to finding the new set of frequent itemsets. After that, the new association rules can be computed from the new frequent itemsets.
2. An old frequent itemset has the potential to become infrequent in the updated database.
3. Similarly, an old infrequent itemset could become frequent in the new database.
4. In order to find the new frequent itemsets "exactly", all the records in the updated database, including those from the original database, have to be checked against every candidate set.

Note that the fourth characteristic is generally accepted when looking for exact frequent itemsets (and thus the association rules) from updated databases. On the contrary, in the data stream environment, the approximation of exact frequent itemsets is a key ingredient due to the high speed and huge volume of input transactions. Consequently, only increment portion of data rather than the whole unchanged portion has to be scanned, leading to an efficient way in performing updates of frequent itemsets. However, it is noted that the quality of approximation should be guaranteed within a probabilistic or deterministic error range, showing that the task of incremental mining on data streams is of more challenge.

To further understand the incremental mining techniques from either transaction databases or data streams, details are generally discussed in following sections. The algorithms of incremental mining which seek for exactly updated association rules from transactional databases are presented in Section 2. The importance and approaches targeting at mining and maintaining approximations incrementally from data streams are explored in Section 3.

2 Mining Association Rules Incrementally from Transactional Databases

Since database updates may introduce new association rules and invalidate some existing ones, it is important to study efficient algorithms for incremental update of association rules in large databases. In this scheme, a major portion of the whole dataset is remain unchanged while new transactions are appended and obsolete transactions may be discarded. By utilizing different core techniques, algorithms for incremental mining from transactional data-

bases can be categorized into Apriori-based, partition-based or pattern growth algorithms, which will be fully explored in this section.

2.1 Apriori-Based Algorithms for Incremental Mining

As mentioned earlier, the Apriori heuristic is an anti-monotone principle. Specifically, *if any itemset is not frequent in the database, its super-itemset will never be frequent.* Consequently, algorithms belonging to this category adopt a levelwise approach, i.e., from shorter itemsets to longer itemsets, on generating frequent itemsets.

Algorithm FUP (Fast UPdate)

Algorithm FUP (Fast UPdate) [13] is the first algorithm proposed to solve the problem of incremental mining of association rules. It handles databases with transaction insertion only, but is not able to deal with transaction deletion. Specifically, given the original database D and its corresponding frequent itemsets $L = \{L_1, ..., L_k\}$. The goal is to reuse the information to efficiently obtain the new frequent itemsets $L' = \{L'_1, ..., L'_k\}$ on the new database $D' = D \cup \Delta^+$.

By utilizing the definition of support and the constraint of minimum support S_{min}. The following lemmas are generally used in algorithm FUP.
1. An original frequent itemset X, i.e., $X \in L$, becomes infrequent in D' if and only if $X.support_{D'} < S_{min}$.
2. An original infrequent itemset X, i.e., $X \notin L$, may become frequent in D' only if $X.support_{\Delta^+} \geq S_{min}$.
3. If a k-itemset X whose (k-1)-subset(s) becomes infrequent, i.e., the subset is in L_{k-1} but not in L'_{k-1}, X must be infrequent in D'.

Basically, similarly to that of Apriori, the framework of FUP, which can update the association rules in a database when new transactions are added to the database, contains a number of iterations [13, 14]. The candidate sets at each iteration are generated based on the frequent itemsets found in the previous iteration. At the k-th iteration of FUP, Δ^+ is scanned exactly once. For the original frequent itemsets, i.e., $\{X | X \in L_k\}$, they only have to be checked against the small increment Δ^+. To discover the new frequent itemsets, the set of candidate itemsets C_k is firstly extracted from Δ^+, and then be pruned according to the support count of each candidate itemset in Δ^+. Moreover, the pool for candidate itemsets can be further reduced by discarding itemsets whose (k-1)-subsets are becoming infrequent.

The flows of FUP can be best understood by the following example. The dataset with the increment portion labeled as Δ^+ is shown in Fig. 7. Note that the first nine transactions are identical to those shown in earlier examples. In addition, the frequent itemsets of the unchanged portion D is also shown in Fig. 7, where the generation process is described earlier and is thus omitted here.

D	T_1	A B C
	T_2	A F
	T_3	A B C E
	T_4	A B D F
	T_5	C F
	T_6	A B C
	T_7	A B C E
	T_8	C D E
	T_9	B D E
Δ^+	T_{10}	B D
	T_{11}	D F
	T_{12}	A B C D

(Original) frequent itemsets $\{L_k\}$ of D

Itemset	Support
{A}	6/9
{B}	6/9
{C}	6/9
{E}	4/9
{A B}	5/9
{A C}	4/9
{B C}	4/9
{A B C}	4/9

Fig. 7. An illustrative database for performing algorithm FUP, and the original frequent itemsets generated using association rule mining algorithm(s)

Example 5: The first iteration of algorithm FUP when performing incremental mining is represented in Fig. 8. The original frequent 1-itemsets are firstly verified on the increment portion Δ^+, and only itemsets with new supports no less than S_{min}(=40%) is retained as a part of new frequent 1-itemsets L'_1, i.e., {A}, {B} and {C}. Then the supports of other possible items are also checked in Δ^+, leading to the construction of C_1:{{D}} to be further verified against the unchanged portion D. Finally, the new L'_1 is generated by integrating the results from both possible sources.

1. Check new supports of L_1

Itemset	Support
{A}	7/12
{B}	8/12
{C}	7/12
{E}	4/12

2. Check other items in Δ^+

Itemset	Support
{D}	3/3
{F}	1/3

3. Check C_1 on D

Itemset	Support
{D}	6/12

L_1'

Itemset	Support
{A}	7/12
{B}	7/12
{C}	7/12
{D}	6/12

Fig. 8. The first iteration of algorithm FUP on the dataset in Fig. 6

The successive iterations work roughly the same as the first iteration. However, since the shorter frequent itemsets have already been discovered, the

information can be utilized to further reduce the pool of candidate itemsets. Specifically, as shown in Fig. 9, since the 1-itemset {E} becomes infrequent as Δ^+ is considered, i.e., {E}$\in (L1 - L_1^{'})$, all itemsets in L_2 containing a subset of {E} should be infrequent and are thus discarded with no doubt. Other itemsets in L_2 are then verified again Δ^+ to see if they are still frequent. The set of candidate 2-itemsets C_2 is constructed by $(L_1' * L_1' - L_2)$ since itemsets in L_2 are already checked. In this example, the 2-itemsets {A D}, {B D} and {C D} are firstly checked against Δ^+. Afterward, only {B D} is being checked against the unchanged portion D, since it is the only frequent itemset in Δ^+. Finally, the new L_2' is generated by integrating the results from both possible sources. The generation of L_k' is of analogous process to that of L_2', and it works iteratively until no more longer candidate itemsets can be formed. In this example, algorithm FUP stops when the only frequent 3-itemset {A B C} is discovered.

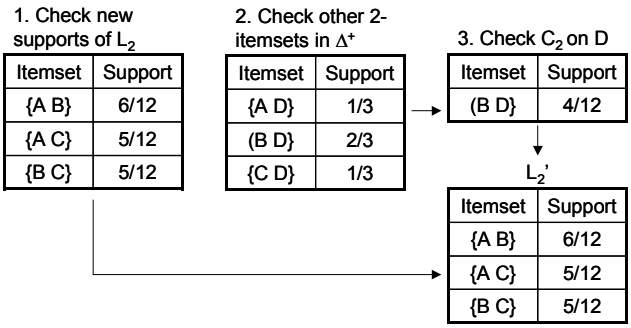

Fig. 9. The second iteration of algorithm FUP on the dataset in Fig. 6

Specifically, the key steps of FUP can be listed below. (1) At each iteration, the supports of the size-k frequent itemsets in L are updated against Δ^+ to filter out those that are no longer in D'. (2) While scanning Δ^+, a set of candidate itemsets, C_k, is extracted together with their supports in Δ^+ counted. The supports of these sets in C_k are then updated against the original database D to find the "new" frequent itemsets. (3) Many itemsets in C_k can be pruned away by checking their supports in Δ^+ before the update against the original database starts. (4) The size of C_k is further reduced at each iteration by pruning away a few original frequent itemsets in Δ^+.

The major idea is to reuse the information of the old frequent itemsets and to integrate the support information of the new frequent itemsets in order to substantially reduce the pool of candidate sets to be re-examined. Consequently, as compared to that of Apriori or DHP, the number of candidate itemsets to be checked against the whole database $D' = D \cup \Delta^+$ is much smaller, showing the major advantage of algorithm FUP.

Algorithms $\mathbf{FUP_2}$ and $\mathbf{FUP_2\mathcal{H}}$

The algorithm FUP [13] updates the association rules in a database when new transactions are *added* to the database. An extension to algorithm FUP was reported in [14] where the authors propose an algorithm FUP_2 for updating the existing association rules when transactions are *added* to and *deleted* from the database. In essence, FUP_2 is equivalent to FUP for the case of insertion, and is, however, a complementary algorithm of FUP for the case of deletion.

For a general case that transactions are added and deleted, algorithm FUP_2 can work smoothly with both the deleted portion Δ^- and the added portion Δ^+ of the whole dataset. A very feature is that the old frequent k-itemsets L_k from the previous mining result is used for dividing the candidate set C_k into two parts: $P_k = C_k \cap L_k$ and $Q_k = C_k - P_k$. In other words, P_k (Q_k) is the set of candidate itemsets that are previously frequent (infrequent) with respect to D. For the candidate itemsets in Q_k, their supports are unknown since they were infrequent in the original database D, posing some difficulties in generating new frequent itemsets. Fortunately, it is noted that if a candidate itemset in Q_k is frequent in Δ^-, it must be infrequent in D^-. This itemset is further identified to be infrequent in the updated database D' if it is also infrequent in Δ^+. This technique helps on effectively reducing the number of candidate itemsets to be further checked against the unchanged portion D^- which is usually much larger than either Δ^- or Δ^+.

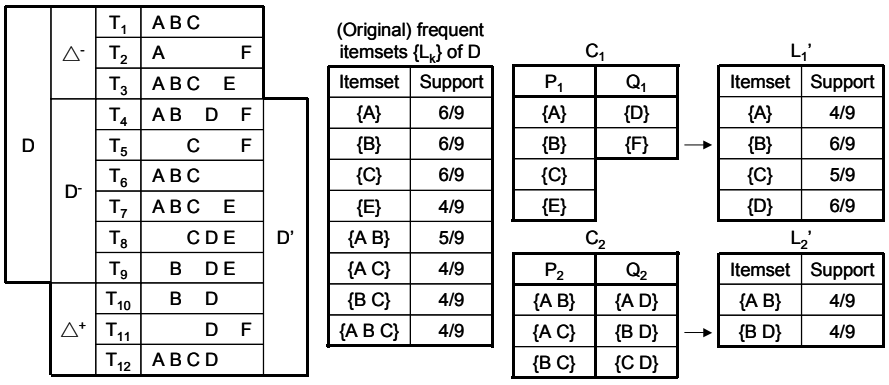

Fig. 10. Generation of new frequent itemsets from the dataset in Fig. 6 using algorithm FUP_2

Example 6: To further illustrate the flow of algorithm FUP_2, an example is provided in Fig. 10. In the first iteration, C_1 is exactly the set of all items. In subsequent iterations, C_k is generated from L'_{k-1}, the frequent itemsets found in the previous iteration. As shown in Fig. 10, the set of candidate itemsets C_i can be further divided into P_i and Q_i. For each itemset in P_i,

the corresponding support for D is known since the itemsets is previously frequent. Therefore, by scanning only the deleted portion Δ^- and the added portion Δ^+, the new support can be obtained. For example, $Count(\{\mathrm{A}\})_{D'} = Count(\{\mathrm{A}\})_D - Count(\{\mathrm{A}\})_{\Delta^-} + Count(\{\mathrm{A}\})_{\Delta^+} = 6 - 3 + 1 = 4$ where $Count(X)_{db}$ represents the occurrence counts for itemset X in dataset db. On the other hand, to verify if an itemset in Q_i is frequent or not, the cost of scanning the unchanged (and usually large) portion D^- could be required since the corresponding support is previously unknown. Fortunately, in some cases, only the scans of Δ^- and Δ^+ are required. For example, it can be easily observed that $Count(\{\mathrm{F}\})_{\Delta^+} - Count(\{\mathrm{F}\})_{\Delta^-} = 0 \leq (|\Delta^+| - |\Delta^-|) \times S_{min} = 0$, showing that the support of itemset $\{\mathrm{F}\}$ could not be improved by introducing the updated transactions (both deleted and added transactions are considered.) Consequently, fewer itemsets have to be further scanned against the unchanged portion D^-. An iteration is finished when all the itemsets in P_i and Q_i are all verified and therefore the new set of frequent itemsets L'_i is generated.

Another FUP-based algorithm, call FUP$_2\mathcal{H}$, was also devised in [14] to utilize the hash technique for performance improvement. In a similar way to that of the algorithm DHP, the counts of itemsets in D^- can be hashed, leading to an immediate improvement on efficiency.

Algorithm UWEP (Update With Early Pruning)

As pointed out earlier, the existing FUP-based algorithms in general suffer from two inherent problems, namely (1) the occurrence of a potentially huge set of candidate itemsets, which is particularly critical for incremental mining since the candidate sets for the original database and the incremental portion are generated separately, and (2) the need of multiple scans of database.

In [6], algorithm UWEP is proposed using the technique of update with early pruning. The major feature of algorithm UWEP over other FUP-based algorithms is that it prunes the supersets of an originally frequent itemset in D as soon as it becomes infrequent in the updated database D', rather than waiting until the k-th iteration. In addition, only itemsets which are frequent in both Δ^+ and $D'(= D \cup \Delta^+)$ are taken to generate candidate itemsets to be further checked against Δ^+. Specifically, if a k-itemset is frequent in Δ^+ but infrequent in D', it is not considered when generating C_{k+1}. This can significantly reduce the number of candidate itemsets in Δ^+ with the trade-off that an additional set of unchecked itemsets has to be maintained during the mining process. Consequently, these early pruning techniques can enhance the efficiency of FUP-based algorithms.

Algorithm Utilizing Negative Borders

Furthermore, the concept of *negative borders* [48] is utilized in [47] to improve the efficiency of FUP-based algorithms on incremental mining. Specifically,

given a collection of frequent itemsets L, closed with respect to the set inclusion relation, the negative border $Bd^-(L)$ of L consists of the *minimal* itemsets $X \subseteq R$ not in L where R is the set of all items. In other words, the negative border consists of all itemsets that were candidates of the level-wise method which did not have enough support. That is, $Bd^-(L_k) = C_k - L_k$ where $Bd^-(L_k)$ is the set of k-itemsets in $Bd^-(L)$. The intuition behind the concept is that given a collection of frequent itemsets, the negative border contains the "closest" itemsets that could be frequent, too. For example, assume the collection of frequent itemsets is L={{A}, {B}, {C}, {F}, {A B}, {A C}, {A F}, {C F}, {A C F}}, by definition the negative border of L is $Bd^-(L)$={{B C}, {B F}, {D}, {E}}.

The algorithm proposed in [47] first generate the frequent itemsets of the increment portion Δ^+. A full scan of the whole dataset is required only if the negative border of the frequent itemsets expands, that is, if an itemset outside the negative border gets added to the frequent itemsets or its negative border. Even in such cases, it requires only one scan over the whole dataset. The possible drawback is that to compute the negative border closure may increase the size of the candidate set. However, a majority of those itemsets would have been present in the original negative border or frequent itemset. Only those itemsets which were not covered by the negative border need to be checked against the whole dataset. As a result, the size of the candidate set in the final scan could potentially be much smaller as compared to algorithm FUP.

Algorithm DELI (Difference Estimation for Large Itemsets)

To reduce the efforts of applying algorithm FUP_2 when database update occurs, the algorithm DELI which utilizes sampling techniques is proposed in [31]. Algorithm DELI can estimate the difference between the old and the new frequent itemsets. Only if the estimated difference is large enough, the update operation using algorithm FUP_2 has to be performed on the updated database to get the exact frequent itemsets.

Recall that in each iteration of algorithm FUP_2, all candidate itemsets in $Q_k (= C_k - P_k)$, i.e., previously infrequent itemsets, have to be further verified. For each candidate itemset $X \in Q_k$, if $X.support_{\Delta^+} \geq S_{min}$, the exact occurrence count of X in D^- has to be further identified. This data scan process is saved by algorithm DELI in the following way. By drawing m transactions from D^- with replacement to form the sample S, the support of itemset X in D^-, i.e., $\widehat{\sigma_X}$, can be estimated as

$$\widehat{\sigma_X} = \frac{T_x}{m} \cdot \left| D^- \right|,$$

where T_x is the occurrence count of X in S. With m sufficiently large, the normal distribution can be utilized to approximate the support of X in D^- with a $100(1-\alpha)\%$ confidence interval $[a_X, b_X]$:

$$a_X = \widehat{\sigma_X} - z_{\frac{\alpha}{2}} \cdot \sqrt{\frac{\widehat{\sigma_X}(|D^-| - \widehat{\sigma_X})}{m}}, \text{ and}$$
$$b_X = \widehat{\sigma_X} + z_{\frac{\alpha}{2}} \cdot \sqrt{\frac{\widehat{\sigma_X}(|D^-| - \widehat{\sigma_X})}{m}}$$

where $z_{\frac{\alpha}{2}}$ is the critical value such that the area under the standard normal curve beyond $z_{\frac{\alpha}{2}}$ is exactly $\frac{\alpha}{2}$. In other words, there is $100(1-\alpha)\%$ chance that the actual value of $X.support_{D^-}$ lies on the interval $[a_X, b_X]$. If the upper bound does not exceed the support threshold, i.e., $b_X < S_{min}$, itemset X is very likely to be infrequent and is thus dropped. Consequently, the resulting estimation for updated set of frequent itemsets $\widehat{L'_k}$ and the previous set of frequent itemsets L_k are compared to see if algorithm FUP_2 has to be resorted to obtain an accurate update.

Algorithms MAAP (Maintaining Association rules with Apriori Property) and PELICAN

Several other algorithms, including the MAAP algorithm [54], and the PELICAN algorithm [49], are proposed to solve the problem of incremental mining. Algorithm MAAP firstly finds the frequent itemset(s) of the largest size based on previously discovered frequent itemsets. If a k-itemset is found to be frequent, then all of its subsets are concluded to be frequent and are thus be added to the new set of frequent itemsets L'. This eliminates the need to compute some frequent itemsets of shorter sizes. The other frequent itemsets are then identified by following the levelwise style of itemset generation, i.e., from 1-itemsets to (k-1)-itemsets.

Both algorithms MAAP and PELICAN are similar to algorithm FUP_2, but they only focus on how to maintain maximum frequent itemsets when the database are updated. In other words, they do not consider non-maximum frequent itemsets, and therefore, the counts of non-maximum frequent itemsets cannot be calculated. The difference of these two algorithms is that MAAP calculates maximum frequent itemsets by Apriori-based framework while PELICAN calculates maximum frequent itemsets based on vertical database format and lattice decomposition. Since these two algorithms maintain maximum frequent itemsets only, the storage space and the processing time for performing each update can be thus reduced.

2.2 Partition-Based Algorithms for Incremental Mining

In contrast to the Apriori heuristic, the partition-based technique well utilizes the partitioning on the whole transactional dataset. Moreover, after the partitioning, it is understood that *if X is a frequent itemset in database D which is divided into n partitions p_1, p_2, ..., p_n, then X must be a frequent itemset in at least one of the n partitions.* Consequently, algorithms belonging

to this category work on each partition of data iteratively and gather the information obtained from the processing of each partition to generate the final (integrated) results.

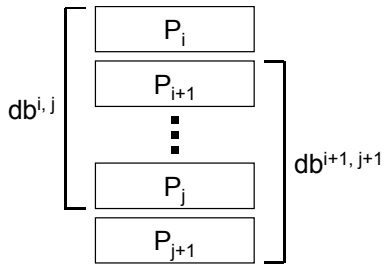

Fig. 11. Incremental mining for an ongoing time-variant transaction database

Consider a partitioned transaction database in Fig. 11. Note that $db^{i,j}$ is the part of the transaction database formed by a continuous region from partition P_i to partition P_j. Suppose we have conducted the mining for the transaction database $db^{i,j}$. As time advances, we are given the new data of P_{j+1}, and are interested in conducting an incremental mining against the new data. With the model of sliding window which is usually adopted in temporal mining, our interest is limited to mining the data within a fixed period of time rather than taking all the past data into consideration. As a result, the mining of the transaction database $db^{i+1,j+1}$ is called for.

Algorithm SWF (Sliding–Window Filtering)

By partitioning a transaction database into several partitions, algorithm SWF (sliding-window filtering) [28] employs a filtering threshold in each partition to deal with the candidate itemset generation. Under SWF, the cumulative information of mining previous partitions is selectively carried over toward the generation of candidate itemsets for the subsequent partitions. After the processing of a partition, algorithm SWF outputs a *cumulative filter*, which consists of a progressive candidate set of itemsets, their occurrence counts and the corresponding partial support required. Specifically, the cumulative filter produced in processing each partition constitutes the key component to realize the incremental mining.

The key idea of algorithm SWF is to compute a set of candidate 2-itemsets as close to L_2 as possible. The concept of this algorithm is described as follows. Suppose the database is divided into n partitions P_1, P_2, ..., P_n, and processed one by one. *For each frequent itemset I, there must exist some partition P_k such that I is frequent from partition P_k to P_n.* A list of 2-itemsets CF is maintained by algorithm SWF to track the possible frequent 2-itemsets. For

each partition P_i, algorithm SWF adds (locally) frequent 2-itemsets (together with its starting partition P_i and supports) that is not in CF and checks if the present 2-itemsets are continually frequent from its stating partition to the current partition. If a 2-itemset is no longer frequent, it is deleted from CF. However, if a deleted itemset is indeed frequent in the whole database, it must be frequent from some other partition P_j $(j > i)$, where we can add it to CF again.

It was shown that the number of the reserved candidate 2-itemsets will be close to the number of the frequent 2-itemsets. For a moderate number of candidate 2-itemsets, scan reduction technique [41] can be applied to generate all candidate k-itemsets. Therefore, one database scan is enough to calculate all candidate itemsets with their supports and to then determine frequent ones. In summary, the total number of database scans can be kept as small as two.

The flows of algorithm SWF is fully explored in the following example.

Example 7: Consider the transaction database in Fig. 6. A partitioned version is shown in Fig. 12.

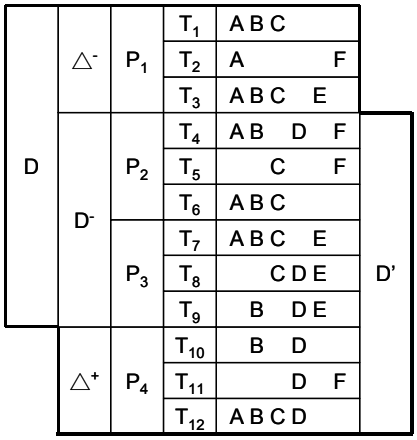

Fig. 12. A partitioned transaction database whose data records are identical to those in Fig. 6

Firstly, the goal is to generate all frequent itemsets in the original database D, that is defined as the *preprocessing* step in algorithm SWF. With S_{min}=40%, the generation of frequent 2-itemsets in each partition is shown in Fig. 13. After scanning the first three transactions, partition P_1, 2-itemsets {A B}, {A C} and {B C} are found to be frequent in P_1 and are thus potential candidates for D. It can be noted that each itemset shown in Fig. 13 has two attributes, i.e., "*Start*" contains the identity of the starting partition when the itemset was added, and "*Count*" contains the number of occurrences of this

itemset since it was added. In addition the filtering threshold is $\lceil 3 * 0.4 \rceil = 2$ which is the minimal count required in P_1.

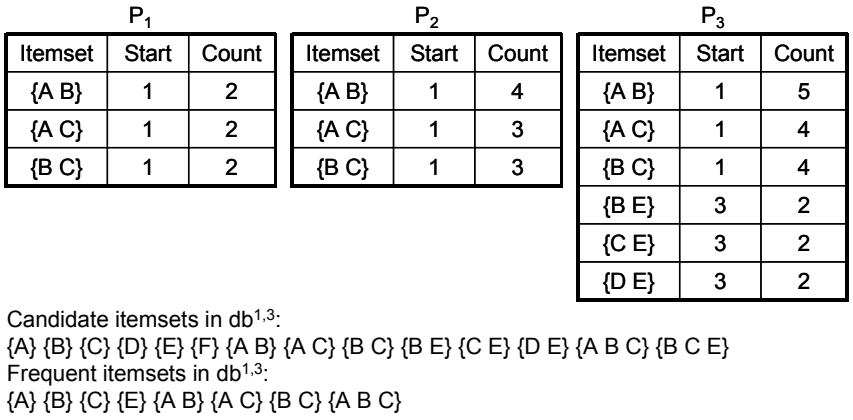

Fig. 13. Generation of frequent 2-itemsets in each partition using algorithm SWF

Note that the frequent itemsets discovered in each partition are carried to subsequent partitions. New itemsets could be added into the set of potential candidate itemsets, i.e., C_2, in a new partition while the counts of existing itemsets are increased and then be verified to see if these itemsets are still frequent from their starting partition to the current one. For example, there is no new 2-itemset added when processing P_2 since no extra frequent 2-itemsets in addition to the ones carried from P_1. Moreover, the counts for itemsets {A B}, {A C} and {B C} are all increased, making these itemsets frequent since their counts are no less than $\lceil 6 * 0.4 \rceil = 3$ when there are total six transactions, i.e., $P_1 \cup P_2$, are considered. Finally, after the analogous process for P_3, the resulting C_2={{A B}, {A C}, {B C}, {B E}, {C E}, {D E}} which are also shown in Fig. 13.

After generating C_2 from the first scan of database $db^{1,3}$, we employ the scan reduction technique and use C_2 to generate C_k $(k = 2, 3, ..., n)$, where C_n is the candidate *last*-itemsets. It can be verified that a C_2 generated by SWF can be used to generate the candidate 3-itemsets and its sequential C'_{k-1} can be utilized to generate C'_k. Clearly, a C'_3 generated from $C_2 * C_2$, instead of from $L_2 * L_2$, will have a size greater than $|C_3|$ where C_3 is generated from $L_2 * L_2$. However, since the $|C_2|$ generated by SWF is very close to the theoretical minimum, i.e., $|L_2|$, the $|C'_3|$ is not much larger than $|C_3|$. Similarly, the $|C'_k|$ is close to $|C_k|$. All C'_k can be stored in main memory, and we can find L_k $(k = 1, 2, ..., n)$ together when the second scan of the database $db^{1,3}$ is performed. Thus, only two scans of the original database $db^{1,3}$ are required in the preprocessing step. In addition, instead of recording all L_ks in main mem-

ory, we only have to keep C_2 in main memory for the subsequent incremental mining of an ongoing time variant transaction database. The itemsets of C_2 and L_2 in $db^{1,3}$ are also shown in Fig. 13 for references.

The merit of SWF mainly lies in its incremental procedure. As depicted in Fig. 12, the mining database will be moved from $db^{1,3}$ to $db^{2,4}$. Thus, some transactions, i.e., t_1, t_2, and t_3, are *deleted* from the mining database and other transactions, i.e., t_{10}, t_{11}, and t_{12}, are *added*. For ease of exposition, this incremental step can also be divided into three sub-steps: (1) generating C_2 in $D^- = db^{1,3} - \Delta^-$, (2) generating C_2 in $db^{2,4} = D^- + \Delta^+$ and (3) scanning the database $db^{2,4}$ only once for the generation of all frequent itemsets L_k. The flows of these steps are presented in Fig. 14 where shaded itemsets are identified to be infrequent and are thus discarded during the mining process. Specifically, the counts of existing itemsets in Δ^- are firstly subtracted, and the processing of Δ^+ is then trivial and easy to achieve. After the new potential candidates for $D'(= db^{2,4})$ are generated, algorithm SWF can obtain new frequent itemsets with scanning $db^{2,4}$ only once.

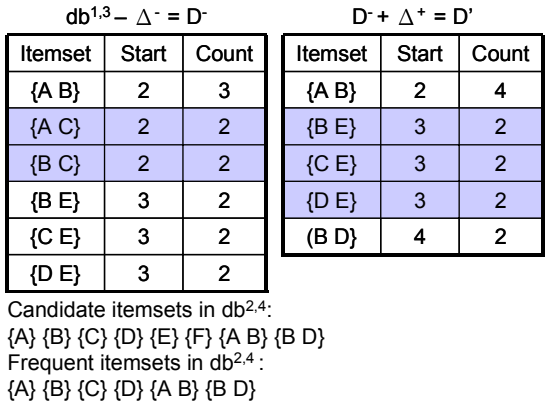

Fig. 14. Incremental mining using algorithm SWF where shaded itemsets are identified to be infrequent when Δ^- is deleted or Δ^+ is added

The advantages of algorithm SWF not only can be fully exploited in the problem of incremental mining, but also are beneficial to the development of weighted mining [29].

Algorithms FI_SWF and CI_SWF

In [8], the algorithm SWF is extended by incorporating previous discovered information. Two enhancements are proposed, namely the algorithm FI_SWF (SWF with Frequent Itemset) and the algorithm CI_SWF (SWF with Candidate Itemset). These two algorithms reuse either the frequent itemsets or

the candidate itemsets of previous mining task to reduce the number of new candidate itemsets. Therefore, the execution time for both algorithms can be improved and is thus better than that of algorithm SWF.

Consider the example provided earlier, it is easily observed that there are several (candidate or frequent) itemsets are identical for both the preprocessing procedure, and the incremental procedure. Therefore, if the previous mining result, i.e., the counts of the frequent itemsets, is incorporated, new counts can be obtained by only scanning the changed portions of the whole dataset in the incremental procedure.

2.3 Pattern Growth Algorithms for Incremental Mining

Both the Apriori-based algorithms and the partition-based algorithms aim at the goal of reducing the number of scans on the entire dataset when updates occur. Generally speaking, the updated portions, i.e., Δ^- and Δ^+, could be scanned several times during the levelwise generation of frequent itemsets in works belonging to these two categories. On the contrary, the algorithm FP-growth [25] (frequent pattern growth) along with the FP-tree structure adopts the *divide-and-conquer* strategy to mine association rules. The major difficulties that FP-tree cannot be directly applied to the problem of incremental mining are eased in some recent works [15, 18]. These works, in general, utilize alternative forms of FP-tree to store required data to achieve the goal of avoiding the overhead resulting from extra database scans.

Algorithms DB-tree and PotFp-tree (Potential Frequent Pattern)

In [18], two alternative forms of FP-tree are proposed to solve the problem of incremental mining. One is the algorithm DB-tree, which stores all the items in an FP-tree rather than only frequent 1-itemsets in the database. Besides, the construction of a DB-tree is exactly the same way as that of a FP-tree. Consequently, the DB-tree can be seen as an FP-tree with $S_{min} = 0$, and is thus a generalized form of FP-tree. When new transactions are added, corresponding branches of the DB-tree could be adjusted or new branches may be created. On the other hand, when old transactions are deleted, corresponding branches are also adjusted or removed. This retains the flexibility to accommodate the FP-tree to database changes when performing incremental mining. However, since the whole dataset being considered could be quite large, a much more space could be needed to maintain this DB-tree structure even a high compression is made by the nature of tree projection. This drawback may cause the problem of insufficient memory even more severe when the size of the DB-tree is far above the memory capacity.

The other algorithm proposed in [18] is the PotFp-tree, which stores only some potentially frequent items in addition to the frequent 1-itemsets at present. A tolerance parameter (or alternatively the watermark [25]) t is proposed to decide if an item is with the potential. Namely, for items with

supports s where $t \leq s \leq S_{min}$ are defined to be potentially frequent items. Therefore, the need to scan the whole old database in order to update the FP-tree when updates occur is likely to be effectively reduced. Generally speaking, the PotFp-tree is seeking for the balance of required extra storage and possibility of re-scanning the dataset.

It is noted that the FP-tree is a subset of either the DB-tree or the PotFp-tree. To mine frequent itemsets, the FP-tree is firstly projected from either the DB-tree or the PotFp-tree. The frequent itemsets are then extracted from the FP-tree in the way described in [25]. The flows of utilizing algorithms DB-tree and PotFp-tree to mine association rules incrementally are best understood in the following example.

Example 8: Consider the transaction database in Fig. 6. By utilizing the algorithm DB-tree, the resulting DB-tree is shown in Fig. 15 where all the items, i.e., both the frequent ones (circles with solid lines) and the infrequent ones (circles with dotted lines), are included. It is noted that the corresponding FP-tree (which is shown in Fig. 5 earlier) forms a subset in the DB-tree and is located in the top portion of the DB-tree. For a clearer presentation, the node-links of individual items are ignored in Fig. 15.

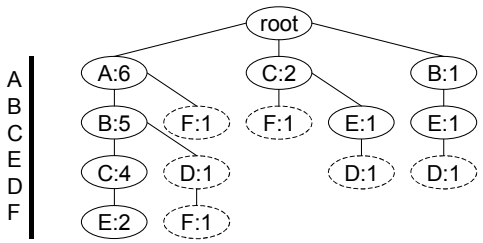

Fig. 15. Construction of the DB-tree based on the transaction database D in Fig. 6

On the other hand, by setting the tolerance t=30% which is slightly lower than S_{min}, the PotFp-tree can be also built by including items with supports no less than t. The resulting PotFp-tree happens to be identical to the DB-tree in Fig. 15.

After constructing either the DB-tree or the PotFp-tree, the generation of frequent itemsets is very simple. Firstly, the corresponding FP-tree is extracted. Then, the frequent itemsets can be then discovered from the FP-tree in the way as described in earlier section.

When the updated database D' in Fig. 6 is being considered, some old transactions have to be removed from the DB-tree and some new transactions have to be included in the DB-tree. Note that the operations of removal and inclusion could cause the ordering of some items to be changed since some originally frequent items become infrequent while some originally infrequent

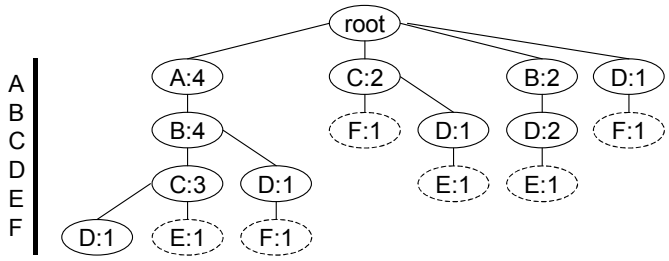

Fig. 16. Construction of the DB-tree based on the transaction database D' in Fig. 6

items become frequent. For example, the ordering of items D and E in Fig. 16 are exchanged since item D becomes frequent while item E becomes infrequent. On the other hand, it is not so necessary to change the item order in the DB-tree when only the ordering of supports of frequent itemsets changes.

Algorithm FELINE (FrEquent/Large patterns mINing with CATS trEe)

In [15], another alternative form of FP-tree is proposed to aim at the problem of incremental mining. Namely, the CATS tree (compressed and arranged transaction sequences tree) is with several common properties of FP-tree. Also, the CATS tree and the DB-tree are very alike since they both store all the items no matter they are frequent or not. This feature enables the CATS tree to be capable of avoiding re-scans of databases when updates occur. However, the construction of the CATS tree is different to that of an FP-tree and a DB-tree. The items along a path in the CATS-tree can be re-ordered to achieve locally optimized. Specifically, the FP-tree is built based on the ordering of global supports of all frequent items, while the CATS-tree is built based on the ordering of local supports of items in its path. Consequently, the CATS-tree is sensitive to the ordering of input transactions, making the CATS-tree not optimal since no preliminary analysis is done before the tree construction. This in turns can reduce the data scan required to only once, showing the advantage of this algorithm.

Example 9: Consider the transaction database in Fig. 6. The construction of CATS-tree for the nine transactions in D is shown in Fig. 17.

By removing Δ^- (T_1~T_3) and adding Δ^+ (T_{10}~T_{12}), the incremental modification of the CATS-tree is shown in Fig. 18. Since no specific ordering is required when there is no conflict among items in a path, the CATS-tree can be built and modified easily.

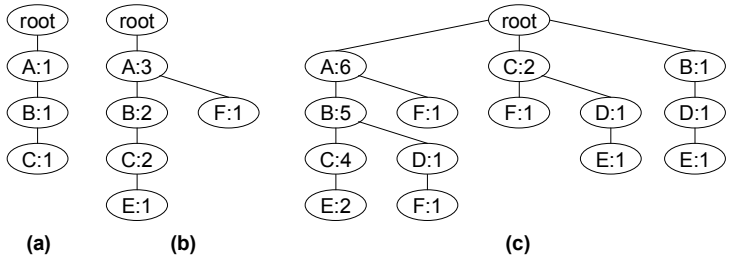

Fig. 17. Construction of the CATS-tree: (a) T_1 is added; (b) T_2 and T_3 are added; (c) T_4~T_9 are added

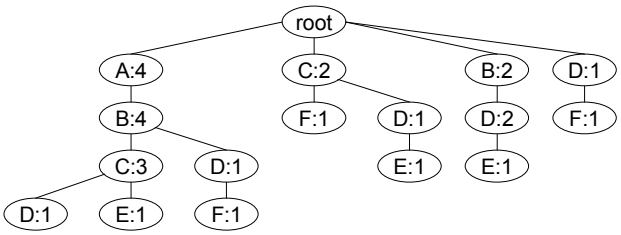

Fig. 18. The modified CATS-tree when database updates occur

3 Mining Frequent Patterns from Data Streams

In several emerging applications, data is in the form of continuous *data streams*, as opposed to finite stored databases. Examples include stock tickers, network traffic measurements, web logs, click streams, data captured from sensor networks and call records. Consequently, mining association rules (or frequent patterns) incrementally in such a data stream environment could encounter more challenges than that in a static database with occasional updates. To explore related issues, several techniques proposed in recent works are summarized in this section.

3.1 Mining from Online Transaction Flows

For data stream applications, the volume of data is usually too huge to be stored on permanent devices or to be scanned thoroughly for more than once. It is hence recognized that both approximation and adaptability are key ingredients for executing queries and performing mining tasks over rapid data streams. With the computation model presented in Fig. 19 [20], a stream processor and the synopsis maintenance in memory are two major components for generating results in the data stream environment. Note that a buffer can be optionally set for temporary storage of recent data from data streams.

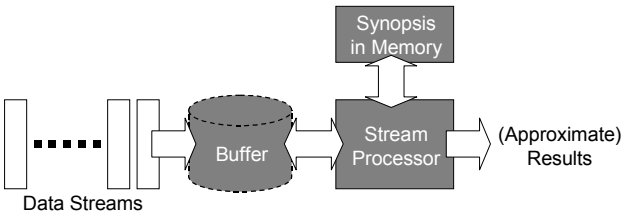

Fig. 19. Computation model for data streams

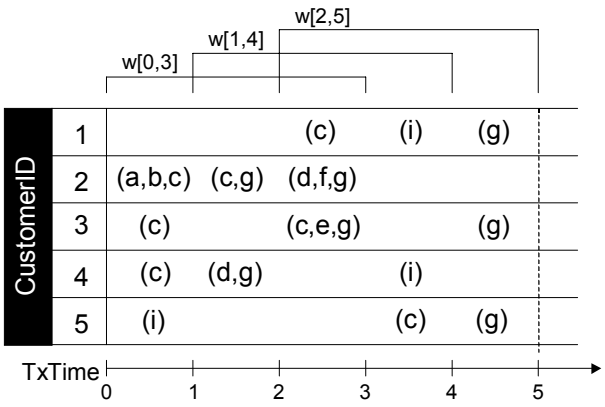

Fig. 20. An example of online transaction flows

For time-variant databases, there is a strong demand for developing an efficient and effective method to mine frequent patterns [17]. However, most methods which were designed for a traditional database cannot be directly applied to a dynamic data stream due not only to the high complexity of mining temporal patterns but also to the pass-through nature of data streams. Without loss of generality, a typical market-basket application is used here for illustrative purposes. The transaction flow in such an application is shown in Fig. 20 where items a to g stand for items purchased by customers. For example, the third customer bought item c during time t=[0, 1), items c, e and g during t=[2, 3), and item g during t=[4, 5). It can be seen that in such a data stream environment it is intrinsically very difficult to conduct the frequent pattern identification due to the limited time and space constraints.

3.2 Approximate Frequency Counts

As is mentioned in Section 1 in this chapter, the critical step to discover association rules lies in the frequency counting for all frequent itemsets. To obtain precise supports for each itemset, the fastest algorithms to date must employ

two data scans. However, for the data stream environment, it is required to have only one data scan for online (and incremental) maintenance of association rules. Due to this limitation in processing data streams, two algorithms are proposed in [35] to generate approximate frequency counts for itemsets. One algorithm is based on the sampling technique to obtain a probabilistic error bound while the other is based on the data partitioning technique to obtain a deterministic error bound.

The approximation generated from both algorithms, i.e., the sticky sampling and the lossy counting, are with the following guarantees, i.e., the ϵ-deficient synopsis:
1. There are *no false positives*, i.e., all itemsets whose true frequency exceed the minimum support constraint are output.
2. No itemsets whose true frequency is less than (1-ϵ)S_{min} is output where ϵ is a error parameter.
3. Estimated frequencies are less than the true frequencies by at most ϵ (in percentage).

Note that the error ϵ is a user-specified parameter in both algorithms. In addition, the input for both algorithms is assumed to be a data stream of singleton items at first. The lossy counting algorithm is then extended to handle data streams of transactions which contains a set of items.

Sticky Sampling Algorithm

This algorithm is probabilistic with a user-specified parameter δ, i.e., probability of failure that ϵ-deficient synopsis cannot be guaranteed. The elements of the input stream is processed one by one and the current length of the stream is denoted by N.

The data structure maintained is a set S of entries of the form (e, f), where f estimates the frequency of an element e belonging to the stream. Initially, S is empty, and the sampling rate $r = 1$ is set. For each incoming element e, if an entry for e already exists in S, the corresponding frequency f is increased by one. Otherwise, if the element is selected (with probability $\frac{1}{r}$) by sampling, an entry $(e, 1)$ is added to S.

The sampling rate r varies over the lifetime of a stream. Let $t = \frac{1}{\epsilon} log(\frac{1}{S \times \delta})$. The first $2t$ elements are sampled at rate $r = 1$, the next $2t$ elements are sampled at rate $r = 2$, the next $4t$ elements are sampled at rate $r = 4$, and so on. Whenever the sampling rate changes, the entries in S is scanned and updated through a coin-tossing process, i.e., f is diminished by one for every unsuccessful outcome until the repeatedly toss is successful. If f becomes 0 during this process, the corresponding entry is deleted from S.

When a user requests the list of frequent items, the entries in S with $f \geq N(1 - \epsilon)S_{min}$ are output. It is proved in [35] that true supports of these frequent items are underestimated by at most ϵ with probability $1 - \delta$.

Intuitively, S sweeps over the stream like a magnet, attracting all elements which already have an entry in S. The sample rate r increases logarithmically

proportional to the size of the stream. The most advantage of this algorithm is that the space complexity is independent of the current length of the stream.

Lossy Counting Algorithm

In contrast to the sticky sampling algorithm, the incoming data stream is conceptually divided into buckets of width $w = \lceil \frac{1}{\epsilon} \rceil$ transactions in the lossy counting algorithm. Each bucket is labeled with a bucket id, starting from 1. The data structure maintained is a set S of entries of the form $(e, f, \triangle)$, where f estimates the frequency of an element e belonging to the stream and $\triangle$ is the maximum possible error in f. Initially, S is empty. Whenever a new element e arrives, if the corresponding entry already exists, its frequency f is increased by one. Otherwise, a new entry of the form $(e, 1, b_{current} - 1)$ is created where $b_{current}$ denotes the current bucket id $(= \lceil \frac{N}{w} \rceil)$. The idea behind the lossy counting algorithm is to set a criterion that *a tracked item should at least occur once (in average) in each bucket*. Consequently, the maximum possible error $\triangle = b_{current} - 1$ for an entry added in bucket $b_{current}$. At bucket boundaries, S is pruned by deleting entries satisfying $f + \triangle \leq b_{current}$.

When a user requests the list of frequent items, the entries in S with $f \geq N(1 - \epsilon)S_{min}$ are output. It is proved in [35] that true supports of these frequent items are underestimated by at most ϵ.

To further discover frequent itemsets from data streams of transactions, the lossy counting algorithm is also extended in [35]. A significant difference is that the input stream is not processed transaction by transaction. Instead, the available main memory is filled with as many buckets of transactions as possible, and such a *batch* of transactions is processed together. If the number of buckets in main memory in the current batch being processed is β, then the new entry $(set, 1, b_{current} - \beta)$ is created when the itemset *set* is not present in S. Other methods are nearly identical to the original ones for streams of singleton items. Similarly, when a user requests the list of frequent itemsets, the entries in S with $f \geq N(1 - \epsilon)S_{min}$ are output.

3.3 Finding Recent Frequent Itemsets Adaptively

In order to differentiate the information of recently generated transactions from the obsolete information of old transactions, a weighting scheme and the *estDec* method for discovering recent frequent itemsets from data streams are developed in [9]. A delay rate d $(0 < d < 1)$ is introduced in this weighting scheme that the weight value of an old transaction T_k with id k (starting from 1) is $d^{(current-k)}$ where *current* is the current transaction id being processed. Consequently, when the first transaction is looked up, the total number of transactions is obviously 1 since there is no previous transaction whose weight should be decayed. As time advances, when k transactions are processed, the total number of transactions can be expressed by

$$|D|_k = d^{k-1} + d^{k-2} + \cdots + d + 1 = \frac{1 - d^k}{1 - d}.$$

Also, it is obvious that

$$|D|_k = \begin{cases} 1 & \text{if } k = 1 \\ |D|_{k-1} \times d + 1 & \text{if } k \geq 2 \end{cases}.$$

Moreover, the *estDec* method proposed in [9] maintains a monitoring lattice to track frequent itemsets. Each node in this lattice is an entry of the form $(cnt, err, MRtid)$ for a corresponding itemset X, where cnt is the occurrence count, err is the maximum error count and $MRtid$ is the transaction id of the most recent transaction containing X. The critical step in this approach is to update occurrence counts for tracked itemsets. If an itemset X is contained in the current transaction T_k and the previous entry is $(cnt_{pre}, err_{pre}, MRtid_{pre})$, its current entry $(cnt_k, err_k, MRtid_k)$ can be updated as

$$\begin{aligned} cnt_k &= cnt_{pre} \times d^{(k - MRtid_{pre})} + 1, \\ err_k &= err_{pre} \times d^{(k - MRtid_{pre})}, \text{ and} \\ MRtid_k &= k. \end{aligned}$$

Therefore, when the updated support of an itemset becomes less than the threshold, i.e., $\frac{cnt_k}{|D|_k} < S_{min}$, the entry corresponding to this itemset is dropped. The threshold for pruning here can be adjusted to be slightly less than the minimum support to reduce the overhead of reinserting a previously deleted itemset.

For inserting newly discovered itemsets, another feature of the *estDec* method is that only when all of the (n-1)-subsets of an n-itemset are frequent, this n-itemset is able to become a candidate. In other words, a frequent itemset may not be identified immediately since all its subsets have to become frequent first, leading to a possible delayed insertion.

3.4 A Scheme for Mining Temporal Patterns

In a temporal database, frequent patterns are usually targets of mining tasks. In addition to the mining of association rules, similar techniques can be extended to facilitate the mining of other types of temporal patterns.

Mining of Temporal Patterns

Prior works have developed several models of temporal patterns, including the inter-transaction association rule [34], the causality rule [30], the episode [37] and the sequential pattern [4]. Note that the very difference among the above temporal patterns lies the ordering of occurrences. Mining of sequences corresponds to the one with strict order of events, while mining inter-transaction

associations corresponds to the one without limitation on order of events. Between these two extremes, mining of causalities and episodes mainly emphasizes the ordering of triggering events and consequential events. Although the mining procedures may vary when being applied to discover different types of temporal patterns, a typical Apriori framework is commonly adopted. By utilizing the downward closure property in this framework [53], a fundamental issue of mining frequent temporal patterns is the frequency counting of patterns.

In many applications, a time-constraint is usually imposed during the mining process to meet the respective constraint. Specifically, the sliding window model is introduced here, i.e., data expires after exactly N time units after its arrival where N is the user-specified window size. Consequently, a temporal pattern is frequent if its support, i.e., occurrence frequency, in the current window is no less than the threshold.

Support Framework for Temporal Patterns

To evaluate the importance of a temporal pattern, the support, i.e., occurrence frequency, is a metric commonly used. However, the definition of support for a pattern may vary from one application to another. Consider again the market-basket database as an example. In mining sequential patterns [4], all the transactions of a customer can be viewed as a sequence together and the support for a sequential pattern is the fraction of customers whose purchasing sequences contain that pattern. Analogously, we have the model of frequency counting in mining causality rules [30]. On the other hand, in mining inter-transaction association rules [34], the repetitive occurrences of a pattern from an identical customer are counted cumulatively. Moreover, when the sliding window constraint is introduced in mining episodes [37], the support is defined to be the fraction of windows in which an episode occurs.

To deal with data streams, problems arise due to different support definitions. Specifically, since it is not possible to store all the historical data in the memory, to identify repetitive occurrences of a pattern is difficult. As a result, it is very important to properly formulate the support of temporal patterns. With the sliding window model, the support or the occurrence frequency of a temporal pattern X at a specific time t is denoted by *the ratio of the number of customers having pattern X in the current time window to the total number of customers.*

Example 10: Given the window size N=3, three sliding windows, i.e., w[0,3], w[1,4] and w[2,5], are shown in Fig. 20 for the transaction flows. For example, according to the support definition, supports of the inter-transaction itemset $\{c, g\}$ from TxTime t=1 to t=5 are obtained as in Table 1. Accordingly, the support variations can be presented as a time series as shown in Fig. 21. For simplicity, the total number of customers is a constant in this example, and could be a variable as time advances in real applications.

Table 1: The support values of the inter-transaction itemset $\{c, g\}$

TxTime		Occurrence(s) of $\{c, g\}$	Support
t=1	w[0,1]	none	0
t=2	w[0,2]	CustomerID={2, 4}	2/5=0.4
t=3	w[0,3]	CustomerID={2, 3, 4}	3/5=0.6
t=4	w[1,4]	CustomerID={2, 3}	2/5=0.4
t=5	w[2,5]	CustomerID={1, 3, 5}	3/5=0.6

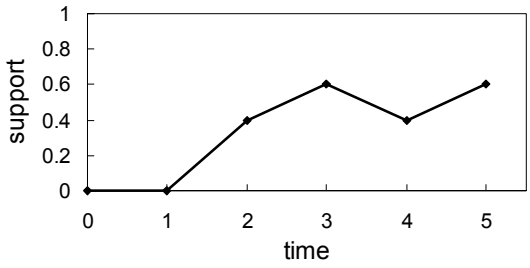

Fig. 21. Support variations of the inter-transaction itemset $\{c, g\}$

Algorithm FTP-DS (Frequent Temporal Patterns of Data Streams)

Note that although the approaches proposed in [35] work successfully for counting supports of singleton items, as the number of items increases, the rapidly increasing number of temporal patterns can cause problems of prohibitive storage and computing overheads. Explicitly, if the lossy counting scheme proposed in [35] is adopted, patterns with supports no less than ϵ are maintained during the mining process to guarantee the error range to be within ϵ. However, since the threshold ϵ, whose value could be one-tenth of S_{min}, is usually too small to filter out uninteresting patterns, the storage space could be quite large.

To address this point, the algorithm FTP-DS is developed in [46]. With the sliding window model employed, only the occurrences of singleton items are being counted in the first time window. After the counting iteration, frequent items which have supports no less than the specified threshold are identified. These frequent items can be joined to generate candidate patterns of size two, which are then being counted in later iterations. After some patterns of size two are identified frequent, the candidate patterns of size three are generated and counted subsequently. As a result, longer candidate patterns are gradually generated, counted and verified to be frequent during the counting

iterations. From the downward closure property [53], it follows that only patterns whose sub-patterns are all frequent are taken as candidates and to be counted subsequently.

Example 11: Given the support threshold S_{min}=40%, the window size N=3 and the transaction flows in Fig. 20, suppose the frequent inter-transaction associations are being generated. Since the temporal order is not required for inter-transaction associations, we have the frequent temporal itemset generation shown in Table 2. The support calculation of each itemset is the same as the process in Table 1. The averaged support value is represented by (accumulated supports over windows)/(number of recorded windows) in Table 2 where only itemsets with supports no less than S_{min}=40% are listed. In addition, frequent itemsets generated in previous time window are used to generate longer candidates to be examined later. For example, according to Definition 1, the supports of itemset {d} during t=1 to t=5 are 0, 0.2, 0.4, 0.4 and 0.2, respectively. Not until t=3 does the support value satisfy the threshold, meaning that itemset {d} is being tracked since t=3 as shown in Table 2(c) and Table 2(d). However, the averaged support of itemset {d} is (0.4+0.4+0.2)/3=1.0/3 which is less than the S_{min}=40%, making this itemset discarded in Table 2(e). Moreover, the inclusion of itemset {d} at t=3 results in the generation of related candidate itemsets, i.e., {c,d} and {d,g}, to be examined at t=4. However, only itemset {d, g} satisfies the support threshold and is included in Table 2(d).

Table 2: Generation of frequent temporal itemsets (S_{min}=40%)

t=1	
{c}	0.6/1

(a)

t=2	
{c}	1.2/2
{g}	0.4/1

(b)

t=3	
{c}	2/3
{d}	0.4/1
{g}	1/2
{c,g}	0.6/1

(c)

t=4	
{c}	2.8/4
{d}	0.8/2
{g}	1.6/3
{i}	0.4/1
{c,g}	1/2
{d,g}	0.4/1

(d)

t=5	
{c}	3.4/5
{g}	2.4/4
{i}	0.8/2
{c,g}	1.6/3

(e)

It can be seen that this approach can generate patterns of various lengths as time advances. However, as pointed out earlier, since a pattern is not taken as a candidate to accumulate its occurrence counts before all its subsets are found frequent, the phenomenon of *delayed pattern recognition* exists, i.e., some patterns are recognized with delays due to the candidate forming process in the data stream. For example, since items c and g are not both identified frequent until t=2, the candidate itemset $\{c, g\}$ is generated and counted at t=3. However, it can be verified from Table 1 that $\{c, g\}$ is actually frequent at t=2. Therefore, a delay of one time unit is introduced for discovering

this itemset $\{c, g\}$. It is worth mentioning that only long transient frequent patterns could be neglected in this pattern generation process. As time advances, patterns with supports near the threshold will be further examined and identified to be frequent if so qualified.

With a support threshold S_{min} to filter out uninteresting patterns, only new patterns whose frequencies in the current time unit meet this threshold are being recorded. Supports of existing patterns, i.e., patterns which were already being recorded in previous time unit, are updated according to their support values in the current time unit. Note that, as time advances, patterns whose averaged supports fall below the threshold are removed from the records. Therefore, only frequent patterns are monitored and recorded. In practice, since a frequent pattern is not always with a very steady frequency, the above mentioned removal can be delayed to allow a pattern whose statistics are already recorded to stay in the system longer with an expectation that this pattern will become frequent again soon. This will be an application-dependent design alternative.

4 Concluding Remarks

The mining of association rules among huge amounts of transactional data can provide very valuable in information on customer buying behavior, and thus improve the quality of business decisions. This *market basket analysis* is a crucial area of application when performing data mining techniques. Specifically, the mining of association rules is a two-step process where the first step is to find all *frequent* itemsets satisfying the minimum support constraint, and the second step is to generate association rules satisfying the minimum confidence constraint from the frequent itemsets. Since to identify the frequent itemsets is of great computational complexity, usually the problem of mining association rules can be reduced to the problem of discovering frequent itemsets.

According to the numerous works on mining association rules, the approaches utilized for efficiently and effectively discovering frequent itemsets can be categorized into three types. Namely, they are the *Apriori-based* algorithms, the *partition-based* algorithms and the *pattern growth* algorithms. The Apriori-based algorithms adopt a levelwise style of itemset generation while the database may need to be scanned for a few times. The partition-based algorithms firstly divide the whole database into several partitions and then perform the mining task on each partition separably and iteratively. The pattern growth algorithms usually project the whole database into a highly compressed tree structure and then utilize the divide-and-conquer strategy to mine required frequent itemsets.

Recent important applications have called for the need of incremental mining. This is due to the increasing use of the record-based databases whose data are being continuously added. As time advances, old transactions may become

obsolete and are thus discarded from the database of interests to people. For the purpose of maintaining the discovered association rules, some previously valid rules may become invalid while some other new rules may show up. To efficiently reflect these changes, how to utilize discovered information well is undoubtedly an important issue. By extending the techniques used in mining ordinary association rules, several works reached a great achievement by developing either the Apriori-based algorithms, the partition-based algorithms and the pattern-growth algorithms.

Moreover, a challenging and interesting area of conducting the mining capabilities in a data stream environment is becoming popular in data mining society. To further extend the concept of mining and maintaining association rules from data streams, some recent works are also included in this chapter.

References

1. R. C. Agarwal, C. C. Aggarwal, and V. V. V. Prasad. A Tree Projection Algorithm for Generation of Frequent Itemsets. *Journal of Parallel and Distributed Computing (Special Issue on High Performance Data Mining)*, 61(3):350–371, 2001.
2. R. Agrawal, T. Imielinski, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, May 1993.
3. R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules in Large Databases. *Proceedings of the 20th International Conference on Very Large Data Bases*, pages 478–499, September 1994.
4. R. Agrawal and R. Srikant. Mining Sequential Patterns. *Proceedings of the 11th International Conference on Data Engineering*, pages 3–14, March 1995.
5. J. M. Ale and G. Rossi. An Approach to Discovering Temporal Association Rules. *Proceedings of the 2000 ACM Symposium on Applied Computing*, pages 294–300, March 2000.
6. N. F. Ayan, A. U. Tansel, and M. E. Arkun. An Efficient Algorithm to Update Large Itemsets with Early Pruning. *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 287–291, August 1999.
7. S. Brin, R. Motwani, J. D. Ullman, and S. Tsur. Dynamic Itemset Counting and Implication Rules for Market Basket Data. *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, pages 255–264, May 1997.
8. C.-H. Chang and S.-H. Yang. Enhancing SWF for Incremental Association Mining by Itemset Maintenance. *Proceedings of the 7th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, April 2003.
9. J. H. Chang and W. S. Lee. Finding Recent Frequent Itemsets Adaptively over Online Data Streams. *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 487–492, August 2003.
10. M.-S. Chen, J. Han, and P. S. Yu. Data Mining: An Overview from Database Perspective. *IEEE Transactions on Knowledge and Data Engineering*, 8(6):866–883, December 1996.

11. M.-S. Chen, J.-S. Park, and P. S. Yu. Efficient Data Mining for Path Traversal Patterns. *IEEE Transactions on Knowledge and Data Engineering*, 10(2):209–221, April 1998.
12. X. Chen and I. Petr. Discovering Temporal Association Rules: Algorithms, Language and System. *Proceedings of the 16th International Conference on Data Engineering*, 2000.
13. D. Cheung, J. Han, V. Ng, and C. Y. Wong. Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique. *Proceedings of the 12th International Conference on Data Engineering*, pages 106–114, February 1996.
14. D. Cheung, S. D. Lee, and B. Kao. A General Incremental Technique for Updating Discovered Association Rules. *Proceedings of the Fifth International Conference On Database Systems for Advanced Applications*, pages 185–194, April 1997.
15. W. Cheung and O. R. Zaiane. Incremental Mining of Frequent Patterns without Candidate Generation or Support Constraint. *Proceedings of the 7th International Database Engineering and Application Symposium*, July 2003.
16. E. Cohen, M. Datar, S. Fujiwara, A. Gionis, P. Indyk, R. Motwani, J. D. Ullman, and C. Yang. Finding Interesting Associations without Support Pruning. *IEEE Transactions on Knowledge and Data Engineering*, 13(1):64–78, January/February 2001.
17. G. Das, K.-I. Lin, H. Mannila, G. Renganathan, and P. Smyth. Rule Discovery from Time Series. *Proceedings of the 4th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 16–22, August 1998.
18. C. I. Ezeife and Y. Su. Mining Incremental Association Rules with Generalized FP-Tree. *Proceedings of the 15th Canadian Conference on Artificial Intelligence*, May 2002.
19. V. Ganti, J. Gehrke, and R. Ramakrishnan. Mining Data Streams under Block Evolution. *SIGKDD Explorations*, 3(2):1–10, January 2002.
20. M. N. Garofalakis, J. Gehrke, and R. Rastogi. Querying and Mining Data Streams: You Only Get One Look. *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, June 2002.
21. J. Han, G. Dong, and Y. Yin. Efficient Mining of Partial Periodic Patterns in Time Series Database. *Proceeding of the 15th International Conference on Data Engineering*, pages 106–115, March 1999.
22. J. Han and Y. Fu. Discovery of Multiple-Level Association Rules from Large Databases. *Proceedings of the 21th International Conference on Very Large Data Bases*, pages 420–431, September 1995.
23. J. Han, L. V. S. Lakshmanan, and R. T. Ng. Constraint-Based, Multidimensional Data Mining. *COMPUTER (Special Issue on Data Mining)*, pages 46–50, 1999.
24. J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M. C. Hsu. FreeSpan: Frequent Pattern-Projected Sequential Pattern Mining. *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 355–359, August 2000.
25. J. Han, J. Pei, and Y. Yin. Mining Frequent Patterns without Candidate Generation. *Proceedings of the 2000 ACM-SIGMOD International Conference on Management of Data*, May 2000.

26. J. Hipp, U. Guntzer, and G. Nakhaeizadeh. Algorithms for Association Rule Mining - A General Survey and Comparison. *SIGKDD Explorations*, 2(1):58–64, July 2000.
27. L. V. S. Lakshmanan, R. Ng, J. Han, and A. Pang. Optimization of Constrained Frequent Set Queries with 2-Variable Constraints. *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, pages 157–168, June 1999.
28. C.-H. Lee, C.-R. Lin, and M.-S. Chen. Sliding-Window Filtering: An Efficient Algorithm for Incremental Mining. *Proceeding of the ACM 10th International Conference on Information and Knowledge Management*, November 2001.
29. C.-H. Lee, J.-Z. Oh, and M.-S. Chen. Progressive Weighted Miner: An Efficient Method for Time-Constraint Mining. *Proceedings of the 7th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, April 2003.
30. C.-H. Lee, P. S. Yu, and M.-S. Chen. Causality Rules: Exploring the Relationship between Triggering and Consequential Events in a Database of Short Transactions. *Proceedings of the 2nd SIAM International Conference on Data Mining*, April 2002.
31. S. D. Lee, D. W. Cheung, and B. Kao. Is Sampling Useful in Data Mining? A Case Study in the Maintenance of Discovered Association Rules. *Data Mining and Knowledge Discovery*, 2(3):233–262, 1998.
32. J. L. Lin and M. H. Dunham. Mining Association Rules: Anti-Skew Algorithms. *Proceedings of the 14th International Conference on Data Engineering*, pages 486–493, 1998.
33. B. Liu, W. Hsu, and Y. Ma. Mining Association Rules with Multiple Minimum Supports. *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, August 1999.
34. H. Lu, J. Han, and L. Feng. Stock Movement Prediction and N-Dimensional Inter-Transaction Association Rules. *Proceedings of the 1998 ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery*, pages 12:1–12:7, June 1998.
35. G. S. Manku and R. Motwani. Approximate Frequency Counts over Streaming Data. *Proceedings of the 28th International Conference on Very Large Data Bases*, pages 346–357, August 2002.
36. H. Mannila, H. Toivonen, and A. I. Verkamo. Efficient Algorithms for Discovering Association Rules. *Proceedings of AAAI Workshop on Knowledge Discovery in Databases*, pages 181–192, July 1994.
37. H. Mannila, H. Toivonen, and A. I. Verkamo. Discovery of Frequent Episodes in Event Sequences. *Data Mining and Knowledge Discovery*, 1(3):259–289, 1997.
38. A. Mueller. Fast Sequential and Parallel Algorithms for Association Rule Mining: A Comparison. *Technical Report CS-TR-3515, Dept. of Computer Science, Univ. of Maryland, College Park, MD*, 1995.
39. R. T. Ng and J. Han. Efficient and Effective Clustering Methods for Spatial Data Mining. *Proceedings of the 20th International Conference on Very Large Data Bases*, pages 144–155, September 1994.
40. J.-S. Park, M.-S. Chen, and P. S. Yu. An Effective Hash-Based Algorithm for Mining Association Rules. *Proceedings of the ACM-SIGMOD International Conference on Management of Data*, May 1995.
41. J.-S. Park, M.-S. Chen, and P. S. Yu. Using a Hash-Based Method with Transaction Trimming for Mining Association Rules. *IEEE Transactions on Knowledge and Data Engineering*, 9(5):813–825, October 1997.

42. J. Pei and J. Han. Can We Push More Constraints into Frequent Pattern Mining? *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, August 2000.
43. J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M. C. Hsu. PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth. *Proceedings of the 17th International Conference on Data Engineering*, 2001.
44. A. Savasere, E. Omiecinski, and S. Navathe. An Efficient Algorithm for Mining Association Rules in Large Databases. *Proceedings of the 21th International Conference on Very Large Data Bases*, pages 432–444, September 1995.
45. R. Srikant and R. Agrawal. Mining Generalized Association Rules. *Proceedings of the 21th International Conference on Very Large Data Bases*, pages 407–419, September 1995.
46. W.-G. Teng, M.-S. Chen, and P. S. Yu. A Regression-Based Temporal Pattern Mining Scheme for Data Streams. *Proceedings of the 29th International Conference on Very Large Data Bases*, pages 93–104, September 2003.
47. S. Thomas, S. Bodagala, K. Alsabti, and S. Ranka. An Efficient Algorithm for the Incremental Updation of Association Rules in Large Databases. *Proceedings of the 3rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 263–266, August 1997.
48. H. Toivonen. Sampling Large Databases for Association Rules. *Proceedings of the 22th International Conference on Very Large Data Bases*, pages 134–145, September 1996.
49. A. Veloso, B. Possas, W. M. Jr., and M. B. de Carvalho. Knowledge Management in Association Rule Mining. *Workshop on Integrating Data Mining and Knowledge Management (in conjuction with ICDM2001)*, November 2001.
50. K. Wang, Y. He, and J. Han. Mining Frequent Itemsets Using Support Constraints. *Proceedings of the 26th International Conference on Very Large Data Bases*, pages 43–52, September 2000.
51. K. Wang, S. Q. Zhou, and S. C. Liew. Building Hierarchical Classifiers Using Class Proximity. *Proceedings of the 25th International Conference on Very Large Data Bases*, pages 363–374, 1999.
52. C. Yang, U. Fayyad, and P. Bradley. Efficient Discovery of Error-Tolerant Frequent Itemsets in High Dimensions. *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2001.
53. M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New Algorithms for Fast Discovery of Association Rules. *Proceedings of the 3rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 283–286, August 1997.
54. Z. Zhou and C. I. Ezeife. A Low-Scan Incremental Association Rule Maintenance Method. *Proceedings of the 14th Canadian Conference on Artificial Intelligence*, June 2001.