Chapter 5.
Incremental Association Mining Algorithms

5.1 Introduction

In this chapter two incremental mining algorithms are presented. The first algorithm Incremental mining $K^{th}$ frequent itemset, in short INK, for update database is extension to AMKIS algorithm as presented in previous chapter but does not use lookup table and item encoding and decoding functions. The second algorithm is Incremental Frequent Itemset Mining, in short INFRIM, which generates required frequent itemsets using recursive itemset generation function.

Most of the research is reported on developing highly scalable and efficient algorithms for frequent itemset mining but there is very limited work has been reported in the literature on incremental mining. In a typical data mining process, full data is rarely collected in single attempt. The collection of data is a continuous and ongoing process. In typical business operations, the data collection time will be as small as in weeks, days, hours, minutes and even seconds from operational data sources. The data collection time mostly depends on the nature of the business and business value. In many cases, data collection is carried out in phases or batches. Consequently, the content of the underlying database changes over time. In order to study the behavior of frequent itemset patterns in continuously growing databases, mining algorithms need to run iteratively or repeatedly. In simple case, the mining algorithm has to run on full database that is a monotonous task. Alternatively the mining algorithm that uses previous mining results before to run on incremental database which is known as incremental mining that was first proposed by David W Cheung et al. [140].

*Part of this chapter has been:


c) Presented orally and published in the proceedings of the 8th Int. Conf. on Information Processing, Management and Intelligent Information Technology (ICIPT 2013), April 1- 3, 2013, Seoul, Republic of Korea.
The concept of incremental mining includes not only to update, insertion and deletion operations on database but also on extracted knowledge to get timely update of frequent patterns. In most of the business applications, database grows continuously. New items will include in the catalogs and few items will become obsolete over a period. Hence, there is a need to mine database again to understand association rules that includes earlier discovered rules and also for discovering new rules. According to N.L. Sarda et al. [128] applying mining algorithms on the updated database (the older database plus the incremental database) is costly activity.

In real time business scenario, the users are interested in specific frequent itemset patterns instead of finding all frequent itemset patterns. Most of the existing algorithms in the literature will find all frequent itemsets patterns from transaction database which start from one itemset generation to the specific itemset. So, the generation of specific frequent itemset in continuously growing databases is found to be an open research problem. This motivates us to develop incremental mining algorithms for $K$th frequent itemset generation.

The objective of this chapter is to design, develop and implement two incremental mining algorithms which are suitable to operational level of the business for timely decision making in current time.

- The first algorithm is an Incremental mining for generating $K$th frequent itemset for update case in short INK. This algorithm is in extension to the AMKIS algorithm presented in the previous chapter.
- The second algorithm is Incremental Frequent Itemset Mining in short INFRIM. This algorithm is in extension to INK algorithm.

5.2 Related Work

Recent studies have been found [41], [104], [123], [124], [125], [126], [127], [128], [129], [130], [132], [133], [134], [135], [136], [137], [138], [139], [140], [141], [143], [150], [153], [161], [165] various incremental mining algorithms and as well as maintenance of association rules in the literature. Different types of incremental mining association algorithms reported in the literature and their functionality are of the proposed system is discussed. The available incremental mining algorithms can be broadly classified into two major groups known as Apriori and FP growth.
5.2.1 Apriori based Algorithms

David W Cheung et al. [140] was first studied the problem of maintenance of association rules in 1996 and proposed Fast Update Algorithm (FUP). The framework of FUP is based on Apriori that adopts the pruning techniques used in the DHP [151]. FUP algorithm first calculates large itemsets mainly from newly inserted transactions and compares them with the previous large itemsets from the original database. According to the comparison results, FUP determines whether re-scanning the original database is needed, thus saving some time in maintaining the association rules. Although the FUP algorithm can indeed improve mining performance for incrementally growing databases, original databases still need to be scanned when necessary. The FUP algorithm can handle the maintenance problem in the case of insertion only. However, the FUP algorithm does not handle the case of deleting transactions from the database. Similarly, modification of transactions is not addressed.

David W Cheung et al. [123] were developed FUP2 which address the new rules in an update cases including insertion, deletion and modification of transactions. The FUP2 algorithm makes use of the previous mining results to cut down the cost of finding the new rules in an updated database. FUP2 is a complementary algorithm of FUP. In insertion case FUP2 is equivalent to FUP. The FUP2 technique updates the association rules when old transactions are removed from the database and new transactions are added to it. This algorithm uses information available from a previous mining.

Toshi Chandraker et al. [124] were proposed an incremental mining algorithm based on the concept of pre-larger itemsets based on two support thresholds which avoids the movements of itemsets directly from large to small and vice versa. Further this algorithm reduces the need for rescanning original database.

Zequin Zhou et al. [150] were developed Maintaining Association rule with Apriori Property (MAAP) algorithm for incrementally maintaining association rules in the updated database. The MAAP algorithm applies Apriori property to the set of large itemsets in the old database, generates some parts of the lower level large itemsets in the new database. The MAAP algorithm uses all previous old large itemsets and that are confirmed to be still large in the new database. Thus, MAAP algorithm eliminates the need to compute parts of lower level large itemsets that saves rule maintenance time by reducing scanning of database number of times. It achieves more benefit when high level large itemsets can be used to generate a lot of low level large itemsets in the first step of applying the Apriori property.
Chin Chan Chang et al. [155] were developed New Fast UPdate algorithm (NFUP) for incremental mining which is Apriori like algorithm. In this the incremental database is logically divided into \( n \) partitions according to unit time interval. The latest information is at the last partition of incremental database. Therefore, NFUP scans each partition backward which is suitable for frequently updated database. The NFUP does not require scanning of the original database.

5.2.2 FP Growth based Algorithms

Han et al. [98] were proposed FP growth algorithm to discover frequent patterns using FP tree without candidate generation. FP growth traverses the FP tree in a depth first manner. Since the inception of FP tree, several incremental mining algorithms have been proposed [130], [138], [139], [104] in the literature that are based on FP growth.

Lei Chang et al. [153] were developed an Incremental Mining of Closed Sequential Patterns (IMCS) algorithm which maintains the Closed Sequence Tree (CSTree) when the sequence database is updated incrementally which is capable of mining the closed frequent sequences.

Jia Ling Koh et al. [139] were proposed Adjusting FP-Tree for Incremental Mining (AFPIM) algorithm which updates previously constructed FP-tree. The adjusting FP tree contains frequent items based on user specified minimum support threshold by scanning only the incremental part of the dataset. As items are arranged in descending order of support count based on original dataset, AFPIM re-sorts the items according to new values of support count based on incremental dataset through bubble-sort. There are two major drawbacks of AFPIM algorithm: First, computational expensiveness of sorting process. Second, when new frequent patterns emerge, as a result of scanning of incremental dataset, AFPIM has to construct a new FP-Tree.

William Cheung et al. [104] were developed Compressed and Arranged Transaction Sequence (CATS) Tree based frequent pattern mining algorithm which addresses the limitations of AFPIM algorithm. Unlike AFPIM, the CATS tree considers all the items in the transactions for representation into tree, regardless of whether items are frequent or not. This allows CATS tree to represent even new emerging frequent patterns from incremental dataset. The CATS tree algorithm arranges the nodes based on their local support count, which helps to achieve high compactness of the tree. For incremental mining CATS tree updates the existing tree.
by considering the transactions of the incremental dataset one by one and merging them with existing tree branches. However, CATS tree has two limitations. First, for each new transaction, it is required to find the right path for the new transaction to merge in. Second, it is required to swap and merge the nodes during the updates as the nodes in CATS tree are locally sorted.

Carson Kai Sang Leung et al. [138] were proposed Canonical-order Tree (CanTree) algorithm. Construction of CanTree is very much similar to CATS tree except that, in CanTree items are arranged according to some canonical order. The canonical order can be determined by the user prior to mining process. Canonical ordering can be lexicographic or based on certain property values of items. Since the canonical order is fixed and not based on the support count, CanTree allows easy insertion of nodes. Unlike the CATS Tree, transaction insertions in CanTree require no extensive searching of mergeable paths. CanTree too has some limitations. It generates compact tree if and only if majority of the transactions contain common pattern-base in canonical order. It generates skewed tree with too many branches. Further, though the CanTree takes less time for tree construction, it requires more memory and more time for extracting frequent patterns from the generated CanTree.

Shashikumar G. Totad et al. [130] were proposed Batch Incremental Tree (BIT) algorithm which merges the small consecutive duration of FP-trees to obtain a FP-tree. The resultant FP tree is equivalent to FP-tree obtained when the entire database is processed at once from the beginning of the first duration to the end of the second duration. BIT algorithm takes the advantage of previously obtained periodical FP-tree. BIT algorithm takes FP tree of the two periodic datasets. This reads the itemsets of one of the FP-tree (T1) one by one along with their frequency counts and searches for the mergeable prefix path of the other FP-tree (T2). The itemset of T1 merges with the mergeable prefix by updating frequency count of the items and inserting remaining non-prefix items (if any) by extending the tree branch after the last matching prefix item of the mergeable pattern. BIT algorithm takes less time for tree construction as compared to normal FP tree.

5.3 INK: Incremental Mining for K\textsuperscript{th} Frequent Itemset Algorithm

The proposed incremental mining K\textsuperscript{th} frequent itemset algorithm for update database is envisaged in algorithm 5.1 and 5.2. This algorithm is called in short as INK. The inputs to the algorithm are binary representation of transactional database and the output is K\textsuperscript{th} frequent itemsets and their count. The INK algorithm generates
frequent itemset, where $K > 1$, from transactional database directly without generating lower frequent itemsets. It is considered that previous mining results are available in main memory inorder to generate updated knowledge database. The process of generating updating mining results is not specified in the proposed algorithm which is carried as a background task in predefined time interval.

**Algorithm 5.1. Itemset generation**

<table>
<thead>
<tr>
<th>Inputs:</th>
<th>Transaction database and itemset to be generated (K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td>$K^{th}$ Frequent itemsets generation</td>
</tr>
<tr>
<td>1.</td>
<td>Transaction database $I_{db} = {t_1, t_2, ..., t_N}$</td>
</tr>
<tr>
<td>2.</td>
<td>repeat</td>
</tr>
<tr>
<td>3.</td>
<td>for each transaction in $I_{db}$ do</td>
</tr>
<tr>
<td>4.</td>
<td>$t_r = \text{itemsInTransaction}(t_i)$</td>
</tr>
<tr>
<td>5.</td>
<td>if (length of $t_r \geq K$)</td>
</tr>
<tr>
<td>6.</td>
<td>$L_k = \text{ItemsetGen}(t_r)$</td>
</tr>
<tr>
<td>7.</td>
<td>for each itemset element in $L$ do</td>
</tr>
<tr>
<td>8.</td>
<td>Item count ++</td>
</tr>
<tr>
<td>9.</td>
<td>end for</td>
</tr>
<tr>
<td>10.</td>
<td>end if</td>
</tr>
<tr>
<td>11.</td>
<td>end for</td>
</tr>
<tr>
<td>12.</td>
<td>until $I_{db} = \emptyset$</td>
</tr>
</tbody>
</table>

**Algorithm 5.2. Itemset generation**

<table>
<thead>
<tr>
<th>Inputs:</th>
<th>Itemset to be generated (K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td>$K^{th}$ Frequent itemsets generation</td>
</tr>
<tr>
<td>1.</td>
<td>Declare an vector of $items$</td>
</tr>
<tr>
<td>2.</td>
<td>Declare $loopcounter$</td>
</tr>
<tr>
<td>3.</td>
<td>Initialize the variable $K$ itemset</td>
</tr>
<tr>
<td>4.</td>
<td>for 1 to $loopcounter$</td>
</tr>
<tr>
<td>5.</td>
<td>Generate binary of $loopcounter$</td>
</tr>
<tr>
<td>6.</td>
<td>generate subsets of size $k$</td>
</tr>
<tr>
<td>7.</td>
<td>Add itemset to $items$ vector</td>
</tr>
<tr>
<td>8.</td>
<td>end if</td>
</tr>
<tr>
<td>9.</td>
<td>end for</td>
</tr>
<tr>
<td>10.</td>
<td>return $items$ vector</td>
</tr>
<tr>
<td>11.</td>
<td>end for</td>
</tr>
</tbody>
</table>

### 5.3.1 Binary Representation of Transaction Data

The input to the proposed algorithm is a binary vector format of transactional data as shown in Table 5.1. Each record represents a transaction with transaction identification (TID) and each column represents an item (I) in the transaction. Each
row contains only ‘1’ and ‘0’s. Binary ‘1’ in the transaction represents the presence of an item and ‘0’ represents the absence of an item.

Table 5.1. Binary representation of transaction data

<table>
<thead>
<tr>
<th>TID</th>
<th>I₁</th>
<th>I₂</th>
<th>I₃</th>
<th>I₄</th>
<th>I₅</th>
</tr>
</thead>
<tbody>
<tr>
<td>T100</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T200</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>T300</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>T400</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>T500</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

5.3.2 Working of Algorithm

The inputs to the algorithm 5.1 are incremental database, previous mining results of original database and the value of frequent itemset (K) is to be determined. It is assumed that the previous mining results are available to the proposed algorithm in main memory as frequent itemsets and their count.

The algorithm 5.1 reads each transaction of incremental database and then finds the number of items present in a transaction called item count (tᵢ). If the length of item count is more than or equal to the itemset (K) to be find then generates all possible sub itemsets otherwise transaction is pruned. Itemset generation is obtained by the use of an algorithm 5.2. This algorithm uses simple binary principles to generate all possible sub-itemsets directly without generating lower itemsets. Finally, the count of each itemset will be incremented. This process will repeat till the end of incremental database. Finlay, extracted mining results of incremental database is updated with previous mining results which are available main memory.

5.3.3 Output of Algorithm

The output of algorithm consists of all frequent Kᵗʰ itemsets and their count of incremental database. In addition, updated frequent itemsets results are also available.

5.4 Datasets

Testing of the proposed algorithm on different synthetic datasets is conducted. The specifications of these data sets are provided in table 5.2.

Table 5.2. Specifications of datasets

<table>
<thead>
<tr>
<th>Name of dataset</th>
<th>No. of transactions</th>
<th>Size, MB</th>
<th>No. of items</th>
<th>Average items</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS#01</td>
<td>1M</td>
<td>10.773</td>
<td>5</td>
<td>2.562</td>
</tr>
<tr>
<td>DS#02</td>
<td>1M</td>
<td>14.649</td>
<td>7</td>
<td>3.593</td>
</tr>
<tr>
<td>DS#03</td>
<td>1M</td>
<td>20.508</td>
<td>10</td>
<td>5.155</td>
</tr>
</tbody>
</table>
5.5 Experiments

Several experiments are conducted to measure the performance of the proposed algorithms and compared with Apriori algorithm. The proposed algorithm is implemented in Java programming language. The experimental setup is same as presented in Section 1.7.3 of Chapter 1 of this thesis. Tested experimental results of these algorithms by measuring runtime against varying size of databases of original and incremental databases in fixed propositions (40:60) and keeping the total database size fixed. The size of incremental database is varied in steps of 10% of total database.

5.6 Results and Discussion

The performance of the proposed algorithm is compared with Apriori [99] and few results of experiments are presented in graphs as shown in figure 5.1(a) to 5.1(d).

![Figure 5.1(a)](image1.png)

![Figure 5.1(b)](image2.png)

![Figure 5.1(c)](image3.png)

![Figure 5.1(d)](image4.png)

Figure 5.1. Performance comparison of INK algorithm with Apriori
The proposed INK algorithm is simple in structure that generates $K^{th}$ frequent itemsets directly from the given incremental database for update case. The algorithm uses simple binary concepts for generation of item sub-sets from each transaction. These figures show that the proposed algorithm takes less runtime as compared with Apriori for different itemset values ranges from 2 to 5. The support count of Apriori was considered as 0.01 (1%) for these experiments. The performance of the proposed algorithm is compared with Apriori and found higher performance, linearly scalable with database size and scans database only once.

The speedup ratio ($s$) was calculated [128] and is given below. The value of $s$ increases as increasing itemset (K) that ranges between 2.35 and 11.07 during our experimental results.

$$s = \frac{(\text{time for Apriori})_{Wdb} + (\text{time for Apriori})_{Idb}}{(\text{time for proposed algorithm})_{Wdb+Idb}}$$ (5.1)

5.7 INFRIM: Incremental Frequent Itemset Mining Algorithm

5.7.1 Introduction

Most of the proposed algorithms in association mining focus on batch mining that do not permit full scan of database for every update and does not utilize previous mining results. In most of real world applications the organizational database grows continuously and is updated into data warehouse. Thus for each update demands to rerun of data mining algorithms inorder to maintain discovered information. Mining is a costly activity and needs to reduce full scan of database for each update. The concept of incremental mining was proposed [149] to maintain the discovered rules during previous mining process. Incremental mining means to find delta from source, extract knowledge from delta and update previous mining results with extracted knowledge. The objective of incremental mining [146] is to avoid re-learning of rules from the old data and utilize knowledge that has already been discovered.

A novel algorithm for Incremental Frequent Itemsets Mining (INFRIM) which extracts frequent itemsets for updating data is presented. It is assumed previous mining results are available in main memory as lookup to the proposed algorithm. The INFRIM algorithm will provide not only extracted mining results of incremental database but also updated database.
5.7.2 Symbols and Notation

The various symbols used in this paper and their meaning are provided in table 5.3.

Table 5.3. Symbols and meaning

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>( Wdb )</td>
<td>Original database</td>
</tr>
<tr>
<td>( Idb )</td>
<td>Incremental database</td>
</tr>
<tr>
<td>( Udb )</td>
<td>Updated database</td>
</tr>
<tr>
<td>( N )</td>
<td>Number of items</td>
</tr>
<tr>
<td>( K )</td>
<td>Number of itemset</td>
</tr>
<tr>
<td>( S_k )</td>
<td>Frequent ( K )-itemset</td>
</tr>
<tr>
<td>((Kdb)_w)</td>
<td>Original database mining results</td>
</tr>
<tr>
<td>((Kdb)_{int})</td>
<td>Intermediate mining results database</td>
</tr>
<tr>
<td>((Kdb)_{inc})</td>
<td>Incremental database mining results</td>
</tr>
<tr>
<td>((Kdb)_U)</td>
<td>Updated database mining results</td>
</tr>
</tbody>
</table>

**Original database (Wdb):** This is most commonly referred as data warehouse. It is a central repository of data which is created by integrating data from multiple disparate sources. This stores historical data that is used for reporting and data analysis. This is commonly used for creating trend reports for senior management reporting for strategic and tactical level decision making. This acts single source of truth of an organization.

**Incremental database (Idb):** This is the data obtained from the operational systems that is transaction databases in a given window of time interval and the time interval will vary form case to case basis and typically ranges days, hours or even minutes. The extraction of delta from the operational data sources is out of scope of this paper. It is assumed that incremental database is available as one of the inputs to the algorithm.

**Updated database (Udb):** This database is a combination of original database (Wdb) and incremental database (Idb).

**Knowledge database (Kdb):** In order to extract the knowledge from database, mining algorithm to be run. The extracted knowledge contains the frequent patterns and frequent item counts which is known as knowledge database. The incremental mining system contains original \((Kdb)_w\), intermediate \((Kdb)_{int}\), incremental \((Kdb)_{inc}\) and updated \((Kdb)_U\) knowledge databases. Original knowledge database is the database which contains previous mining results of original database. Intermediate knowledge database is formed by the combination of frequent 1-item counts of incremental database and previous mining results of original database. From these intermediate mining results, all frequent itemsets will be generated based on support.
threshold. The proposed incremental mining algorithm extracts knowledge from incremental database based on the itemsets to be generated and all other are pruned from extraction. In the proposed algorithm, the previous mining details of original database are updated with incremental mining results that were obtained from incremental database and forms as updated mining results.

Incremental mining: The total functionality of INFRIM is summarized as extraction of each transaction of incremental database, generated frequent itemsets dynamically and updated previous extracted knowledge database and as well as database of data warehouse. The output will be frequent itemsets.

5.7.3 Algorithm

The algorithm of proposed incremental frequent itemset mining is envisaged below from algorithm 5.3 to 5.7. The inputs to the algorithm are original database ($W_{db}$), previous mining results of original database ($K_{db}_{W}$), 1-item count of incremental database. The outputs will be frequent itemsets of incremental database ($K_{db}_{I}$) and frequent itemsets of updated database ($K_{db}_{U}$). The overall functionality of the proposed algorithm is envisaged as “to extract the knowledge from delta and update previous mining results with extracted knowledge.”

Algorithm 5.2. INFRIM: Incremental frequent itemset mining

| Input: Incremental database ($Idb$), Previous mining results ($K_{db}_{W}$) |
| Output: Frequent itemsets of $Idb$ and $Udb$ |
| 1. begin |
| 2. Extract_Mining_Results() |
| 3. Frequent_Patterns_Under_Study() |
| 4. Incremental_Mining() |
| 5. Update_Database() |
| 6. end |
Algorithm 5.3. Extract mining results

**Input:** 1-Itemset count of \((Idb)\), Previous mining results \((Kdb)\)

**Output:** Intermediate mining results

1. begin
2. get Transaction Count of \(Idb\)
3. get Transaction Count of \(Wdb\)
4. update Transaction Count of \(Udb\)
5. for each 1- Item \(\in Idb\) do
6. get 1- item Count of \(Idb\)
7. get 1-item Count of \(Wdb\)
8. update 1-Item Count of \(Udb\)
9. end for
10. End

Algorithm 5.4. Frequent patterns under study

**Input:** 1-Itemset count of \((Idb)\), Previous mining results \((Kdb)\)

**Output:** Intermediate mining results

1. begin
2. get Transaction Count of \(Idb\)
3. get Transaction Count of \(Wdb\)
4. update Transaction Count of \(Udb\)
5. for each 1- Item \(\in Idb\) do
6. get 1- item Count of \(Idb\)
7. get 1-item Count of \(Wdb\)
8. update 1-Item Count of \(Udb\)
9. end for
10. end

Algorithm 5.5. Incremental mining

**Input:** Incremental database \((Idb)\), Frequent patterns under study

**Output:** Frequent itemsets of \(Idb\)

1. repeat
2. for each \(t \in Idb\) do
3. for each frequent pattern do
4. itemSetGen()
5. updateItemCount()
6. end for
7. end for
8. until \(Idb = \emptyset\)
Algorithm 5.6. Itemset generation using recursion

**Input:** Itemset to be generated (K), Transaction of Idb.

**Output:** Specific Itemset

1. ksubsets = {}
2. k, n,
3. set ← {1, 2, 3, …, n}
4. S_k getKSubsets ()
5. {
6. S = {}
7. generateKSubset (0,0,S)
8. retn ksubsets
9. }
10. generateKSubset (start, index, ksubset)
11. {
12. if (index = k)
13. ksubsets.add (ksubset)
14. else
15. generateKSubset( start +1, index+1, ksubset)
16. }

Algorithm 5.7. Update database

**Input:** Mining results of Idb (Kdb)_I, Previous mining results (Kdb)_W

**Output:** Frequent itemsets of Idb and Udb

1. repeat
2. for each itemset count of Idb do
3.     get item count of (Kdb)_I
4.     get item count of (Kdb)_W
5.     update item count of (Kdb)_U
6. end for
7. until (Kdb)_I = ø
5.7.4 Functionality of INFRIM Algorithm

The functionality of the proposed algorithm is shown in figure 5.2 and summarized as five step process. The description of each step is outlined in the subsequent paragraph.

**Figure 5.2. Functionality of INFRIM algorithm**

1) *Data pre-processing*: The extracted data from source systems is available as incremental database. This incremental database is pre-processed and then transformed into a binary vector form. Each row corresponds to a transaction and each column corresponds to an item. An item can be treated as a binary variable whose value is one if the item is present in a transaction and zero otherwise. During this data pre-processing phase count of 1-itemsets are computed.

2) *Mining results extraction*: In this step, 1-itemset count of incremental database is combined with previous mining frequent itemsets count of original database and forms as intermediate mining results. In addition to this, it will also update parameter like transaction counts and support threshold during this phase. Finally, the output of this phase will be an intermediate frequent patterns knowledge database (Kdb)\text{Int}.

3) *Frequent patterns under study*: In this step, as per the user defined support count threshold, system will check and generate frequent items to be required for extraction from incremental database.

4) *Scanning of incremental database*: During this step, it scans incremental database transaction by transaction, generates frequent itemsets with their count. It prunes all unwanted frequent patterns that are below support threshold (minimum support) during knowledge extraction. Finally, it generates all frequent itemsets of incremental database.
5) *Mining results update:* Frequent itemsets results obtained from incremental database are updated with previous mining results of original database and forms as updated frequent mining results of updated database. The outputs contain mining results of both incremental database as well as update database.

### 5.7.5 Datasets

Experiments were performed on the following two synthetic databases known as Bakery and Retail stores. The specifications of bakery database consist of four datasets whose specifications are given in table 5.4.

#### Table 5.4. Specifications of Bakery datasets

<table>
<thead>
<tr>
<th>Name of dataset</th>
<th>No. of transactions</th>
<th>Size</th>
<th>No. of items</th>
<th>Average items</th>
<th>Maximum no. of items</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS#01</td>
<td>1000</td>
<td>99 KB</td>
<td>50</td>
<td>3.538</td>
<td>8</td>
</tr>
<tr>
<td>DS#02</td>
<td>5000</td>
<td>494 KB</td>
<td>50</td>
<td>3.547</td>
<td>8</td>
</tr>
<tr>
<td>DS#03</td>
<td>20000</td>
<td>1.97 MB</td>
<td>50</td>
<td>3.556</td>
<td>8</td>
</tr>
<tr>
<td>DS#04</td>
<td>75000</td>
<td>7.40 MB</td>
<td>50</td>
<td>3.549</td>
<td>8</td>
</tr>
</tbody>
</table>

The specifications of retail store database are provided in table 5.5 that contains seven different datasets.

#### Table 5.5. Specifications of Retail store datasets

<table>
<thead>
<tr>
<th>Name of dataset</th>
<th>No. of transactions</th>
<th>Size</th>
<th>No. of items</th>
<th>Average items</th>
<th>Maximum no. of items</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS#01</td>
<td>1000</td>
<td>41 KB</td>
<td>20</td>
<td>1.83</td>
<td>6</td>
</tr>
<tr>
<td>DS#02</td>
<td>5000</td>
<td>201 KB</td>
<td>20</td>
<td>2.43</td>
<td>9</td>
</tr>
<tr>
<td>DS#03</td>
<td>30K</td>
<td>1.20 MB</td>
<td>20</td>
<td>2.77</td>
<td>9</td>
</tr>
<tr>
<td>DS#04</td>
<td>100K</td>
<td>4.01 MB</td>
<td>20</td>
<td>1.82</td>
<td>6</td>
</tr>
<tr>
<td>DS#05</td>
<td>500K</td>
<td>20.02 MB</td>
<td>20</td>
<td>2.03</td>
<td>11</td>
</tr>
<tr>
<td>DS#06</td>
<td>1M</td>
<td>40.04 MB</td>
<td>20</td>
<td>4.06</td>
<td>15</td>
</tr>
<tr>
<td>DS#07</td>
<td>4.5M</td>
<td>180.28 MB</td>
<td>20</td>
<td>4.01</td>
<td>15</td>
</tr>
</tbody>
</table>

### 5.7.6 Experimental Results

The proposed algorithm, INFRIM, is simple in structure and generates frequent itemsets from the intermediate mining results which is obtained from previous mining results and extracted results of incremental database. The proposed algorithm is tested on synthetic databases of various sizes whose specifications are provided in table 5.4 and 5.5. The experimental setup used to measure the performance of INFRIM and Apriori as similar presented in Section 1.7.3 of Chapter 1.
The performance of the proposed algorithm is compared with Apriori and few experimental results are provided in figure 5.3(a) to 5.3(b).

Figure 5.3(a) Bakery Database

Figure 5.3(b) Retail Store Database

Figure 5.3. Performance comparison of INFRIM algorithm and Apriori

Figure 5.4 (c) shows the speedup ratio of the proposed algorithm with Apriori for dataset (DS#07) of Retail Store database which is specified in table 5.5.

Figure 5.4. Performance of INFRIM algorithm with Apriori for large datasets

The performance of INFRIM algorithm was compared with Apriori [99]. The Apriori was run the database (Wdb) transactions and also on (Idb). The speedup ratio was computed as described in [126]. In our experiments, the speedup ratio ranges from 1.2 to 3.1.

The proposed INFRIM algorithm is simple in structure which dynamically generates frequent itemsets from incremental database. In order to find the performance of the proposed algorithm experiments were conducted on two different
databases whose transactions ranges from several thousands to millions. Figure 5.3(a) shows performance comparison of the proposed algorithm with Apriori of Bakery for 2-itemset of bakery database. It is clearly seen that the amount of time required for the proposed algorithm is much lower. Similarly, figure 5.3(b) shows the performance comparison for 2-itemset of retail database. From these two graphs, it is noticed that as transaction volume increased the run time decreases as compared to Apriori. Figure 5.4 shows the speedup ratio of the proposed algorithm with Apriori for 2-itemset to 5-itemset and found highly scalable and efficient for even large datasets. During our experiments it is found that the speedup ratio ranges from 1.2 to 3.1. The proposed algorithm not only provides timely knowledge to various operational users of business organization for their decision making but also supports incremental mining in maintenance of data warehouse and business intelligence projects.

5.7.7 Salient Features of INFRIM Algorithm

The proposed INFRIM algorithm has the following salient features:

- It uses previous mining results of original database thereby it avoids rescanning of original database.

- It works dynamically. Each transaction of incremental database is processed and then the corresponding itemsets will be updated in knowledge database which includes itemset counts, itemsets, transactions count and updated database.

- It avoids re-computing large itemsets that have already been discovered.

- In the proposed algorithm there is no need of multiple scan or even re-scan of the database for computing large itemsets because it uses previous mining results and scans database only once.

- The proposed algorithm focuses on newly transactions obtained from incremental database. Thus, it greatly reduces the number of candidate itemsets generation which intern increases computation time. As a result it improves the efficiency of the proposed algorithm.

- It uses a simple check of 1-itemset support count of extracted knowledge database before to update the frequent item counts and then generates dynamically the required frequent itemsets from each transaction of incremental database. As a result of this itemsets generation is optimized this in turn saves computation time. Keeping all the above features of the
proposed incremental frequent Itemset mining algorithm for updating records is highly scalable for any large incremental databases.

- The INFRIM algorithm is highly suitable for operational business intelligence systems which provide miniscule details of frequent itemsets information for low level decision making and as well as strategic and tactical level decision making to the organizations.

5.8 Conclusion

In this chapter two incremental association mining algorithms are presented. The first algorithm is Incremental mining for \( K^{th} \) frequent itemset for update database in short INK. The second algorithm is Incremental Frequent Itemset Mining in short INFRIM.

The INK algorithm directly generates the required \( K^{th} \) frequent itemsets from the given transactional database without generating any lower values of frequent itemsets. This uses simple binary concepts for generating \( K^{th} \) frequent itemsets for update case. The previous mining results of original database are updated with the extracted knowledge in order to provide updated knowledge database. The performance of this algorithm is higher as compared with Apriori and found satisfactory results over wide range of transactions that is linearly scalable with database size. The speedup ratio ranges between 2.35 and 11.07 during our experimental results. The INK algorithm runs significantly faster than mining the updated database from scratch.

The INFRIM algorithm updates the extracted knowledge database with dynamically generated frequent itemsets from the incremental transaction database. It uses the knowledge available from the previous mining to reduce the amount of work to discover the frequent patterns in the updated database. The performance of INFRIM is compared with Apriori. The speedup ratio ranges from 1.2 to 3.1 for generation of 2-itemset to 5-itemset. The INFRIM algorithm works well over wide ranges of system parameter values in terms of size of incremental database, number of transactions in incremental transaction database, size of original database, size of knowledge database. Finally, the salient features of the INFRIM algorithm are presented.