Minimally Infrequent Itemset Mining using Pattern-Growth Paradigm and Residual Trees

Ashish Gupta
ashgupta@cse.iitk.ac.in

Akshay Mittal
amittal@cse.iitk.ac.in

Arnab Bhattacharya
arnabb@iitk.ac.in

Department of Computer Science and Engineering,
Indian Institute of Technology, Kanpur, India.

Abstract

Itemset mining has been an active area of research due to its successful application in various data mining scenarios including finding association rules. Though most of the past work has been on finding frequent itemsets, infrequent itemset mining has demonstrated its utility in web mining, bioinformatics and other fields. In this paper, we propose a new algorithm based on the pattern-growth paradigm to find minimally infrequent itemsets. A minimally infrequent itemset has no subset which is also infrequent. We also introduce the novel concept of residual trees. We further utilize the residual trees to mine multiple level minimum support itemsets where different thresholds are used for finding frequent itemsets for different lengths of the itemset. Finally, we analyze the behavior of our algorithm with respect to different parameters and show through experiments that it outperforms the competing ones.

1 Introduction

Mining frequent itemsets has found extensive utilization in various data mining applications including consumer market-basket analysis [1], inference of patterns from web page access logs [11], and iceberg-cube computation [2]. Extensive research has, therefore, been conducted in finding efficient algorithms for frequent itemset mining, especially in finding association rules [3].

However, significantly less attention has been paid to mining of infrequent itemsets, even though it has got important usage in (i) mining of negative association rules from infrequent itemsets [12], (ii) statistical disclosure risk assessment where rare patterns in anonymous census data can lead to statistical disclosure [7], (iii) fraud detection where rare patterns in financial or tax data may suggest unusual activity associated with fraudulent behavior [7], and (iv) bioinformatics where rare patterns in microarray data may suggest genetic disorders [7].

The large body of frequent itemset mining algorithms can be broadly classified into two categories: (i) candidate generation-and-test paradigm and (ii) pattern-growth paradigm. In earlier studies, it has been shown experimentally that pattern-growth based algorithms are computationally faster on dense datasets.

Hence, in this paper, we leverage the pattern-growth paradigm to propose an algorithm IFP_min for mining minimally infrequent itemsets. For some datasets, the set of infrequent itemsets can be exponentially large. Reporting an infrequent itemset which has an infrequent proper subset is redundant, since the former can be deduced from the latter. Hence, it is essential to report only the minimally infrequent itemsets.

Haglin et al. proposed an algorithm, MINIT, to mine minimally infrequent itemsets [7]. It generated all potential candidate minimal infrequent itemsets using a ranking order of the items based on their supports and then validated them against the entire database.

Instead, our proposed IFP_min algorithm proceeds by processing minimally infrequent itemsets by partitioning the dataset into two parts, one containing a particular item and the other that does not.

If the support threshold is too high, then less number of frequent itemsets will be generated resulting in loss of valuable association rules. On the other hand, when the support threshold is too low, a large number of frequent itemsets and consequently large number of association rules are generated, thereby making it difficult for the user to choose the important ones. Part of the problem lies in the fact that a single threshold is used for generating frequent itemsets irrespective of the length of the itemset. To alleviate this problem, Multiple Level Minimum Support (MLMS) model was proposed [5], where separate thresholds are assigned to itemsets of different sizes in order to constrain the number of frequent itemsets mined. This model finds extensive applications in market basket analysis [5] for optimizing the number of association rules generated. We extend our IFP_min algorithm to the MLMS framework as well.

In summary, we make the following contributions:

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

Table 1: Example database for infrequent itemset mining.

- We propose a new algorithm IFP_min for mining minimally infrequent itemsets. To the best of our knowledge, this is the first such algorithm based on pattern-growth paradigm (Section 5).

- We introduce the concept of *residual trees* using a variant of the FP-tree structure termed as inverse FP-tree (Section 4).

- We propose an optimization on the Apriori algorithm to mine minimally infrequent itemsets (Section 5.4).

- We present a detailed study to quantify the impact of variation in the density of datasets on the computation time of Apriori, MINIT and our algorithm.

- We extend the proposed algorithm to mine frequent itemsets in the MLMS framework (Section 6).

1.1 Problem Specification

Consider $I = \{x_1, x_2, \ldots, x_n\}$ to be a set of items. An *itemset* $X \subseteq I$ is a subset of items. If its length or *cardinality* is k, it is referred to as a *k-itemset*. A *transaction* T is a tuple (tid, X) where tid is the transaction identifier and X is an itemset. It is said to *contain* an itemset Y if and only if $Y \subseteq X$. A *transaction database* TD is simply a set of transactions.

Each itemset has an associated statistical measure called *support*. For an itemset X, $supp(X, TD) = X.count$ where $X.count$ is the number of transactions in TD that contains X. For a user defined threshold σ, an itemset is *frequent* if and only if its support is greater than or equal to σ. It is *infrequent* otherwise.

As mentioned earlier, the number of infrequent itemsets for a particular database may be quite large. It may be impractical to generate and report all of them. A key observation here is the fact that if an itemset is infrequent, so will be all its supersets. Thus, it makes sense to generate only *minimal* infrequent itemsets, i.e., those which are infrequent but whose all subsets are frequent.

Definition 1 (Minimally Infrequent Itemset). *An itemset X is said to be* minimally infrequent *for a support threshold σ if it is infrequent and all its proper subsets are frequent, i.e., $supp(X) < \sigma$ and $\forall Y \subset X, supp(Y) \geq \sigma$.*

Tid	Transactions	Items in *i-flist* order
T_1	A, C, T, W	A, T, W, C
T_2	C, D, W	D, W, C
T_3	A, C, T, W	A, T, W, C
T_4	A, D, C, W	A, D, W, C
T_5	A, T, C, W, D	A, D, T, W,
T_6	C, D, T, B	B, D, T, C

Table 2: Example database for MLMS model.

σ_k	Frequent *k-itemsets*
$\sigma_1 = 4$	{C}, {W}, {T}, {D}, {A}
$\sigma_2 = 4$	{C, D}, {C, W}, {C, A}, {W, A}, {C, T}
$\sigma_3 = 3$	{C, W, T}, {C, W, D}, {C, W, A}, {C, T, A}, {W, T, A}
$\sigma_4 = 2$	{C, W, T, A}, {C, D, W, A}
$\sigma_5 = 1$	{C, W, T, D, A}

Table 3: Frequent *k-itemsets* for database in Table 2.

Given a particular support threshold, our goal is to efficiently generate all the minimally infrequent itemsets (MIIs) using the pattern-growth paradigm.

Consider an example transaction database shown in Table 1. If $\sigma = 2$, $\{B, D\}$ is one of the minimally infrequent itemsets for the transaction database. All its subsets, i.e., $\{B\}$ and $\{D\}$, are frequent but it itself is infrequent as its support is 1. The whole set of MIIs for the transaction database is $\{\{E, B\}, \{E, C\}, \{E, D\}, \{B, D\}, \{A, B, C\}, \{A, C, D\}, \{A, E\}, \{F\}\}$. Note that $\{B, F\}$ is not a MII since one of its subsets $\{F\}$ is infrequent as well.

In the MLMS framework, the problem is to find all frequent (equivalently, infrequent) itemsets with different support thresholds assigned to itemsets of different lengths. We define σ_k as the minimum support threshold for a *k-itemset* ($k = 1, 2, \ldots, n$) to be frequent. A *k-itemset* X is frequent if and only if $supp(X, TD) \geq \sigma_k$.

For efficient processing of MLMS itemsets, it is useful to sort the list of items in each transaction in increasing order of their support counts. This order is called the *i-flist* order. Also, let σ_{low} be lowest minimum support threshold.

Most applications use the constraint $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$. This is intuitive as the support of larger itemsets can only decrease or at most remain constant. In this case, $\sigma_{low} = \sigma_n$. Our algorithm IFP_MLMS, however, does not depend on this assumption, and works for any general $\sigma_1, \ldots, \sigma_n$.

Consider an example transaction database shown in Table 2. The frequent itemsets corresponding to the different thresholds are shown in Table 3.

2 Related Work

The problem of mining frequent itemsets was first introduced by Agrawal et al. [1], who proposed the *Apriori-*algorithm. Apriori is a bottom-up, breadth-first search algorithm that exploits the downward closure property "all subsets of frequent itemsets are frequent". Only candidate

frequent itemsets whose subsets are all frequent are generated in each database scan. Apriori needs l database scans if the size of the largest frequent itemset is l. In this paper, we propose a variation of the Apriori algorithm for mining minimally infrequent itemsets (MIIs).

In [8], Han et al. introduced a novel algorithm known as the *FP-growth* method for mining frequent itemsets. The FP-growth method is a depth-first search algorithm. A data structure called the *FP-tree* is used for storing the frequency information of itemsets in the original transaction database in a compressed form. Only two database scans are needed for the algorithm and no candidate generation is required. This makes the FP-growth method much faster than Apriori. In [6], Grahne et al. introduced a novel *FP-array* technique that greatly reduces the need to traverse the FP-trees. In this paper, we use a variation of the FP-tree for mining the MIIs.

To the best of our knowledge there has been only one other work that discusses the mining of MIIs. In [7], Haglin et al. proposed the algorithm *MINIT* which is based upon the *SUDA2* algorithm developed for finding unique itemsets (itemsets with no unique proper subsets) [9, 10]. The authors also showed that the minimal infrequent itemset problem is NP-complete [7].

In [5], Dong et al. proposed the MLMS model for constraining the number of frequent and infrequent itemsets generated. A candidate generation-and-test based algorithm *Apriori_MLMS* was proposed in [5]. The downward closure property is absent in the MLMS model, and thus, the Apriori_MLMS algorithm checks the supports of all possible *k-itemsets* occurring at least once in the transaction database, for finding the frequent itemsets. Generally, the support thresholds are chosen randomly for different length itemsets with the constraint $\sigma_i \geq \sigma_j, \forall i < j$. In [4], Dong et al. extended their proposed algorithm from [5] to include an interestingness parameter while mining frequent and infrequent itemsets.

3 Need for Residual Trees

By definition, an itemset is a minimally infrequent itemset (MII) if and only if it is infrequent and all its subsets are frequent. Thus, a trivial algorithm to mine all MIIs would be to compute all the subsets for every infrequent itemset and check if they are frequent. This involves finding all the frequent and infrequent itemsets in the database and proceeding with checking the subsets of infrequent itemsets for occurrence in the large set of frequent itemsets. This is a simple but computationally expensive algorithm.

The use of *residual trees* reduces the computation time. A residual tree for a particular item is a tree representation of the residual database corresponding to the item, i.e., the entire database with the item removed. We show later that a MII found in the residual tree is a MII of the entire transaction database.

The projected database, on the other hand, corresponds to the set of transactions that contains a particular item. A potential minimal infrequent itemset mined from the pro-

jected tree must not have any infrequent subset. The itemset itself is a subset since it is actually the union with the item of the projected tree that is under consideration. As we show later, the support of only this itemset needs to be computed from the corresponding residual tree.

In this paper, our proposed algorithm *IFP_min* uses a structure similar to the FP-tree [8] called the *IFP-tree*. This is due to the fact that the IFP-tree provides a more visually simplified version of the residual and projected trees that leads to enhanced understanding of the algorithm. A similar algorithm FP_min can be designed that uses the FP-tree. The time complexity remains the same. In the next section, we describe in detail the IFP-tree and the corresponding structures, projected tree and residual tree.

4 Inverse FP-tree (IFP-tree)

The *inverse FP-tree (IFP-tree)* is a variant of the FP-tree [8]. It is a compressed representation of the whole transaction database. Every path from the root to a node represents a transaction. The root has an empty label. Except the root node, each node of the tree contains four fields: (i) item id, (ii) count, (iii) list of child links, and (iv) a node link. The item id field contains the identifier of the item. The count field at each node stores the support count of the path from the root to that node. The list of child links point to the children of the node. The node link field points to another node with the same item id that is present in some other branch of the tree.

An *item header table* is used to facilitate the traversal of the tree. The header table consists of items and a link field associated with each item that points to the first occurrence of the item in the IFP-tree. All link entries of the header table are initially set to *null*. Whenever an item is added into the tree, the corresponding link entry of the header table is updated. Items in each transaction are sorted according to their order in *i-flist*.

For inserting a transaction, a path that shares the same prefix is searched. If there exists such a path, then the count of the common prefix is incremented by one in the tree and the remaining items of the transaction (which do not share the path) are attached from the last node with their count value set to 1. If items of a transaction do not share any path in the tree, then they are attached from the root. The IFP-tree for the sample transaction database in Table 1 (considering only the transactions T_2 to T_9) is shown in Figure 1.

The *IFP_min* algorithm recursively mines the minimally infrequent itemsets (MIIs) by dividing the IFP-tree into two sub-trees: projected tree and residual tree. The next two sections describe them.

4.1 Projected Tree

Suppose TD be the transaction database represented by the IFP-tree T and TD_x denotes the database of transactions that contain the item x. The *projected database* corresponding to the item x is the database of these transactions
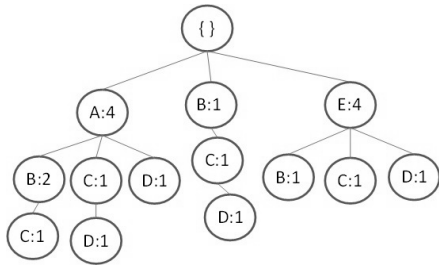
Figure 1: IFP-tree corresponding to the transaction database in Table 1 (T_2-T_9).
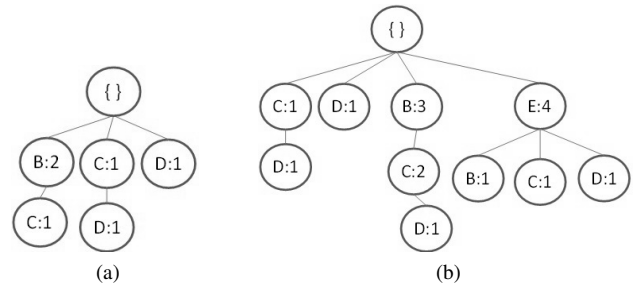

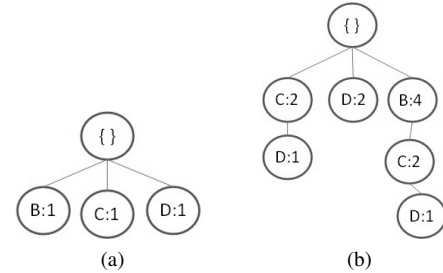Figure 2: (a) Projected tree T_{P_A} and (b) Residual tree T_{R_A} of item A for the IFP-tree shown in Figure 1.

Figure 3: (a) Projected tree and (b) Residual tree of item E in the residual tree T_{R_A} shown in Figure 2b.

TD_x, but after removing the item x from the transactions. The IFP-tree corresponding to this database is called the *projected tree* T_{P_x} of item x in T. Figure 2a shows the projected tree of item A, which is the least frequent item in Figure 1.

The IFP_min algorithm considers the projected tree of only the *least frequent item (lf-item)*. Henceforth, for simplicity, we associate every IFP-tree with only a single projected tree which is that of the *lf-item*. Moreover, since the items are sorted in the *i-flist* order, there would be a single node of the *lf-item* x in the IFP-tree. Thus, the projected tree of x can be obtained directly from the IFP-tree by considering the subtree rooted at node x.

4.2 Residual Tree

The *residual database* corresponding to an item x is the database of transactions obtained by removing item x from TD. The IFP-tree corresponding to this database is called *residual tree* T_{R_x} of item x in T. Figure 2b shows the residual tree of the least frequent item A. It is obtained by deleting the node corresponding to item A and then merging the subtree below that node into the main tree at appropriate positions.

Similar to the projected tree, the IFP_min algorithm considers the residual tree of only the lf-item. Since there is only a single node of the lf-item x in the IFP-tree, the residual tree of x can be obtained directly from the IFP-tree by deleting the node x from the tree and then merging the subtree below the node x with the rest of the tree.

Furthermore, the projected and residual tree of the next item (i.e., E) in the i-flist is associated with the residual tree of the current item (i.e., A). Figure 3 shows the projected and residual trees of the item E for the tree T_{R_A}.

5 Mining Minimally Infrequent Itemsets

In this section, we describe the IFP_min algorithm that uses a recursive approach to mine minimally infrequent itemsets (MIIs). The steps of the algorithm are shown in Algorithm 1.

The algorithm uses a dot-operation ($\bullet$) that is used to unify sets. The unification of the first set with the second produces another set whose i^{th} element is the union of the entire first set with the ith element of the second set. Mathematically, $\{x\} \bullet \{S_1, \ldots, S_n\} = \{\{x\} \cup S_1, \ldots, \{x\} \cup S_n\}$

and $\{x\} \bullet \varnothing = \varnothing$.

The infrequent 1-itemsets are trivial MIIs and so they are reported and pruned from the database in Step 1. After this step, all the items present in the modified database are individually frequent. IFP_min then selects the least frequent item (lf-item, Step 3) and divides the database into two non-disjoint sets: projected database and residual database of the lf-item (Step 11 and Step 12 respectively).

The IFP_min algorithm is then applied to the residual database in Step 13 and the corresponding MIIs are reported in Step 14. In the base case (Step 6 to Step 12) when the residual database consists of a single item, the MII reported is either the item itself or the empty set accordingly as the item is infrequent or frequent respectively.

After processing the residual database, the algorithm mines the projected database in Step 15. The itemsets in the projected database share the lf-item as a prefix. The MIIs obtained from the projected database by recursively applying the algorithm are compared with those obtained from residual database. If an itemset is found to occur in the second set, it is not reported; otherwise, the lf-item is included in the itemset and is reported as an MII of the original database (Step 16).

IFP_min also reports the 2-itemsets consisting of the lf-item and frequent items not present in the projected database of the lf-item (Step 17). These 2-itemsets have support zero in the actual database and hence also qualify as MIIs.

Algorithm 1 IFP_min

Input: T is an IFP-tree
Output: Minimally infrequent itemsets (MIIs) in $db(T)$

1: $infreq(T) \leftarrow$ infrequent 1-itemsets
2: $T \leftarrow$ IFP-tree$(db(T) - infreq(T))$
3: $x \leftarrow$ least frequent item in T (the *lf-item*)
4: **if** T has a single node **then**
5:     **if** $\{x\}$ is infrequent in T **then**
6:         **return** $\{x\}$
7:     **else**
8:         **return** $\varnothing$
9:     **end if**
10: **end if**
11: $T_{P_x} \leftarrow$ projected tree of x
12: $T_{R_x} \leftarrow$ residual tree of x
13: $S_R \leftarrow$ IFP_min (residual tree T_{R_x})
14: $S_{R_{infreq}} \leftarrow S_R$
15: $S_P \leftarrow$ IFP_min (projected tree T_{P_x})
16: $S_{P_{infreq}} \leftarrow \{x\} \bullet (S_P - S_R)$
17: $S_2(x) \leftarrow \{x\} \bullet (items(T_{R_x}) - items(T_{P_x}))$
18: **return** $\{S_{R_{infreq}} \cup S_{P_{infreq}} \cup S_2(x) \cup infreq(T)\}$

5.1 Example

Consider the transaction database TD shown in Table 1. Figure 4 shows the recursive partitioning of the tree T corresponding to TD. The box associated with each tree represents the MIIs in that tree for $\sigma = 2$.

The algorithm starts by separating the infrequent items from the database. This results in removal of the item F (which is inherently a MII). The lf-item in the modified tree T is A. The algorithm then constructs the projected tree T_{P_A} and the residual tree T_{R_A} corresponding to item A and recursively processes them to yield MIIs containing A and MIIs not containing A respectively.

- MIIs not containing A (itemsets obtained from T_{R_A})

  The lf-item in T_{R_A} is E. Therefore, similar to the first step, T_{R_A} is again divided into projected tree T_{P_E} and residual tree T_{R_E}, which are then recursively mined.

  ◇ MIIs not containing E (itemsets obtained from T_{R_E})
  Every 1-itemset is frequent. By recursively processing the residual and projected trees of B, $\{B, D\}$ is obtained as a MII. Itemset $\{C, D\}$ is also obtained as a potential MII. However, since it is frequent, it is not returned. Since $\{B, C\}$ is also frequent, only $\{B, D\}$ is returned from this tree.

  ◇ MIIs containing E (itemsets obtained from T_{P_E})
  All the 1-itemsets $\{B\}, \{C\}, \{D\}$ are infrequent (Step 6 of the algorithm). E is included with these itemsets to return $\{E, B\}, \{E, C\}, \{E, D\}$.

  T_{P_E} and T_{R_E} are mutually exclusive. Hence, the combined set $\{\{B, D\}, \{E, B\}, \{E, C\}, \{E, D\}\}$ forms the MIIs not containing A.

- MIIs containing A (itemsets obtained from T_{P_A})

  Item $\{E\}$ is infrequent ($support = 0$). Hence, $\{A, E\}\}$ forms a MII. Similarly, itemset $\{B, D\}$ with $support = 0$ is also obtained as a potential MII. The other MIIs obtained from recursive processing are $\{B, C\}, \{C, D\}$. The itemset $\{B, D\}$, however, appears as a MII in both T_{P_A} and T_{R_A}. Hence, it is removed (Step 16 of the algorithm). This avoids the inclusion of the itemset $\{A, B, D\}$ which is not a MII since $\{B, D\}$ is infrequent (as shown in T_{R_A}). A is included with the remaining set of MIIs from T_{P_A} to form the itemsets $\{A, B, C\}, \{A, C, D\}$.

  The combined set $\{\{A, B, C\}, \{A, C, D\}, \{A, E\}\}$ thus forms the MIIs containing A.

As mentioned in Step 3 of the algorithm, the infrequent 1-itemset $\{F\}$ is also included. Hence, in all, the algorithm returns the set $\{\{B, D\}, \{E, B\}, \{E, C\}, \{E, D\}, \{A, B, C\}, \{A, C, D\}, \{A, E\}, \{F\}\}$ as the MIIs for the database TD.

5.2 Completeness and Soundness

In this section, we prove formally the our algorithm is *complete* and *sound*, i.e., it returns all minimally infrequent itemsets and all itemsets returned by it are minimally infrequent.

Consider the lf-item x in the transaction database. MIIs in the database can be exhaustively divided into the following sets:

- Group 1: MIIs not containing x
  This can be again of two kinds:

  (a) Itemsets of length 1:
  These constitute the infrequent items in the database obtained from the initial pruning.

  (b) Itemsets of length greater than 1:
  These constitute the minimally infrequent itemsets obtained from the residual tree of x.

- Group 2: MIIs containing x
  This consists of itemsets of the form $\{x\} \cup S$ where S can be of following two kinds:

  (a) S occurs with x in at least one transaction:
  All items in S occur in the projected tree of x.

  (b) S does not occur with x in any transaction:
  Note that the path corresponding to S does not exist in the projected tree of x. Also, S is a 1-itemset. Assume the contrary, i.e., S contains two or more items. A subset of S would then exist, which would not occur with x in any transaction. As a result, the subset would also be infrequent and $\{x\} \cup S$ would not be qualified as a MII. Thus, S is a node that is absent in the projected tree of x.
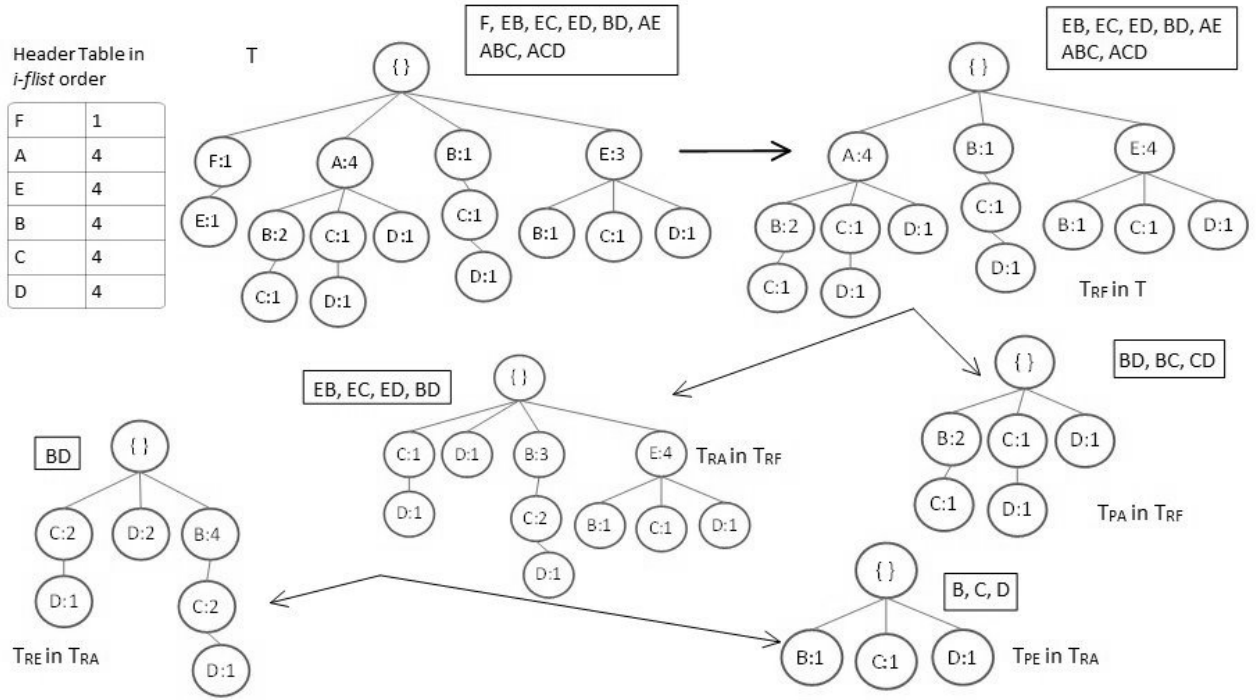
Figure 4: Running example for the transaction database in Table 1.

The following set of observations and theorems prove the correctness of the IFP_min algorithm. They exhaustively define and verify the set of itemsets returned by the algorithm.

The first observation relates the (in)frequent itemsets of the database to those present in the residual database.

Observation 1. *An itemset S (not containing the item x) is frequent (infrequent) in the residual tree T_{R_x} if and only if the itemset S is frequent (infrequent) in T, i.e.,*

$$S \text{ is frequent in } T \Leftrightarrow S \text{ is frequent in } T_{R_x} \ (x \notin S)$$
$$S \text{ is infrequent in } T \Leftrightarrow S \text{ is infrequent in } T_{R_x} \ (x \notin S)$$

Proof. Since S does not contain x, it does not occur in the projected tree of x. All occurrences of S must, therefore, be only in the residual tree T_{R_x}, i.e.,

$$supp(S, T) = supp(S, T_{R_x}) \quad [x \notin S] \quad (1)$$

Hence, S is frequent (infrequent) in T if and only if S is frequent (infrequent) in T_{R_x}. $\square$

The following theorem shows that the MIIs that do not contain the item x can be obtained directly as MIIs from the residual tree of x.

Theorem 1. *An itemset S (not containing the item x) is minimally infrequent in T if and only if the itemset S is minimally infrequent in the residual tree T_{R_x} of x, i.e.,*

$$S \text{ is minimally infrequent in } T$$
$$\Leftrightarrow S \text{ is minimally infrequent in } T_{R_x} \ (x \notin S)$$

Proof. Suppose S is minimally infrequent in T. Therefore, it is itself infrequent, but all its subsets $S' \subset S$ are frequent.

As S does not contain x, all occurrences of S or any of its subsets $S' \subset S$ must occur in the residual tree T_{R_x} only. Hence, using Observation 1, in T_{R_x} also, S is infrequent but all $S' \subset S$ are frequent. Therefore, S is minimally infrequent in T_{R_x} as well.

The converse is also true, i.e., if S is minimally infrequent in T_{R_x}, since all its occurrences are in T_{R_x} only, it is globally minimally infrequent (i.e., in T) as well.

$\square$

Theorem 1 guides the algorithm in mining MIIs not containing the least frequent item (lf-item) x. The algorithm makes use of this theorem recursively, by reporting MIIs of residual trees as MIIs of the original tree. For mining of MIIs that contain x, the following observation and theorem are presented.

The second observation relates the (in)frequent itemsets of the database to those present in the projected database.

Observation 2. *An itemset S is frequent (infrequent) in the projected tree T_{P_x} if and only if the itemset obtained by including x in S (i.e., $x \cup S$), is frequent (infrequent) in T, i.e.,*

$$x \cup S \text{ is frequent in } T \Leftrightarrow S \text{ is frequent in } T_{P_x}$$
$$x \cup S \text{ is infrequent in } T \Leftrightarrow S \text{ is infrequent in } T_{P_x}$$

Proof. Consider the itemset $x \cup S$. All occurrences of it are only in the projected tree T_{P_x}. The projected tree, however,

does not list x, and therefore, we have,

$$supp(x \cup S, T) = supp(x \cup S, T_{P_x}) = supp(S, T_{P_x}) \tag{2}$$

Hence, $x \cup S$ is frequent (infrequent) in T if and only if S is frequent (infrequent) in T_{P_x}. $\square$

The next theorem shows that the potential MIIs obtained from the projected tree of an item x (by appending x to it) is a MII provided it is not a MII in the corresponding residual tree of x.

Theorem 2. *An itemset $\{x\} \cup S$ is minimally infrequent in T if and only if the itemset S is minimally infrequent in the projected tree T_{P_x} but not minimally infrequent in the residual tree T_{R_x}, i.e.,*

$$\{x\} \cup S \text{ is minimally infrequent in } T$$
$$\Leftrightarrow S \text{ is minimally infrequent in } T_{P_x} \text{ and}$$
$$S \text{ is not minimally infrequent in } T_{R_x}$$

Proof. LHS to RHS:

Suppose $\{x\} \cup S$ is minimally infrequent in T. Therefore, it is itself infrequent, but all its subsets $S' \subset \{x\} \cup S$, including S, are frequent.

Since S is frequent and it does not contain x, using Observation 1, S is frequent in T_{R_x} and is, therefore, not minimally infrequent in T_{R_x}.

From Observation 2, S is infrequent in T_{P_x}. Assume that S is not minimally infrequent in T_{P_x}. Since, it is infrequent itself, there must exist a subset $S' \subset S$ which is infrequent in T_{P_x} as well. Now, consider the itemset $\{x\} \cup S'$. From Observation 2, it must be infrequent in T. However, since this is a subset of $\{x\} \cup S$, this contradicts the fact that $\{x\} \cup S$ is minimally infrequent. Therefore, the assumption that S is not minimally infrequent in T_{P_x} is false.

Together, it shows that if $\{x\} \cup S$ is minimally infrequent in T, S is minimally infrequent in T_{P_x} but not in T_{R_x}.

RHS to LHS:

Given that S is minimally infrequent in T_{P_x} but not in T_{R_x}, assume that $\{x\} \cup S$ is not minimally infrequent in T.

Since S is infrequent in T_{P_x}, using Observation 2, $\{x\} \cup S$ is also infrequent in T.

Now, since we have assumed that $\{x\} \cup S$ is not minimally infrequent in T, it must contain a subset $A \subset \{x\} \cup S$ which is infrequent in T as well.

Suppose A contains x, i.e., $x \in A$. Consider the itemset B such that $A = \{x\} \cup B$. Note that since $A \subset \{x\} \cup S$, $B \subset S$. Since $A = \{x\} \cup B$ is infrequent in T, from Observation 2, B is infrequent in T_{P_x}. However, since $B \subset S$, this contradicts the fact that S is minimally infrequent in T_{P_x}. Hence, A cannot contain x.

Therefore, it must be the case that $x \notin A$. To show that this leads to a contradiction as well, we first show that

Now, if S is minimally infrequent in T_{P_x} but not in T_{R_x} and $\{x\} \cup S$ is not minimally infrequent in T, then every subset $S' \subset S$ is frequent in T_{P_x}. Thus, $\forall S' \subset S, S'$ is frequent in T as well (since T_{P_x} is only a part of T). Then, if S

is infrequent, it must be a MII. However, from Theorem 1, it becomes a MII in T_{R_x} which is a contradiction. Therefore, S cannot be infrequent and is, therefore, frequent in T.

Since, $A \subset \{x\} \cup S$ but $x \notin A$, therefore, $A \subset S$. We have already shown that A is infrequent in T. Using the Apriori property, S cannot then be frequent. Hence, it contradicts the original assumption that $\{x\} \cup S$ is not minimally infrequent in T.

Together, we get that if S is minimally infrequent in T_{P_x} but not in T_{R_x}, then $\{x\} \cup S$ is minimally infrequent in T. $\square$

Theorem 2 guides the algorithm in mining MIIs containing the least frequent item (lf-item) x. The algorithm first obtains the MIIs from its projected tree and then removes those that are also found in the residual tree. It thus shows the connection between the two parts of the database, projected and residual.

5.3 Correctness

We now formally establish the correctness of the algorithm IFP_min by showing that the MIIs as enumerated in Group 1 and Group 2 in Section 5.2 are generated by it.

In Step 1, the algorithm first finds the infrequent items present in the tree. These 1-itemsets cover the Group 1(a).

Consider the least frequent item x. In Step 11 and Step 12, the tree T is divided into smaller trees, the residual tree T_{R_x} and the projected tree T_{P_x}.

In Step 13, Group 1(b) MIIs are obtained by the recursive application of IFP_min on the residual tree T_{R_x}. Theorem 1 proves that these are MIIs in the original dataset.

In Step 14, potential MIIs are obtained by the recursive application of IFP_min on the projected tree T_{P_x}. MIIs obtained from T_{R_x} are removed from this set. Combined with the item x, these form the MIIs enumerated as Group 2(a). Theorem 2 proves that these are indeed MIIs in the original dataset.

The projected database consist of all those transactions in which x is present. The Group 2(b) MIIs are of length 2 (as shown earlier). Thus, single items that are frequent but do not appear in the projected tree of x, when combined with x, constitute MIIs with support count of zero. These items appear in T_{R_x} though as they are frequent. Hence, they are obtained as single items that appear in T_{R_x} but not in T_{P_x} as shown in Step 17 of the algorithm.

The algorithm is, hence, complete and it exhaustively generates all minimally infrequent itemsets.

5.4 MIIs using Apriori

In this section, we show how the Apriori algorithm [1] can be improved to mine the MIIs. Consider the iteration where candidate itemsets of length $l + 1$ are generated from frequent itemsets of length l. From the generated candidate set, itemsets whose support satisfies the minimum support threshold are reported as frequent and the rest are rejected. This rejected set of itemsets constitute the MIIs of length

Algorithm 2 IFP_MLMS

Input: IFP-tree T with $\rho_T = p$
Output: Frequent* itemsets of T
1: **if** $T = \varnothing$ **then**
2:     **return** $\varnothing$
3: **end if**
4: $x \leftarrow$ ls-item in T
5: **if** $supp(\{x\}, T) < \sigma_{low}$ **then**
6:     $S_P \leftarrow \varnothing$
7: **else**
8:     $T_{P_x} \leftarrow$ projected tree of x
9:     $\rho_{T_{P_x}} \leftarrow p + 1$
10:     $S_P \leftarrow$ IFP_MLMS (T_{P_x})
11: **end if**
12: $T_{R_x} \leftarrow$ residual tree of x
13: $\rho_{T_{R_x}} \leftarrow p$
14: $S_R \leftarrow$ IFP_MLMS (T_{R_x})
15: **if** x is *frequent** in T **then**
16:     **return** $(\{x\} \bullet S_P) \cup S_R \cup \{x\}$
17: **else**
18:     **return** $(\{x\} \bullet S_P) \cup S_R$
19: **end if**

$l + 1$. This is attributed to the fact that for such an itemset, all the subsets are frequent (due to the candidate generation procedure) while the itemset itself is infrequent. For the experimentation purposes, we label this algorithm as the *Apriori_min* algorithm.

6 Frequent Itemsets in MLMS Model

In this section, we present our proposed algorithm *IFP_MLMS* to mine frequent itemsets in the MLMS framework. Though most applications use the constraint $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$ for the different support thresholds at different lengths, our algorithm (shown in Algorithm 2) does not depend on it and works for any general $\sigma_1, \ldots, \sigma_n$.

We use the lowest support $\sigma_{low} = \min_{\forall i} \sigma_i$ in the algorithm. The algorithm is based on the item with *least support* which we term as the *ls-item*.

IFP_MLMS is again based on the concepts of residual and projected trees. It mines the frequent itemsets by first dividing the database into projected and residual trees for the ls-item x, and then mining them recursively. We show that the frequent itemsets obtained from the residual tree are frequent itemsets in the original database as well.

The itemsets obtained from the projected tree share the ls-item as a prefix. Hence, the thresholds cannot be applied directly as the length of the itemset changes. The prefix accumulates as the algorithm goes deeper into recursion, and hence, a track of the prefix is maintained at each recursion level. At any stage of the recursive processing, if $supp(ls\text{-}item) < \sigma_{low}$, then this item cannot occur in a frequent itemset of any length (as any superset of it will not pass the support threshold). Thus, its sub-tree is pruned, thereby reducing the search space considerably.

To analyze the prefix items corresponding to a tree, the following two definitions are required:

Definition 2 (Prefix-set of a tree). *The* prefix-set *of a tree is the set of items that need to be included with the itemsets in the tree, i.e., all the items on which the projections have been done. For a tree T, it is denoted by Δ_T.*

Definition 3 (Prefix-length of a tree). *The* prefix-length *of a tree is the length of its prefix-set. For a tree T, it is denoted by ρ_T.*

For a tree T having $\Delta_T = S$ and $\rho_T = p$, the corresponding values for the residual tree are $\Delta_{T_{R_x}} = S$ and $\rho_{T_{R_x}} = p$ while those for the projected tree are $\Delta_{T_{P_x}} = \{x\} \cup S$ and $\rho_{T_{P_x}} = p + 1$. For the original transaction database TD and its tree T, $\Delta_T = \varnothing$ and $\rho_T = 0$.

For any k-itemset S in a tree T, the original itemset must include all the items in the prefix-set. Therefore, if $\rho_T = p$, for S to be frequent, it must satisfy the support threshold for $k + p$-length itemsets, i.e., $supp(X) \geq \sigma_{k+p}$ (henceforth, $\sigma_{k,p}$ is also used to denote σ_{k+p}). The definitions of frequent and infrequent itemsets in a tree T with a prefix-length $\rho_T = p$ are, thus, modified as follows.

Definition 4 (Frequent* itemset). *A k-itemset S is frequent* in T having $\rho_T = p$ if $supp(S, T) \geq \sigma_{k,p}$.*

Definition 5 (Infrequent* itemset). *A k-itemset S is infrequent* in T having $\rho_T = p$ if $supp(S, T) < \sigma_{k,p}$.*

Using these definitions, we explain Algorithm 2 along with an example shown in Figure 5 for the database in Table 2. The support thresholds are $\sigma_1 = 4$, $\sigma_2 = 4$, $\sigma_3 = 3$, $\sigma_4 = 2$, and $\sigma_5 = 1$.

The item with least support for the initial tree T is B. Since its support is above σ_{low} (Step 5), the algorithm extracts the projected tree T_{P_B} and then recursively processes it (Step 11 to Step 13).

The algorithm then processes the residual tree T_{R_B} recursively (Step 12 to Step 14). For that, it breaks it into the projected and residual trees of the ls-item there, which is A. Figure 5 shows all the frequent itemsets mined.

Since B itself is not frequent in T (Step 15), it is not returned. The other itemsets are returned (Step 18).

6.1 Correctness of IFP_MLMS

Let x be the ls-item in tree T (having $\rho_T = p$) and S be a *k-itemset* not containing x.

For computing the *frequent** itemsets for a tree T, the IFP_MLMS algorithm merges the *frequent** itemsets, obtained by the processing of the projected tree and residual tree of ls-item, using the following theorem

Theorem 3. *An itemset $\{x\} \cup S$ is frequent* in T if and only if S is frequent* in the projected tree T_{P_x} of x, i.e.,*

$$\{x\} \cup S \text{ is frequent* in } T \Leftrightarrow S \text{ is frequent* in } T_{P_x}$$

Proof. Suppose S is a k-itemset.
S is *frequent** in T_{P_x}
$\Leftrightarrow supp(S, T_{P_x}) \geq \sigma_{k,p+1}$ (since $\rho_{T_{P_x}} = p + 1$)
$\Leftrightarrow supp(\{x\} \cup S, T) \geq \sigma_{k,p+1}$ (using Observation 2)
$\Leftrightarrow supp(\{x\} \cup S, T) \geq \sigma_{k+1,p}$
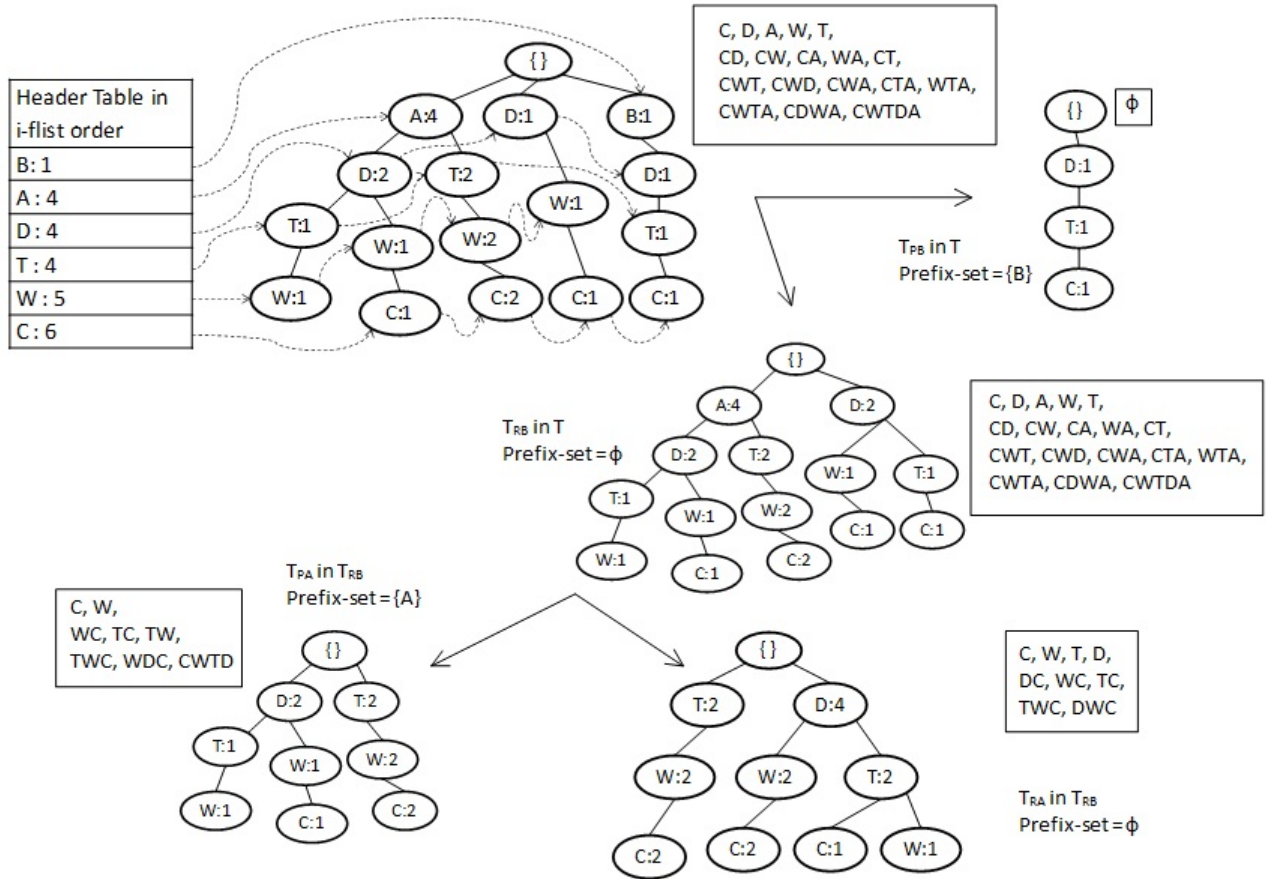$\Leftrightarrow \{x\} \cup S$ is *frequent** in T (since $\rho_T = p$). $\square$

Figure 5: Frequent itemsets generated from the database represented in Table 2 using the MLMS framework. The box associated with each tree represents the frequent* itemsets in that tree.

Theorem 4. *An itemset S (not containing x) is frequent* in T if and only if S frequent* in the residual tree T_{R_x} of x, i.e.,*

$$S \text{ is frequent* in } T \Leftrightarrow S \text{ is frequent* in } T_{R_x}$$

Proof. Suppose S is a k-itemset.
S is *frequent** in T_{R_x}
$\Leftrightarrow supp(S, T_{R_x}) \geq \sigma_{k,p}$ (since $\rho_{T_{R_x}} = p$)
$\Leftrightarrow supp(S, T) \geq \sigma_{k,p}$ (using Observation 1)
$\Leftrightarrow S$ is *frequent** in T (since $\rho_T = p$). $\qquad\square$

The algorithm IFP_MLMS merges the following *frequent** itemsets: (i) *frequent** itemsets obtained by including x with those returned from the projected tree (shown to be correct by Theorem 3), (ii) *frequent** itemsets obtained from the residual tree (shown to be correct by Theorem 4) and (iii) *1-itemset* $\{x\}$ if it is *frequent** in T. The root of the tree T that represents the entire database has a null prefix-set, and therefore, zero prefix-length. Hence, all *frequent** itemsets mined from that tree are the *frequent* itemsets in the original transaction database.

7 Experimental Results

In this section, we report the experimental results of running our algorithms on different datasets. We first report the performance of *IFP_min* algorithm in comparison with the *Apriori_min* and *MINIT* algorithms, followed by that of the *IFP_MLMS* algorithm. The benchmark datasets have been taken from Frequent Itemset Mining Implementations (FIMI) repository http://fimi.ua.ac.be/data/. All experiments were run on a machine with Dual Core Intel Processor running at 2.4GHz with 8GB of RAM.

7.1 IFP_min

The Accident dataset is characteristic of *dense* and *large* datasets. Figure 6 shows the performance of different algorithms on the dataset. The IFP_min algorithm outperforms the MINIT and Apriori_min algorithm by exponential factors. Apriori_min algorithm, due to its inherent property of performing worse than the pattern growth based algorithms on dense datasets, performs the worst.

The Connect (Figure 7), Mushroom (Figure 8) and Chess (Figure 9) datasets are characteristic of *dense* and *small* datasets. The Apriori_min algorithm achieves better reduction on the size of candidate sets. However, when
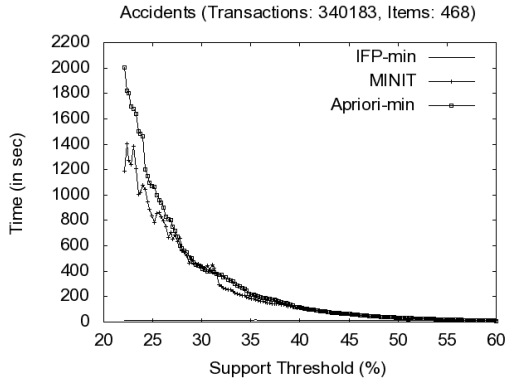
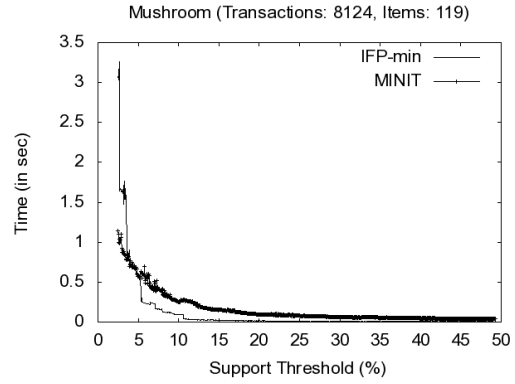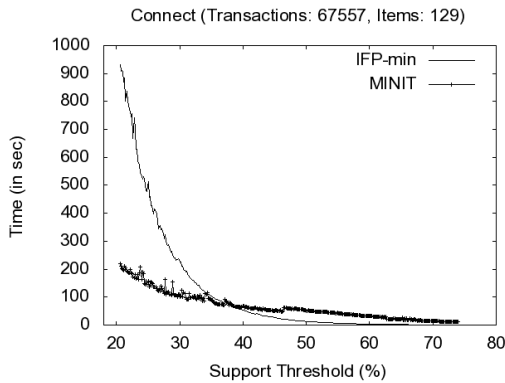Figure 6: Accident Dataset

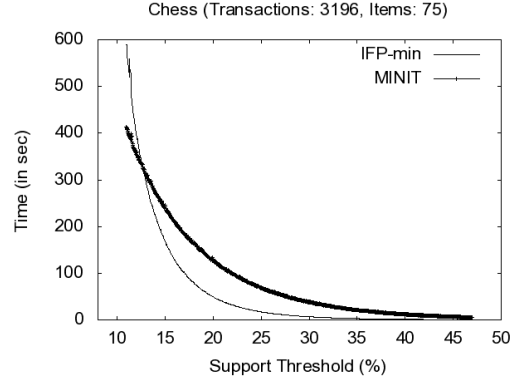Figure 8: Mushroom Dataset

Figure 7: Connect Dataset

Figure 9: Chess Dataset

there exist a large number of frequent itemsets, candidate generation-and-test methods may suffer from generating huge number of candidates and performing several scans of database for support-checking, thereby increasing the computational time. The corresponding computational times have not been shown in the interest of maintaining the scale of the graph present for IFP_min and MINIT.

As can be observed from the figures, dense and small datasets are characterized by a neutral support threshold below which the MINIT algorithm performs better than IFP_min and above which IFP_min performs better than MINIT. The MINIT algorithm prunes an item based on the support threshold and length of the itemset in which the item is present (*minimum support property* [7]). As the support thresholds are reduced, the pruning condition becomes activated and leads to reduction in search space. Above the neutral point, the pruning condition is not effective. In IFP_min algorithm, any candidate MII itemset is checked for set membership in a residual database whereas in MINIT the candidates are validated by computing the support from the whole database. Due to reduced validation space, IFP_min outperforms MINIT.

The T10I4D100K (Figure 10) and T40I10D100K (Figure 11) are *sparse* datasets. Since Apriori_min is a candidate-generation-and-test based algorithm, it halts

when all the candidates are infrequent. As such, it avoids the complete traversal of the database for all possible lengths. However, both IFP_min and MINIT, being based on the recursive elimination procedure, have to complete their full run in order to report the MIIs. This results in higher computational times for these methods.

On analyzing and comparing the MIIs generated by IFP_min, Apriori_min and MINIT algorithms, we found that the MIIs belonging to Group 2b (i.e., the itemsets having zero support threshold in the transaction database) are not reported by the MINIT algorithm, thereby leading to its incompleteness. Based on the experimental analysis, it is observed that for large dense datasets, it is preferable to use IFP_min algorithm. For small dense datasets, MINIT should be used at low support thresholds and IFP_min should be used at larger thresholds. For sparse datasets, Apriori_min should be used for reporting MIIs.

7.2 IFP_MLMS

In this section, we report the performance of IFP_MLMS algorithm in comparison with the Apriori_MLMS [5] algorithm. Several real and synthetic datasets (obtained from http://archive.ics.uci.edu/ml/datasets) have been used for testing the performance of the algorithms. For dense datasets, due to the presence of large number of transac-
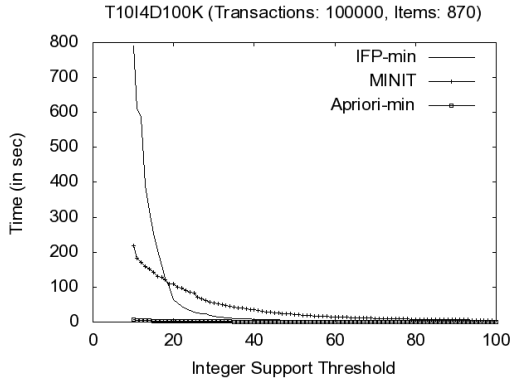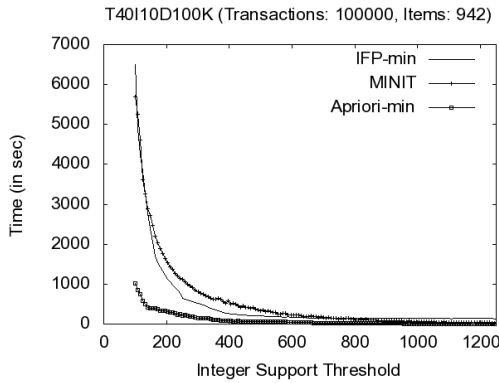
Figure 10: T10I4D100K Dataset
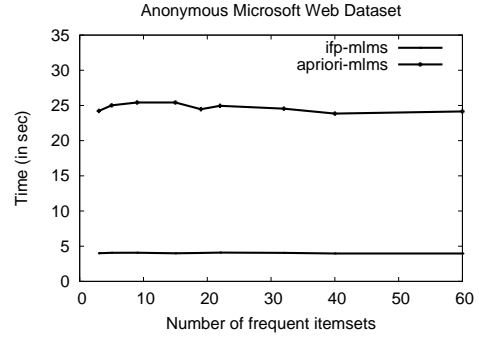

Figure 11: T40I10D100K Dataset

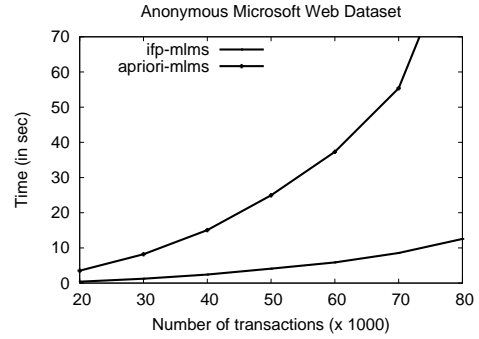Figure 12: The IFP_MLMS and Apriori_MLMS algorithms on the Anonymous Microsoft Web Dataset.

Figure 13: The IFP_MLMS and Apriori_MLMS algorithms on the Anonymous Microsoft Web Dataset.

tions and items, the Apriori_MLMS algorithm crashes for lack of memory space.

The first dataset we used is the Anonymous Microsoft Web dataset that records areas of www.microsoft. com each user visited in a one week time frame in February 1998 (http://archive.ics.uci.edu/ml/datasets/ Anonymous+Microsoft+Web+Data). The dataset consists of 1,31,666 transactions and 294 attributes.

Figure 12 plots the performance analysis of IFP_MLMS and Apriori_MLMS algorithms. The minimum support thresholds for itemsets of different lengths were varied over a distribution window from 2% to 20% at regular intervals. The graph clearly shows the superiority of IFP_MLMS as compared to Apriori_MLMS. We also observe that the time taken by the algorithm to compute the frequent itemsets is roughly independent of the minimum support thresholds for different lengths.

For the plot shown in Figure 13, the number of transactions are varied in the Anonymous Microsoft Web dataset. We observe that the running time for both the algorithms increases with the number of transactions when the support thresholds are kept between 3% to 10%. However, the rate of increase in time for Apriori_MLMS algorithm is much higher and at 80,000 transactions, Apriori_MLMS crashes due to lack of memory.

Since the Apriori_MLMS fails to give results for large datasets, smaller datasets were obtained from http: //archive.ics.uci.edu/ml/datasets/ for comparison purposes with the IFP_MLMS. The characteristics of these datasets are shown in Table 3. For each such dataset, the corresponding time for IFP_MLMS and Apriori_MLMS are plotted in Figure 14. The support threshold percentages are kept the same for all the datasets and were varied between 10% and 60% for the different length itemsets.

The above results clearly show that the IFP_MLMS algorithm outperforms Apriori_MLMS. During experimentation, we found the running time of both IFP_MLMS and Apriori_MLMS algorithms to be independent of support thresholds for the MLMS model. This behavior is attributed to the absence of downward closure property [1] of frequent itemsets in the MLMS model, unlike that of the single threshold model.

Further, consider an alternative FP-Growth algorithm for the MLMS model that mines all frequent itemsets corresponding the lowest support threshold σ_{low} and then filters the σ_k frequent k-itemsets to report the frequent itemsets. In this case, a very large set of frequent itemsets is generated that renders the filtering process computationally expensive. The pruning based on σ_{low} in IFP_MLMS ensures that the search space is same for both the algorithms. Moreover, the filtering required for the former is implicitly
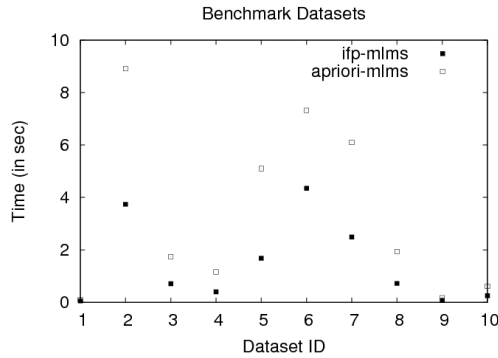
Figure 14: The IFP_MLMS and Apriori_MLMS algorithms on the datasets given in Table 4.

ID	Dataset	Number of items	Number of transactions
1	machine	467	209
2	vowel_context	4188	990
3	abalone	3000	4177
4	audiology	311	202
5	anneal	191	798
6	cloud	2057	11539
7	housing	2922	506
8	forest_fires	1039	518
9	nursery	28	12961
10	yeast	1568	1484

Table 4: Details of smaller datasets.

performed in IFP_MLMS, thus making IFP_MLMS more efficient.

8 Conclusion

In this paper, we have introduced a novel algorithm, IFP_min, for mining minimally infrequent itemsets (MIIs). To the best of our knowledge, this is the first paper that addresses this problem using the pattern-growth paradigm. We have also proposed an improvement of the Apriori algorithm to find the MIIs. The existing algorithms are evaluated on dense as well as sparse datasets. Experimental results show that: (i) for large dense datasets, it is preferable to use IFP_min algorithm, (ii) for small dense datasets, MINIT should be used at low support thresholds and IFP_min should be used at larger thresholds and (iii) for sparse datasets, Apriori_min should be used for reporting the MIIs.

We have also designed an extension of the algorithm for finding frequent itemsets in the multiple level minimum support (MLMS) model. Experimental results show that this algorithm, IFP_MLMS, outperforms the existing candidate-generation-and-test based Apriori_MLMS algorithm.

In future, we plan to utilize the scalable properties of our algorithm to mine maximally frequent itemsets. It will be also useful to do a performance analysis of IFP-tree in par-

allel architecture as well as extend the IFP_min algorithm across different models for itemset mining, including interestingness measures.

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *VLDB*, pages 487–499, 1994.

[2] K. Beyer and R. Ramakrishnan. Bottom-up computation of sparse and iceberg cube. *SIGMOD Rec.*, 28:359–370, 1999.

[3] X. Dong, Z. Niu, X. Shi, X. Zhang, and D. Zhu. Mining both positive and negative association rules from frequent and infrequent itemsets. In *ADMA*, pages 122–133, 2007.

[4] X. Dong, Z. Niu, D. Zhu, Z. Zheng, and Q. Jia. Mining interesting infrequent and frequent itemsets based on mlms model. In *ADMA*, pages 444–451, 2008.

[5] X. Dong, Z. Zheng, and Z. Niu. Mining infrequent itemsets based on multiple level minimum supports. In *ICICIC*, page 528, 2007.

[6] G. Grahne and J. Zhu. Fast algorithms for frequent itemset mining using fp-trees. *Trans. Know. Data Engg.*, 17(10):1347–1362, 2005.

[7] D. J. Haglin and A. M. Manning. On minimal infrequent itemset mining. In *DMIN*, pages 141–147, 2007.

[8] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *SIGMOD*, pages 1–12, 2000.

[9] A. M. Manning and D. J. Haglin. A new algorithm for finding minimal sample uniques for use in statistical disclosure assessment. In *ICDM*, pages 290–297, 2005.

[10] A. M. Manning, D. J. Haglin, and J. A. Keane. A recursive search algorithm for statistical disclosure assessment. *Data Mining Know. Discov.*, 16:165–196, 2008.

[11] J. Srivastava and R. Cooley. Web usage mining: Discovery and applications of usage patterns from web data. *SIGKDD Explorations*, 1:12–23, 2000.

[12] X. Wu, C. Zhang, and S. Zhang. Efficient mining of both positive and negative association rules. *ACM Trans. Inf. Syst.*, 22(3):381–405, 2004.