Novel Techniques to Reduce Search Space in Multiple Minimum Supports-Based Frequent Pattern Mining Algorithms

R. Uday Kiran
International Institute of Information Technology-Hyderabad
Hyderabad
Andhra Pradesh, India
uday_rage@research.iiit.ac.in

P. Krishna Reddy
International Institute of Information Technology-Hyderabad
Hyderabad
Andhra Pradesh, India
pkreddy@iiit.ac.in

ABSTRACT

Frequent patterns are an important class of regularities that exist in a transaction database. Certain frequent patterns with low minimum support (*minsup*) value can provide useful information in many real-world applications. However, extraction of these frequent patterns with single *minsup*-based frequent pattern mining algorithms such as Apriori and FP-growth leads to "rare item problem." That is, at high *minsup* value, the frequent patterns with low *minsup* are missed, and at low *minsup* value, the number of frequent patterns explodes. In the literature, "multiple *minsups* framework" was proposed to discover frequent patterns. Furthermore, frequent pattern mining techniques such as Multiple Support Apriori and Conditional Frequent Pattern-growth (CFP-growth) algorithms have been proposed. As the frequent patterns mined with this framework do not satisfy *downward closure property*, the algorithms follow different types of pruning techniques to reduce the search space. In this paper, we propose an efficient CFP-growth algorithm by proposing new pruning techniques. Experimental results show that the proposed pruning techniques are effective.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications - Data Mining

General Terms

Algorithms

Keywords

Data mining, knowledge discovery, frequent patterns and multiple minimum supports.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

EDBT 2011, March 22–24, 2011, Uppsala, Sweden.

Copyright 2011 ACM 978-1-4503-0528-0/11/0003 ...$10.00

1. INTRODUCTION

Since the introduction of frequent patterns in [1], the problem of mining frequent patterns from the transaction databases has been actively studied in the literature [4]. Most of the frequent pattern mining algorithms (e.g., Apriori [2] and FP-growth [5]) use "single minimum support (*minsup*) framework" to discover complete set of frequent patterns. *Minsup* controls the minimum number of transactions a pattern must cover in a database. The frequent patterns discovered with this framework satisfy *downward closure property*. That is, "all non-empty subsets of a frequent pattern must also be frequent." This property holds the key for minimizing the search space in all of the single *minsup*-based frequent pattern mining algorithms [2, 4].

Most of the real-world databases are non-uniform in nature containing both frequent and rare items. A rare item is an item having low frequency. Frequent patterns containing rare items can provide useful information to the users.

> **Example 1:** In a supermarket, costly goods such as *Bed* and *Pillow* are less frequently purchased than the cheaper goods such as *Bread* and *Jam*. However, the association between the former set of items can be more interesting as it may generate relatively more revenue.

However, mining frequent patterns containing both frequent and rare items with "single *minsup* framework" leads to the *rare item problem* which is as follows: *At high minsup, the frequent patterns containing rare items will be missed, and at low minsup, combinatorial explosion can occur, producing too many frequent patterns.*

To confront *rare item problem*, an effort has been made in [10] to find frequent patterns with "multiple *minsups* framework." In this framework, each pattern can satisfy a different *minsup* depending upon the items within it. The frequent patterns discovered through "multiple *minsups* framework" do not satisfy *downward closure property*. As a result, this property cannot be used for minimizing the search space in multiple *minsups*-based frequent pattern mining algorithms.

In the literature, an Apriori-like algorithm known as Multiple Support Apriori (MSApriori) was proposed to find frequent patterns with "multiple *minsups* framework" [10]. Also, an FP-growth-like algorithm known as Conditional Frequent Pattern-growth (CFP-growth) has been proposed to mine frequent patterns [6]. Since *downward closure property* no

longer holds in "multiple *minsups* framework," the CFP-growth algorithm has to carry out exhaustive search in the constructed $Tree$ structure. In this paper we propose an improved CFP-growth algorithm, called CFP-growth++, by introducing four pruning techniques to reduce the search space. Experimental results on various types of datasets show that the proposed algorithm is efficient and scalable as well.

1.1 Related Work

The occurrence of *rare item problem* with the usage of traditional data mining techniques to discover knowledge involving rare items was introduced in [13]. In [10], "multiple *minsups* framework" has been introduced to address *rare item problem*, and MSApriori algorithm was proposed for extracting frequent patterns. An FP-growth-like algorithm [5], called CFP-growth [6], has been proposed to mine frequent patterns. It was shown that the performance of CFP-growth is better than the MSApriori algorithm. In [14], a new interestingness measure, called *relative support* has been introduced, and an Apriori-like algorithm has been proposed for mining frequent patterns containing both frequent and rare items. An Apriori-like approach which tries to use a different *minsup* at each level of iteration has been discussed in [11]. A stochastic mixture model based on negative binomial distribution has been discussed to mine rare association rules [3]. An approach has been suggested to mine the association rules by considering only infrequent items i.e., items having support less than the *minsup* [16].

We have been investigating improved approaches to mine frequent patterns containing both frequent and rare items. In [8], an improved methodology has been proposed to specify items' MIS values. In [9], a new interestingness measure, called *item-to-pattern difference*, has been used along with the "multiple *minsups* framework" to discover frequent patterns in the databases, where frequencies of the items' vary widely. An effort has been made to extend the notion of multiple constraints to extract periodic-frequent patterns [12]. In [7], we have proposed a preliminary algorithm to improve the performance of CFP-growth by suggesting two pruning techniques for reducing the size of constructed *tree* structure. It is to be noted that the algorithm discussed in [7] performs exhaustive search, like CFP-growth, to discover complete set of frequent patterns as the frequent patterns mined with "multiple *minsups* framework" do not satisfy *downward closure property*.

In this paper, we investigated approaches to reduce the search space while extracting frequent patterns and proposed two additional pruning techniques which significantly reduces the search space by avoiding exhaustive search while extracting frequent patterns from a *tree* structure. Overall, we have proposed a comprehensive algorithm by employing four pruning techniques to efficiently mine frequent patterns.

1.2 Paper Organization

The remaining part of the paper is organized as follows. In Section 2, we explain the necessary background. In Section 3, we discuss the CFP-growth algorithm and its performance issues. In Section 4, we discuss the proposed pruning techniques to reduce the search space and present the CFP-growth++ algorithm. Experimental results are discussed in Section 5. The last section contains conclusions and future work.

2. BACKGROUND

In this section, we explain the basic model of frequent patterns, *rare item problem* and the extended model of frequent patterns based on multiple *minsups*.

2.1 Basic Model of Frequent Patterns

Frequent patterns were first introduced in [1]. The basic model of frequent patterns is as follows:

Let $I = \{i_1, i_2, \cdots, i_n\}$ be a set of items, and a transaction database $DB = \langle T_1, T_2, \cdots, T_n \rangle$, where T_i ($i \in [1..n]$) is a transaction which contains a set of items in I. Each transaction is associated with an identifier, called TID. The *support* of a **pattern** (or an itemset) X, denoted as $S(X)$, is the number transactions containing X in DB. The pattern X can be *frequent* if its support is no less than a user-defined minimum support (*minsup*) threshold value, i.e., $S(X) \geq minsup$. A pattern containing k number of items is a k-pattern. The support of a pattern can also be represented in percentage of $|DB|$. In this paper, we use the terms "itemset" and "pattern" interchangeably.

Example 2: Consider the transaction database of 20 transactions shown in Table 1. The set of items $I = \{a, b, c, d, e, f, g, h\}$. The set of a and b, i.e., $\{a, b\}$ is a pattern. It is a 2-pattern. For simplicity, we write this pattern as 'ab'. It occurs in *tids* of $1, 4, 7, 10, 11, 13, 16$ and 19. Therefore, the support of ab, $S(ab) = 8$. If the user-specified $minsup = 6$, then ab is a frequent pattern because $S(ab) \geq minsup$.

Table 1: A transaction database.

TID	Items	TID	Items
1	a, b	11	a, b
2	a, e, f	12	a, c
3	c, d	13	a, b
4	a, b, h	14	b, e, f, g
5	c, d	15	c, d
6	a, c	16	a, b, d
7	a, b	17	c, d
8	e, f	18	a, c
9	c, d, g	19	a, b, e
10	a, b	20	c, d

Apriori [1] and FP-growth [5] are the two popular algorithms to mine frequent patterns. Apriori uses candidate-generate-and-test-approach to discover the complete set of frequent patterns. FP-growth employs pattern-growth technique to discover complete set of frequent patterns. In the literature, it has been shown that FP-growth performs better than Apriori [5].

2.2 Rare Item Problem

Real-world databases are mostly non-uniform in nature containing both frequent and relatively infrequent (or rare) items. If the items' frequencies in a database vary widely, we encounter the following issues while mining frequent patterns under single *minsup* framework:

i. If *minsup* is set too high, we will miss the frequent patterns containing rare items.

ii. To find frequent patterns that involve both frequent and rare items, we have to set low *minsup*. However,

this may cause combinatorial explosion, producing too many frequent patterns, because those frequent items will combine with one another in all possible ways and many of them are meaningless.

This dilemma is called the *rare item problem*.

Example 3: Consider the database shown in Table 1. At high *minsup*, say $minsup = 6$, we will miss the frequent patterns involving the rare items 'e' and 'f'. To mine the frequent patterns containing 'e' and 'f', we have to specify low *minsup* value. Let the *minsup* value be 3. The frequent patterns discovered at $minsup = 3$ are shown in the fourth column of the Table 2. Among the generated frequent patterns, the pattern 'ac' can be considered uninteresting to the user because it has low support and contains frequently occurring items 'a' and 'c'. This pattern can be considered interesting if it has satisfied high *minsup*, say $minsup = 6$.

Table 2: Frequent patterns generated at $minsup = 3$. The terms "S", "MIS", "SMF" and "MMF" are respectively used as the acronyms to denote support, minimum item support, "single *minsup* framework" and "multiple *minsups* framework." The terms "T" and "F" respectively denote the frequent patterns generated and have not generated in single and multiple *minsups* frameworks.

Patterns	S	MIS	SMF	MMF
a	12	10	T	T
c	9	10	T	F
b	9	8	T	T
d	7	6	T	T
e	4	3	T	T
f	3	3	T	T
ab	8	-	T	T
ac	3	-	T	F
cd	6	-	T	T
ef	3	-	T	T

2.3 Extended Model of Frequent Patterns

To confront the *rare item problem*, an effort has been made in the literature to extend the basic model of frequent patterns to multiple *minsups* [10]. In the extended model, each item in the transaction database is specified with a support constraint known as *minimum item support* (MIS) and *minsup* of a pattern is represented with the minimal MIS value among all its items (see Equation 1).

$$minsup(X) = minimum \begin{pmatrix} MIS(i_1), & MIS(i_2), \\ \cdots, & MIS(i_k) \end{pmatrix} \quad (1)$$

where, $X = \{i_1, i_2, \cdots, i_k\}$, $1 \leq k \leq n$, is a pattern and $MIS(i_j)$, $1 \leq j \leq k$, represents the MIS of an item $i_j \in X$.

The extended model enables the user to simultaneously specify high *minsup* for a pattern containing only frequent items and low *minsup* for a pattern containing rare items. Thus, efficiently addressing the *rare item problem*. The significance of this model is illustrated in Example 4.

Example 4: Continuing with Example 3, let the user-specified MIS values for the items 'a', 'b', 'c', 'd', 'e', 'f', 'g' and 'h' be 10, 8, 10, 6, 3, 3, 3 and 2, respectively. The items' MIS values are specified with respect to their support values. The frequent patterns discovered with the extended model are shown in the fifth column of Table 2. It can be observed that the uninteresting frequent pattern 'ac' that was generated at low *minsup* (i.e., at $minsup = 3$) in Example 3 has failed to be a frequent pattern in this model. It is because $S(ac) < minsup$ ($= minimum(MIS(a), MIS(c))$.

3. CFP-GROWTH AND PERFORMANCE ISSUES

In [10], an Apriori-like algorithm known as Multiple Support Apriori (MSApriori) has been discussed to mine frequent patterns. An FP-growth-like algorithm known as Conditional Frequent Pattern-growth (CFP-growth) has been discussed to efficiently mine frequent patterns [6]. Among the two algorithms, it has been shown that CFP-growth algorithm performs better than MSApriori algorithm. In this section we discuss CFP-growth and its performance issues.

3.1 CFP-growth

The CFP-growth algorithm is developed based on the FP-growth algorithm [6]. Even though it is an FP-growth-like algorithm, the structure, construction and mining procedures of CFP-growth are different from FP-growth. The CFP-growth algorithm accepts transaction database and MIS values of items as an input. Using the items' MIS values as prior knowledge, it discovers complete set of frequent patterns with a **single scan** on the transaction database. Briefly, the working of CFP-growth is as follows.

i. Items are sorted in descending order of their MIS values. Using the sorted list of items, an FP-tree-like structure known as MIS-tree is constructed with a single scan on the transaction database. Simultaneously, the support of each item in the MIS-tree is measured.

ii. To reduce the search space, tree-pruning operation is performed to prune the items that cannot generate any frequent pattern. The criterion used is **prune the items that have support less than the lowest MIS value among all items.**

Table 3: MIS and support values of items.

Items	a	b	c	d	e	f	g	h
MIS	10	8	10	6	3	3	3	2
Support	12	9	9	7	4	3	2	1

Example 5: Table 3 provides information about the MIS and support values of items present in the database of Table 1. The lowest MIS value among all items is 2. Therefore, it is clear that no pattern will have *minsup* less than 2. Based on *apriori property* [2], it turns out that 'h' and its supersets cannot generate any frequent pattern as their supports will be no more than 1. So, CFP-growth prunes 'h' from the MIS-tree.

iii. After tree-pruning operation, tree-merging operation is performed on the MIS-tree to merge the child nodes of a parent node that share same item. The resultant MIS-tree is called *compact MIS-tree*.

iv. Finally, choosing each item in the *compact MIS-tree* as the suffix item (or pattern), its *conditional pattern base* (i.e., prefix sub-paths) is build to discover complete set of frequent patterns. Since frequent patterns do not satisfy *downward closure property*, CFP-growth tries to discover complete set of frequent patterns by building suffix patterns until its respective *conditional pattern base* is empty.

Example 6: Consider the *compact MIS-tree* shown in Figure 2(b). For the (suffix) item 'f', the conditional prefix paths are $\langle a, e : 1\rangle$, $\langle e : 1\rangle$ and $\langle b, e : 1\rangle$. The CFP-growth algorithm builds the suffix patterns $\langle f\rangle$, $\langle f, e\rangle$, $\langle f, b\rangle$, $\langle f, a\rangle$, $\langle f, e, b\rangle$ and $\langle f, e, a\rangle$ to discover the complete set of frequent patterns.

3.2 Performance Issues

The performance issues of CFP-growth algorithm are as follows.

First, the criterion used by CFP-growth to construct *compact MIS-tree* still considers some items which cannot generate any frequent pattern at higher-order.

Example 7: Continuing with Example 5, CFP-growth constructs *compact MIS-tree* with the items 'a', 'b', 'c', 'd', 'e', 'f' and 'g'. However, item 'g' cannot generate any frequent pattern at higher-order because its support (i.e., 2) is less than the lowest MIS value (i.e., 3) among all the items 'a', 'b', 'c', 'd', 'e', 'f' and 'g'.

Second, as CFP-growth continues to build suffix patterns until its respective *conditional pattern base* is empty, CFP-growth searches in some of those (infrequent) suffix patterns which will never generate any higher-order frequent pattern.

Example 8: Continuing with Example 6, the lowest MIS value among the items 'a', 'b', 'e' and 'f' is 3 ($=MIS(f)$). Since the support of 'a' and 'b' in the *conditional pattern base* of 'f' is less than 3, it is straight forward to prove that $\{f,\ a\}$ and $\{f,\ b\}$ cannot be frequent patterns. In addition, their supersets also cannot be frequent patterns. Thus, CFP-growth spends additional resources (i.e., runtime) to discover the complete set of frequent patterns.

4. PROPOSED APPROACH

In this section, we first introduce the properties and theorems that have been identified for reducing the search space. Next, we explain the pruning techniques to reduce the search space and present the algorithm.

4.1 Theorems

The pruning techniques that are proposed for reducing the search space in the "multiple *minsups* framework" are based on *apriori property* (see Property 1) and Theorems 4.1 and 4.2.

PROPERTY 1. *(Apriori property.) In a database DB, if X and Y are two patterns such that $X \subseteq Y$, then $S(X) \geq S(Y)$.*

THEOREM 4.1. *In every frequent pattern, the item having lowest MIS value is a frequent item.*

PROOF. Consider a transaction database DB containing the set of items, $I = \{i_1, i_2, \cdots, i_n\}$, such that $MIS(i_1) \geq MIS(i_2) \geq \cdots \geq MIS(i_n)$. Let $X = \{i_j, \cdots, i_k\} \subseteq I$, where $1 \leq j \leq k \leq n$, be a pattern. If X is frequent, then $S(X) \geq minimum(MIS(i_j), \cdots, MIS(i_k))$. That is, $S(X) \geq MIS(i_k)$. From Property 1, it turns out that $S(i_k) \geq S(X) \geq MIS(i_k)$. Thus, if X is frequent, then i_k is a frequent item. □

THEOREM 4.2. *In every frequent pattern, all non-empty subsets containing the item having lowest MIS value will be frequent.*

PROOF. Consider a transaction database DB containing the set of items, $I = \{i_1, i_2, \cdots, i_n\}$. Let $MIS(i_j)$, where $i_j \in I$, be the user-specified MIS values such that $MIS(i_1) \geq MIS(i_2) \geq \cdots \geq MIS(i_n)$. Let $X = \{i_j, \cdots, i_k\} \subseteq I$, where $1 \leq j \leq k \leq n$, be a frequent pattern. That is, $S(X) \geq MIS(i_k)$ $(= minimum(MIS(i_j), \cdots, MIS(i_k)))$. Let $A \subset X$ be a pattern such that $i_k \in A$. Since i_k has the lowest MIS value among all items in X, it turns out that $minsup(A) = MIS(i_k)$. From Property 1, it can be derived that $S(A) \geq S(X) \geq MIS(i_k)$. Thus, A is a frequent pattern. □

4.2 Techniques to Reduce the Search Space

We propose four techniques to reduce the search space.

4.2.1 Least minimum support

In the *multiple minsups framework*, each pattern can satisfy a different *minsup* depending upon the items within it. The term *least minimum support* (LMS) refers to the lowest *minsup* of all frequent patterns. Since frequent item is a frequent 1-pattern, it is straight forward to prove from Theorem 4.1 that LMS is always equal to the lowest MIS value among all frequent items. LMS has the following two properties.

PROPERTY 2. *If $X = \{i_1, i_2, \cdots, i_k\} \subseteq I$, where $1 \leq k \leq n$, is a pattern such that $S(X) < LMS$, then $S(X) < minimum(MIS(i_1), MIS(i_2), \cdots, MIS(i_k))$.*

PROPERTY 3. *If X and Y are two patterns such that $X \subset Y$ and $S(X) < LMS$, then $S(Y) < LMS$.*

These two properties facilitate to use LMS as a constraint to reduce the search space. In particular, LMS can be used to prune the items (or patterns) that cannot generate any frequent pattern at higher-order. The significance of LMS is illustrated in Example 9.

Example 9: Continuing with Example 5, the frequent items in the transaction database of Table 1 are 'a', 'b', 'd', 'e' and 'f'. Based on Theorem 4.1, it can be said that any frequent pattern that is mined from this database will have one of the above items as the item having lowest MIS value. Thus, lowest *minsup* that can be satisfied by a frequent pattern is lowest MIS value among all these frequent items i.e., 3. Since, the items 'g' and 'h' have support less than 3, their supersets also cannot have support greater than 3 (Property 1). Thus, it is guaranteed that 'g' and 'h' cannot generate any frequent pattern at higher-order.

4.2.2 Conditional Minimum Support

Let $Tree$ be the FP-tree-like structure constructed after scanning a database in MIS descending order of items. If we consider an item i_j that exists in $Tree$ as a suffix item (or 1-pattern) and construct its prefix sub-paths (i.e., *conditional pattern base*), then MIS of i_j will be the lowest MIS value among all the items in the *conditional pattern base*. From the definition of *minsup* in *multiple minsups framework*, it turns out that any frequent pattern that is going to be generated from the *conditional pattern base* of i_j should satisfy MIS value of i_j. Thus, we call the MIS value of the suffix item i_j as the **conditional** *minsup*. The correctness of this idea is shown in Lemma 4.3.

LEMMA 4.3. *Let α be a pattern in MIS-tree and S_α be the support of α. Also, let $minsup_\alpha$ be the minsup that α has to satisfy, B be α's conditional pattern base, and β be an item in B. The support of β in B be $S_B(\beta)$ and MIS_β be the β's MIS value. The minsup of pattern $\langle\alpha,\beta\rangle$ is $minsup_\alpha$.*

PROOF. According to the definition of MIS-tree, $MIS_B(\beta)$ will always be greater than or equal to the $minsup_\alpha$. Therefore, *minsup* of $\langle\alpha,\beta\rangle$ is $minsup_\alpha$. □

Example 10: Consider the *compact MIS-tree* shown in Figure 2(b). For the (suffix) item f, the conditional prefix paths are ⟨a, e: 1⟩, ⟨e: 1⟩ and ⟨b, e: 1⟩. The item having lowest MIS value among all the items 'a', 'b', 'e' and 'f' is 'f' which is the suffix item. As a result, every frequent pattern that gets generated from the *conditional pattern base* of 'f' will have $minsup = MIS(f)$. Thus, $MIS(f)$ is considered as conditional *minsup* for mining frequent patterns from the conditional pattern base of the suffix item 'f'.

4.2.3 Conditional Closure Property

PROPERTY 4. *(Conditional Closure property.) If a suffix pattern is infrequent, then all its super-suffix patterns (i.e., suffix pattern along with other item(s) in its* conditional pattern base*) will also be infrequent.*

The correctness of this property is shown in Lemma 4.4.

LEMMA 4.4. *Let α be a pattern in MIS-tree and S_α be the support of α. Also, let $minsup_\alpha$ be the minsup that α has to satisfy, B be α's conditional pattern base, and β be an item in B. The support of β in B be $S_B(\beta)$ and MIS_β be the β's MIS value. If α is infrequent, then the pattern $\langle\alpha,\beta\rangle$ is also infrequent.*

PROOF. According to the definition of *conditional pattern base* and MIS-tree, each subset in B occurs under the condition of the occurrence of α in the transaction database. If an item β appears in B for n times, it appears with α in n times. From the definition of frequent pattern used in the *minimum constraint model*, the *minsup* of $\langle\alpha,\beta\rangle$ is $minimum(minsup_\alpha, MIS_\beta) = minsup_\alpha$. As $S_\alpha < minsup_\alpha$, the $S_{\langle\alpha,\beta\rangle} < minsup_\alpha$ (apriori property [1]). Therefore, $\langle\alpha,\beta\rangle$ is also infrequent. □

4.2.4 Infrequent leaf node pruning

The leaf nodes of a $Tree$ that belong to infrequent items can be pruned without missing any frequent pattern or changing the support of a frequent pattern. We call this pruning technique as "infrequent leaf node pruning." It is straight forward to prove from the "conditional *minsup*" and *conditional closure property* that the *conditional pattern base* of a suffix item that is infrequent will not result in any frequent pattern.

4.3 CFP-growth++

The proposed CFP-growth++ algorithm is an improvement over CFP-growth algorithm. It successfully addresses the above two issues of CFP-growth. The differences between CFP-growth and CFP-growth++ are as follows:

i. The CFP-growth++ employs a better criterion to identify the items that cannot generate any frequent pattern. This criterion enables CFP-growth++ to construct *compact MIS-tree* with only those items that can generate frequent patterns.

ii. The proposed algorithm will not search for frequent patterns until the *conditional pattern base* of a suffix pattern is empty. Instead, it tries to identify which suffix patterns can generate frequent patterns at higher order and perform search only in them.

The CFP-growth++ algorithm accepts transaction database DB, set of items I and items' MIS values as the input parameters. Using the items' MIS values as the prior knowledge, CFP-growth++ discovers the complete set of frequent patterns with a single scan on the transaction database. The steps involved in CFP-growth++ are as follows: (i) construction of MIS-tree (ii) generating *compact MIS-tree* and (iii) mining frequent patterns from the compact MIS-tree.

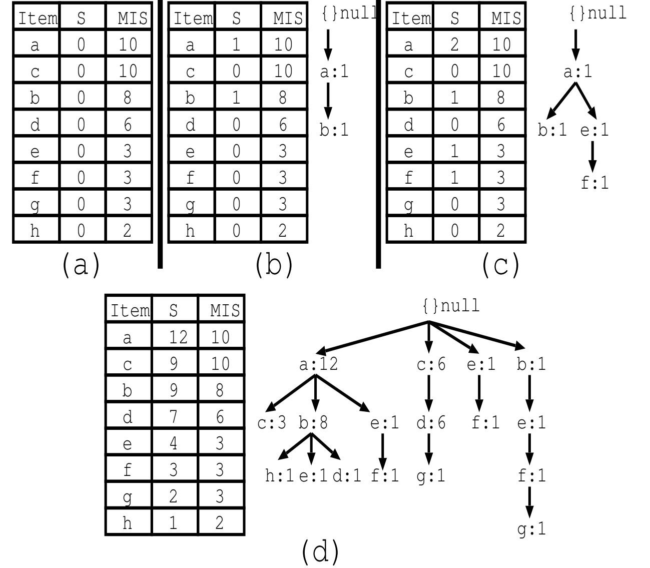

Figure 1: Construction of MIS-tree. (a) Initial MIS-list (b) After scanning first transaction (c) After scanning second transaction and (d) After scanning every transaction.

4.3.1 Construction of MIS-tree

The MIS-tree consists of two components: MIS-list and prefix-tree. The MIS-list is a list having three fields -

Algorithm 1 MIS-tree (DB: transaction database, I: itemset containing n items, MIS: *minimum item support* values for n items)

1: Let L represent the set of items sorted in decreasing order of their MIS values.
2: In L order, insert items into the MIS-list with $S = 0$ and *mis* equivalent to the respective MIS value.
3: Create the root of a MIS-tree, T, and label it as "null".
4: **for** each transaction $t \in DB$ **do**
5: Sort all the items in t in L order.
6: Count the support values of any item i, denoted as $S(i)$ in t.
7: Let the sorted items in t be $[p|P]$, where p is the first element and P is the remaining list. Call $InsertTree([p|P], T)$.
8: **end for**
9: Let $j = n - 1$.
10: **for** $(; j \geq 0; j = j - 1)$ **do**
11: **if** $(S[i_j] < MIS[i_j])$ **then**
12: Delete the item i_j in header table.
13: Call $MisPruning(Tree, i_j)$.
14: **else**
15: $LMS = MIS[i_j]$.
16: break.
17: **end if**
18: **end for**
19: **for** $(; j \geq 0; j = j - 1)$ **do**
20: **if** $(S[i_j] < LMS)$ **then**
21: Delete the item i_j in header table.
22: Call $MisPruning(Tree, i_j)$.
23: **end if**
24: **end for**
25: Name the resulting table as *MinFrequentItemHeaderTable*.
26: Call $MisMerge(Tree)$.
27: Call $InfrequentLeafNodePruning(Tree)$.

item name (*item*), support (S) and *minimum item support* (MIS). The structure of the prefix-tree in MIS-tree is same as that in FP-tree [5]. However, the difference is that items in the prefix-tree of FP-tree are arranged in descending order of their support values, whereas items in the prefix-tree of MIS-tree are arranged in descending order of their MIS values. To facilitate tree-traversal, node-links are maintained in the MIS-tree as in FP-tree.

The construction of MIS-tree in CFP-growth++ algorithm is shown in Algorithm 1. We illustrate this algorithm by using the transaction database shown in Table 1. Let the user-specified MIS values for the items '*a*', '*b*', '*c*', '*d*', '*e*', '*f*', '*g*' and '*h*' be 10, 8, 10, 7, 3, 3, 3 and 2, respectively.

The items are sorted in descending order of their MIS values. Let this sorted list of items be L. Thus, $L = \{a, c, b, d, e, f, g, h\}$ (Step 1 of Algorithm 1). In L order, insert each item into the MIS-list with support equal to zero and MIS equivalent to their respective MIS value (Step 2 of Algorithm 1). The resultant MIS-list is shown in Figure 1(a).

A MIS-tree is then created as follows. First, a root node is created with label "null." The database DB is scanned. The items in each transaction are processed in L order, and a branch is created for each transaction as in FP-growth [5]. Simultaneously, we increment the support values of the respective items in the MIS-list by 1 (Lines 4 to 8 in Algorithm 1 and Procedure 2). For example, the scan of the first transaction, "1: a,b" which contains two items ⟨a, b in L order⟩, leads to the construction of the first branch of the tree with two nodes ⟨a: 1⟩ and ⟨b: 1⟩, where '*a*' is linked as a child of the root and '*b*' is linked as the child node of '*a*'. Next, we increment support values of '*a*' and '*b*' in the MIS-list by 1. The MIS-tree generated after scanning the first transaction is shown in Figure 1(b). The second transaction containing the items '*a*', '*e*' and '*f*' in L order will result in a branch where '*a*' is linked to *root*, '*e*' is linked to '*a*' and '*f*' is linked to '*e*'. However, this branch would share a common prefix, '*a*', with the existing path for 1. Therefore, we instead increment the count of '*a*' node by 1, and create new nodes, ⟨e: 1⟩ and ⟨f: 1⟩, where '*e*' is linked to '*a*' and '*f*' is linked to '*e*'. In the MIS-list, the support of the items '*a*', '*e*' and '*f*' are incremented by 1. The resultant MIS-tree is shown in Figure 1(c). Similar process is repeated for the remaining transactions and MIS-tree is updated accordingly. The resultant MIS-tree after scanning every transaction in the transaction database is shown in Figure 1(d). For the simplicity of figures, we do not show the node traversal pointers in trees, however, they are maintained as in the construction process of FP-tree.

Procedure 2 InsertTree $([p|P], T)$

1: **if** T has a child node N such that p.item-name=N.item-name **then**
2: Increment N's count by 1.
3: **else**
4: Create a new node N, and let its count be 1.
5: Let its parent link be linked to T.
6: Let its node-link be linked to the nodes with the same item-name via the node-link structure.
7: **end if**
8: **if** P is nonempty **then**
9: Call $InsertTree(P, N)$.
10: **end if**

Procedure 3 MisPruning $(Tree, i_j)$

1: **for** each node in the node-link of i_j in $Tree$ **do**
2: **if** the node is a leaf **then**
3: Remove the node directly.
4: **else**
5: Remove the node and then its parent node will be linked to its child node(s).
6: **end if**
7: **end for**

4.3.2 Construction of compact MIS-tree

The *compact MIS-tree* is generated by pruning those items from the MIS-tree that cannot generate any frequent pattern. The pruning techniques, LMS and *infrequent leaf node pruning*, are used in this process. The procedure used for constructing *compact MIS-tree* is as follows.

The MIS-tree is constructed with every item in the transaction database. To decrease the search space, we use LMS as a constraint to prune the items that cannot generate any frequent pattern. A method to prune such items from the MIS-tree is as follows.

i. Starting from the last item of the MIS-list, the items

Procedure 4 MisMerge ($Tree$)

1: **for** each item i_j in the MinFrequentItemHeaderTable **do**
2: **if** there are child nodes with the same item-name then **then**
3: Merge the nodes and set the count as the summation of these nodes' counts.
4: **end if**
5: **end for**

Procedure 5 InfrequentLeafNodePruning($Tree$)

1: Choose the last but one item i_j in *MinFrequentItemHeaderTable*. That is, item having second lowest MIS value.
2: **repeat**
3: **if** i_j item is infrequent **then**
4: Using node-links parse all nodes of i_j in $Tree$.
5: **repeat**
6: **if** i_j node is the leaf of a branch **then**
7: Drop the node-link connecting through the child branch.
8: Create a new node-link from the node in the previous branch to node in the coming branch.
9: Drop the leaf node in the branch.
10: **end if**
11: **until** all the nodes of i_j in $Tree$ are parsed
12: **end if**
13: Choose item i_j which is next in the order.
14: **until** all items in *MinFrequentItemHeaderTable* are completed

that have support less than their respective MIS value are pruned (Lines 9 to 13 in Algorithm 1 and Procedure 3).

ii. Once the frequent item is found, its MIS value is chosen as the LMS value (Lines 14 to 18 in Algorithm 1). Next, support of the remaining items in the MIS-list are compared with LMS value, and those items that have support less than LMS are pruned from the MIS-tree (Lines 19 to 24 in Algorithm 1 and Procedure 3).

We explain the construction of compact MIS-tree by considering the MIS-tree shown in Figure 1(d). The process starts from 'h' as it is the last item in the MIS-list of MIS-tree. This item is an infrequent item, therefore, it is pruned from the MIS-tree. Among the remaining items in the MIS-list, 'g' is the last item in the MIS-list. It is also an infrequent item, therefore, it is pruned from the MIS-tree. Now, the last item in the MIS-list is 'f'. It is a frequent item. Hence, no tree-pruning operation is carried for the item 'f.' Next, using the MIS of the item 'f' as the LMS value, the supports of the remaining items in the MIS-tree are compared. As these items have support greater than or equal to LMS value, the tree-pruning operation ends.

After tree-pruning, tree-merging process is carried out to merge the child nodes of a parent node that share a common item (line 26 in Algorithm 1 and Procedure 4). The resultant MIS-tree is called *compact MIS-tree*. The *compact MIS-tree* generated after tree-pruning and tree-merging operations is shown in Figure 2(a). The process of *infrequent leaf node*

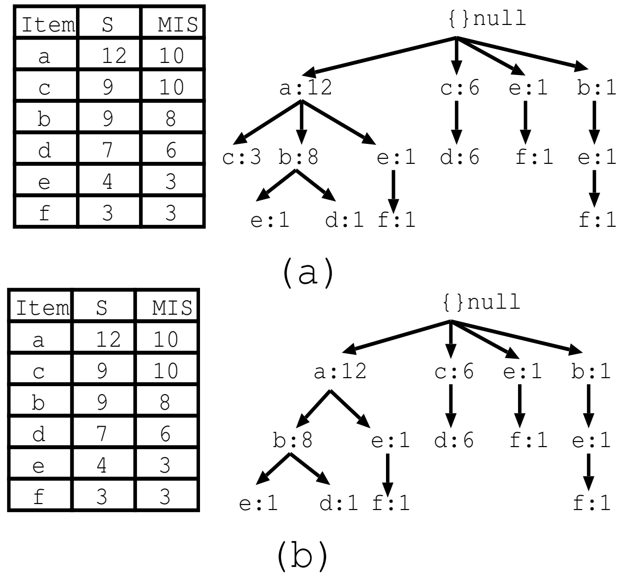

Figure 2: Compact MIS-tree. (a)After pruning items g and h and (b) After infrequent item leaf node pruning.

pruning (line 27 in Algorithm 1 and Procedure 5) is carried on the *compact MIS-tree* to decrease its size. The process is as follows. Among the remaining items in the MIS-list of MIS-tree, 'c' is an infrequent item (i.e., its support is less than the required *minsup* value). Therefore, using the node-links of 'c', we collect all the branches containing 'c'. The branches containing c are {{a, c: 3}, {c, d: 6}}. In the branch ⟨a, c: 3⟩, c is the leaf node, therefore, we prune the node 'c' in this branch. The resultant MIS-tree is shown in Figure 2(b). Note that the node of 'c' in the branch {{c, d: 6}} it not pruned as it is not a leaf node. (Pruning 'c' in this branch will result in missing the frequent pattern 'cd' because $S(cd) \geq minimum(MIS(c),\ MIS(d))$.

Algorithm 6 CFP-growth++ ($Tree$: *compact MIS-tree*)

1: **for** each item i in the header of the $Tree$ **do**
2: Set conditional *minsup*, $Cminsup_i = MIS(i)$.
3: **if** i is a frequent item **then**
4: Generate pattern $\beta = i \cup \alpha$ with $support = i.support$.
5: Construct β's conditional pattern base and β's *conditional MIS-tree* $Tree_\beta$.
6: **if** $Tree_\beta \neq \emptyset$ **then**
7: Call $CFPGrowth++(Tree_\beta, \beta, Cminsup_i)$.
8: **end if**
9: **end if**
10: **end for**

4.3.3 *Mining frequent patterns from* compact MIS-tree

The procedure for mining frequent patterns from *compact MIS-tree* is shown in Algorithm 6. The pruning techniques *conditional minsup* and *conditional closure property* are for mining frequent patterns.

The process of mining frequent patterns from the *compact MIS-tree* of Figure 2(b) is shown in Table 4 and is described

Procedure 7 CFPGrowth++$(Tree, \alpha, Cminsup_i)$

1: **for** each i in the header of $Tree$ **do**
2: Generate pattern $\beta = i \cup \alpha$ with $support = i.support$.
3: Construct β's *conditional pattern base* and then β's *conditional MIS-tree* $Tree_\beta$.
4: **if** $Tree_\beta \neq \emptyset$ **then**
5: **if** $Tree_\beta$ contains a single path P **then**
6: **for** each combination (denoted as γ) of the nodes in the path P **do**
7: Generate pattern $\gamma \cup \beta$ with $support$ = minimum support count of nodes in γ.
8: **end for**
9: **else**
10: Call $CFPGrowth++(Tree_\beta, \beta, Cminsup_i)$.
11: **end if**
12: **end if**
13: **end for**

as follows. Consider the item 'f' that has lowest MIS among all items in the *compact MIS-tree*. It occurs in 3 branches of *compact MIS-tree*. The branches are ⟨a, e, f: 1⟩, ⟨e, f: 1⟩ and ⟨b, e, f: 1⟩. Considering 'f' as a suffix pattern (or item), its conditional prefix paths are ⟨a, e: 1⟩, ⟨e: 1⟩ and ⟨b, e: 1⟩, which form its conditional pattern base. As the compact MIS-tree is constructed in MIS descending order of items, 'f' (suffix item) will have lowest MIS value among all the items in its conditional pattern base. Therefore, using MIS value of the item 'f' (i.e., 3) as the conditional *minsup*, conditional MIS-tree is generated with ⟨e: 3⟩; the items a and b are not included because the support counts are less than the specified conditional *minsup* value (i.e., *conditional closure property*). The single path generates the frequent pattern {e, f: 3}. Similar process is repeated for other remaining items in the *compact MIS-tree* to discover the complete set of frequent patterns.

Table 4: Mining *compact MIS-tree* by using multiple *minsups* and conditional pattern bases. The terms 'SI', 'MS' and 'Cond.' respectively denote 'suffix item', 'conditional *minsup*' and 'Conditional'.

SI	MS	Cond. pattern bases	Cond. MIS-tree	frequent patterns
f	3	⟨a, e: 1⟩ ⟨e: 1⟩ ⟨b, e: 1⟩	⟨e: 3⟩	{{e, f : 3}}
e	3	⟨a, b: 1⟩ ⟨a: 1⟩ ⟨b: 1⟩	-	-
d	6	⟨a, b: 1⟩ ⟨c: 6⟩	⟨c: 6⟩	{{c, d: 6}}
b	8	⟨a: 8⟩	⟨a: 8⟩	{{a, b: 8}}
c	10	-	-	-

5. EXPERIMENTAL RESULTS

In this section, we evaluate the performance of FP-growth, CFP-growth and CFP-growth++ algorithms. We are not considering Apriori and MSApriori algorithms for comparison because it has been shown that FP-growth and CFP-growth algorithms are better than the corresponding Apriori and MSApriori algorithms, respectively [5, 6].

The algorithms are written in GNU C++ and run with Ubuntu 10.04 operating system on a 2.66 GHz machine with 1GB memory. The experiments are pursued on synthetic (*T10I4D100K*) and real-world datasets (*BMS-WebView-1* [15], *Mushroom* and *Kosarak*). $T10I4D100K$, *BMS-WebView-1* and *Kosarak* are sparse datasets and *Mushroom* is a dense dataset. These datasets are widely used in the literature for evaluating the performance of data mining algorithms. The datasets are available at Frequent Itemset Mining repository (http://fimi.cs.helsinki.fi/data/). The details of the datasets are shown in Table 5.

Table 5: Dataset characteristics. The terms "max," "avg," and "trans" respectively denote maximum, average and transactions.

Dataset	Transactions	Distinct Items	Max. Trans. Size	Avg. Trans. Size
T10I4D100k	100000	870	29	10.102
BMS-WebView-1	59602	497	267	2.5
Mushroom	8124	119	23	23
Kosarak	990002	41270	2498	8.1

In the experiment, we used the methodology discussed in [10] to assign items' MIS values. The methodology is as follows:

$$MIS(i_j) \quad = \quad maximum(\beta \times f(i_j), LS) \qquad (2)$$

The $f(i_j)$ and $MIS(i_j)$ variables respectively denote the frequency (or support) and *minimum item support* for an item $i_j \in I$. The variable LS represents the user-specified least minimum item support allowed. In this, $\beta \in [0, 1]$ is a parameter that controls how the MIS values for items should be related to their frequencies. If $\beta = 0$, we have only one minimum support, LS, which is the same as the *minsup* in traditional frequent pattern mining. If $\beta = 1$ and $f(i_j) \geq LS$, then $MIS(i_j) = f(i_j)$.

5.1 Experiment 1

In this experiment, both LS and *minsup* values are set at 0.1% for *T10I4D100K* and *BMS-WebView-1* datasets. For the *Mushroom* dataset, both LS and *minsup* values are set at 10% as it is a dense dataset. To show how β affects the number of frequent patterns found and the performance of the algorithms, we fixed $\beta = \frac{1}{\alpha}$ and varied α. In the sparse datasets (*T10I4D100k* and *BMS-WebView-1*), α is varied from 1 to 20. In the dense dataset (*Mushroom*), α is varied from 1 to 5.

The experimental results regarding how the number of frequent patterns vary with the MIS values in different datasets are shown in the Figures 3(a), 3(b) and 3(c). When α becomes larger, the number of frequent patterns found by the method gets closer to the number of frequent patterns found with the single *minsup* framework. The reason is as follows. At higher values of α, the items' MIS values become equals to LS. As a result, the performance of the "multiple *minsups* framework" is same as the "single *minsup* framework" with $minsup = LS$. It can also be observed that the above phenomenon happens at higher values of α in the sparse datasets (Figure 3(a) and 3(b)) and at lower values of α in the dense dataset (Figure 3(c)).

The experimental results about the runtime performance of FP-growth, CFP-growth and CFP-growth++ algorithms

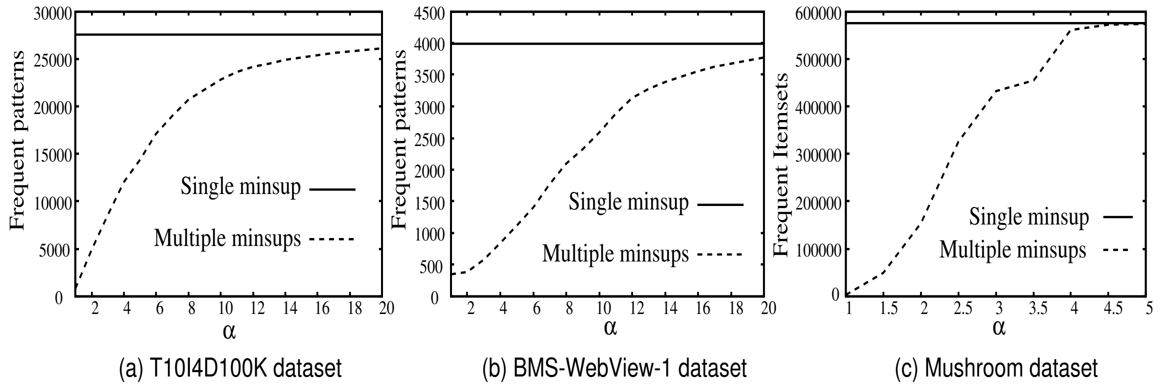

Figure 3: Frequent patterns generated at different MIS values of items.

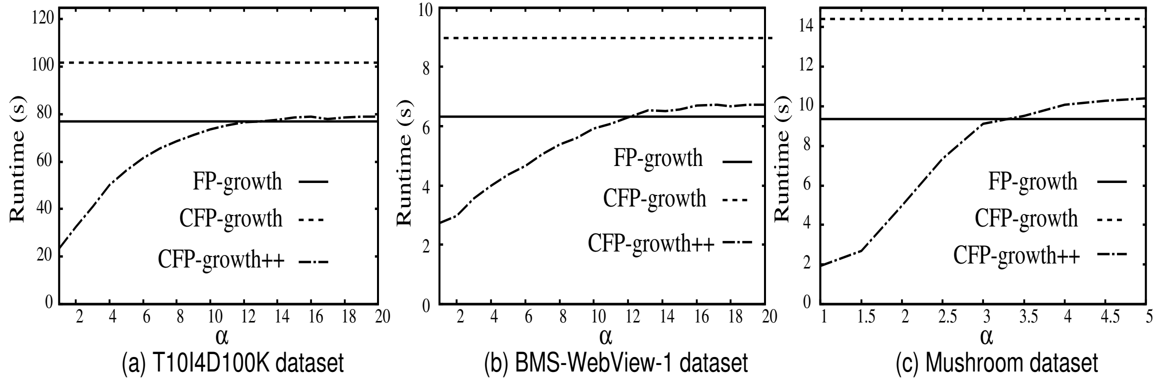

Figure 4: Runtime taken for generating frequent patterns.

on different datasets are shown in the Figures 4(a), 4(b) and 4(c). For CFP-growth and CFP-growth++ algorithms, runtime includes the construction of MIS-tree, construction of *compact MIS-tree* and mining frequent patterns from *compact MIS-tree*. For FP-growth algorithm, runtime includes the construction of FP-tree and mining frequent patterns from the FP-tree.

It can be observed that the runtime performance of the FP-growth is significantly better than the CFP-growth for all datasets independent of the α value. It is due to the fact that CFP-growth carries out exhaustive search as "multiple *minsups* framework" does not satisfy *downward closure property*, whereas FP-growth exploits *downward closure property* to reduce the search space. It can be observed that at all α values, the CFP-growth++ performance is significantly better than the CFP-growth. The performance gap is much higher at lower α values. It is due to the effect of pruning techniques employed by CFP-growth++ to reduce the search space.

It can also be observed that at higher values of α, the runtime consumed by the CFP-growth++ is more than the FP-growth. The difference is more clear in Figures 4(b) and 4(c). The reason is as follows. The CFP-growth++ algorithm has to consider MIS value of the suffix item to specify conditional *minsup* value for its *conditional pattern base*. Whereas, the FP-growth algorithm simply specifies a constant *minsup* for a *conditional pattern base* independent of the suffix item.

In [10], it was mentioned that in many real-world applications, the frequent patterns generated when $\alpha = 4$ were interesting to the users. It can be observed that, at $\alpha = 4$, the proposed CFP-growth++ improves the runtime performance significantly over CFP-growth.

5.2 Experiment 2

In this experiment, we evaluate the scalability performance of CFP-growth and CFP-growth++ algorithms on execution time by varying the number of transactions in a database. We use real-world *kosarak* dataset for the scalability experiment, since it is a huge sparse dataset. We divided the dataset into five portions of 0.2 million transactions in each part and investigated runtime taken by CFP-growth and CFP-growth++ algorithms after accumulating each portion with previous parts. For each experiment, we have fixed $\beta = 0.25\%$ and $LS = 1\%$. The experimental result is shown in Figure 5. It can be observed from the graph that as the database size increases, the runtime of both CFP-growth and CFP-growth++ algorithms increases. However, it can be noted that CFP-growth++ is more scalable than the CFP-growth algorithm. Overall, CFP-growth++ is about an order of magnitude faster than the CFP-growth in large databases, and this gap grows wider with the increase in dataset size.

6. CONCLUSIONS

To mine frequent patterns containing both frequent and rare items, "multiple *minsups* framework" was proposed in the literature. By considering "multiple *minsups* framework," CFP-growth algorithm has been proposed to extract

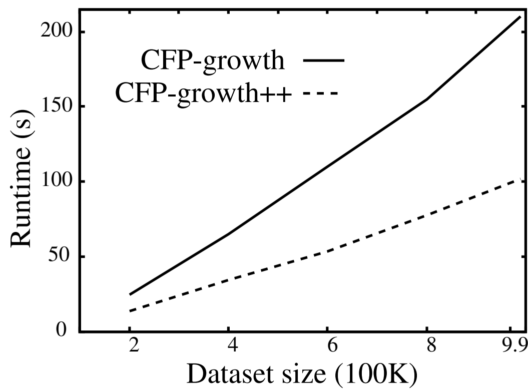

Figure 5: Scalability with number of transactions.

frequent patterns. In this paper, we have proposed an improved CFP-growth algorithm, called CFP-growth++, by introducing the following pruning techniques: *least minimum support*, *conditional minsup*, *conditional closure property* and i*nfrequent leaf node pruning*. By conducting experiments on both synthetic and real-world datasets, we have shown that the proposed algorithm improves the performance significantly over the exciting approaches.

As a part of future work, we are planning to conduct extensive experiments by considering different types of datasets. It is interesting to investigate how the proposed pruning techniques can be extended to improve the performance of generalized multiple-level frequent patterns.

7. REFERENCES

[1] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. *SIGMOD Rec.*, 22:207–216, June 1993.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499, 1994.

[3] M. Hahsler. A model-based frequency constraint for mining associations from transaction data. *Data Min. Knowl. Discov.*, 13(2):137–166, 2006.

[4] J. Han, H. Cheng, D. Xin, and X. Yan. Frequent pattern mining: current status and future directions. *Data Min. Knowl. Discov.*, 15(1):55–86, 2007.

[5] J. Han, J. Pei, Y. Yin, and R. Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Min. Knowl. Discov.*, 8(1):53–87, 2004.

[6] Y.-H. Hu and Y.-L. Chen. Mining association rules with multiple minimum supports: a new mining algorithm and a support tuning mechanism. *Decis. Support Syst.*, 42(1):1–24, 2006.

[7] R. U. Kiran and P. K. Reddy. An improved frequent pattern-growth approach to discover rare association rules. In *Proceedings of the International Conference on Knowledge Discovery and Information Retrieval*, pages 43–52, 2009.

[8] R. U. Kiran and P. K. Reddy. An improved multiple minimum support based approach to mine rare association rules. In *IEEE Symposium on Computational Intelligence and Data Mining*, pages 340–347, 2009.

[9] R. U. Kiran and P. K. Reddy. Mining rare association rules in the datasets with widely varying items' frequencies. In *DASFAA (1)*, pages 49–62, 2010.

[10] B. Liu, W. Hsu, and Y. Ma. Mining association rules with multiple minimum supports. In *KDD '99: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 337–341. ACM, 1999.

[11] C. S. K. Selvi and A. Tamilarasi. Mining association rules with dynamic and collective support thresholds. *International Journal on Open Problems Computational Mathematics*, 2(3):427–438, 2009.

[12] R. Uday Kiran and P. Krishna Reddy. Towards efficient mining of periodic-frequent patterns in transactional databases. In *Database and Expert Systems Applications*, volume 6262 of *Lecture Notes in Computer Science*, pages 194–208. Springer, 2010.

[13] G. M. Weiss. Mining with rarity: a unifying framework. *SIGKDD Explor. Newsl.*, 6(1):7–19, 2004.

[14] H. Yun, D. Ha, B. Hwang, and K. H. Ryu. Mining association rules on significant rare data using relative support. *J. Syst. Softw.*, 67:181–191, September 2003.

[15] Z. Zheng, R. Kohavi, and L. Mason. Real world performance of association rule algorithms. In *KDD '01: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 401–406. ACM, 2001.

[16] L. Zhou and S. Yau. Association rule and quantitative association rule mining among infrequent items. In *International Workshop on Multimedia Data Mining*, 2007.